

# Cluster monitoring at NCSA

T. Roney, A. Bailey & J. Fullop  
*National Center for Supercomputing Applications*  
*University of Illinois, USA.*

## Abstract

System monitoring and performance monitoring at the National Center for Supercomputing Applications have always been treated as separate arenas, due to the nature of their different functions. The function of a system monitor has been that of a virtual system operator, monitoring for events that depict a system fault and also automating the management of such events. The function of a performance monitor has been to collect data from different layers of the system -- hardware, kernel, service, and application layers -- for administrators and users who collect such data for diagnostic purposes in order to finely tune the performance of the different system layers. The collected performance data has never been associated with the system-monitoring framework. However, the legacy of this division of interest dissolves within the new arena of cluster monitoring. This paper explains why this is so, and how the National Center for Supercomputing Applications has merged the arenas of system monitoring and performance monitoring in a cluster environment.

## 1 Introduction

System monitoring at the National Center for Supercomputing Applications (NCSA) has played an integral role in the reliability and integrity of its computational environment. Performance monitoring has never before played an operational role, as its function has typically been commissioned to the users' domain, to enhance application performance. But, now that Linux clusters have become a computational platform at NCSA, system monitoring has become a performance issue.

To understand the performance issue, we can look at how a single monitor performs its task on a 512-processor supercomputer, and compare this to how it performs the same task on a 512-processor clustered system. The 512-processor supercomputer runs one copy of the monitoring software, while the 512-processor clustered system of 256 dual-processor hosts must run 256 copies of the monitoring software to perform the same monitoring task. Monitoring a clustered system can require hundreds or even thousands of times the number of monitoring processes required by the traditional supercomputer.

A performance monitor, however, is designed with performance in mind, minimizing its own perturbing effect on system-performance measurements. This is necessary because, while a system monitor might run many times per hour, a performance monitor might run many times per minute. And, since performance becomes an issue when monitoring for possible system events, the performance monitor becomes a good match for the task of system monitoring on a clustered system. The purpose of this paper is to describe the way in which NCSA incorporates a performance monitor into its general system-monitoring environment in order to perform the task of event management on a clustered system. Figure 1 illustrates this description in graphical form.

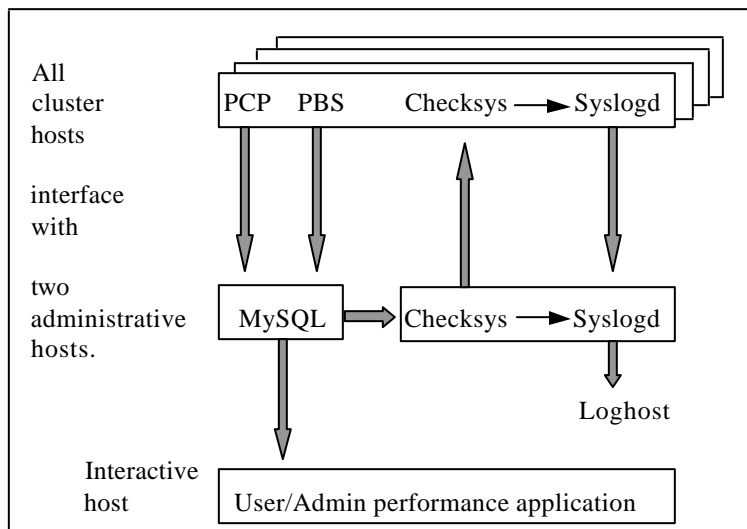


Figure 1: A graphical illustration of the monitoring schema used to monitor a clustered system, depicting the general subject matter of this paper.

Section 2 of this paper explains NCSA's general system-monitoring environment, mostly involving Checksys, which is an event monitor as well as an event manager. (Please note that the terms event monitor and system monitor will be used interchangeably whenever one will more likely than the other promote a better sense and a better flow of words.) Section 3 describes how Checksys is made to operate only on an as-needed basis, and only as an event manager, no longer as an event monitor, for a clustered system. Section 4 delineates the details of a web-based cluster-performance monitor, built by interfacing Performance Co-Pilot (PCP) [1] and Portable Batch System (PBS) [2] with MySQL [3], and also describes how the system monitor is dependent upon the performance monitor.

## **2 NCSA's General System Monitoring Environment**

All monitored systems at NCSA forward system messages via the process syslogd to a loghost for pattern matching, which is performed by Swatch [4]. Each system also runs a system monitor called Checksys. All messages from Swatch and Checksys are displayed on a web-based interface, which is an NCSA-modified version of Netsaint [5].

### **2.1 The Swatch Monitor**

The syslogd on every monitored host at NCSA is configured to forward its system messages to a central loghost, at which point all messages are monitored by Swatch. Swatch is configured to look for certain patterns generated at the hardware, kernel, service and application layers, and to take action when a pattern is found.

Hosts within a cluster, rather than forwarding their system messages on to the central loghost, forward their messages on to a cluster-admin host, which is also a host within the cluster, but designated to perform administrative duties only. Swatch runs on the admin host, monitoring the system messages coming in from all other clustered hosts. As can be seen in Figure 1, the admin host might also filter and forward certain messages on to the central loghost.

Currently, there are only two actions taken by Swatch when a pattern within a system message is matched. One of the actions records the message, which will be archived. The other action informs the Netsaint monitor (explained below) to display the message for NCSA's Technology Management Group, who staff a helpdesk 24 hours a day, seven days a week.

### **2.2 The Netsaint Display**

The Netsaint monitor is an event monitor in its own right. NCSA is using it

in a modified way, for its web-based display feature only. This display feature of the monitoring schema is not included in Figure 1. The feature can be explained easily enough and, to include representative boxes and other diagrams for it, would only serve to confuse an otherwise elegant illustration.

NCSA engages the Netsaint display in such a way that it becomes aware of the fact that a particular message is to be displayed by the presence of a particular file. For example, the file `/MyPath/MyHostname/load_2` would inform Netsaint that the system load on `MyHostname` exceeds a configured threshold, and that the event's associated message should be displayed: *"MyHostname: Load exceeds threshold."* If the filename was to end in `_0`, this would translate as the load being acceptable, whereas `_1` would signify a warning, rather than an alarm condition.

Netsaint retrieves a list of such files from all monitored hosts at regular time intervals, either via a socket connection or by reading a directory tree onto which a monitored host can place files depicting its monitoring stats. Any monitor that can be coded to place a file can be integrated with the display monitor in this way.

The Netsaint display will be replaced in the near future with a similar display tool built specifically for the Checksys event monitor. This will allow Checksys to be packaged as a complete event-management tool, with a web-based display monitor and a web-based configuration interface.

### **2.3 The Checksys Monitor**

Checksys is a Perl script written at NCSA, and originally intended for NCSA's specific use. It was never intended to be a packaged system monitor. However, with the development of the Alliance Virtual Machine Room (VMR) [6], and the need for a VMR-wide system-monitoring tool, Checksys is now running at all VMR sites, monitoring all VMR resources. Although Checksys is still, at this time, not a package ready for out-of-the-box use, this is its newly intended purpose, following some further development.

Checksys is modular in its design. A master script is configured to know about and to make calls to any and all available subroutines written for its use. Thus, the master script is configured for the entire NCSA environment. Each host running the Checksys master also has its own associated configuration file, listing a subset of subroutines the master script is to execute on that host. All subroutines are written as a Perl Package and require certain parameters, which are configurable and modified within each host's configuration file.

At NCSA, the Checksys master script runs from cron every ten minutes, beginning at one minute past the hour. Most subroutines are marked for every ten-minute iteration of the master, though some run only once a day, or once an hour, this also being configurable. Figure 2 provides a simple example of a Checksys master's configuration file, while Figure 3 provides a simple example of a given host's Checksys configuration file.

It should be noted that the examples provided are indeed oversimplified, not only because they do not convey actual code, but because they misrepresent the amount of event monitoring that is sometimes deemed adequate. For instance, some of NCSA's hosts run as many as 23 Checksys subroutines, and the `root_procs` subroutine is sometimes configured to monitor more than 40 processes. Checksys now has 35 subroutines, 30 of which run at ten-minute intervals.

```
When called by MyHost's cron daemon, Checksys is to

run these subroutines      run these subroutines
every ten minutes:         once per hour:

root_procs                 intruder_alert
disk_usage                 mail_queue
load                       net_stat
local_mounts
remote_mounts
```

Figure 2: A simplified example of a configuration file for a Checksys master script. All subroutines are given a time interval in which to run.

The simplified example given in Figure 2 shows that the Checksys master knows of a list of subroutines it might be asked to run, and at what intervals they are to run. When the Checksys master runs from cron at its designated ten-minute interval on a given host, the master reads the host's Checksys configuration file, a simplified example of which is given in Figure 3, to see which subroutines are to be run on that host. When running a given subroutine, the Checksys master refers again to that host's Checksys configuration file, to retrieve parameters required by the subroutine.

When the Checksys master script asks for my Checksys configuration, tell it to run	
these subroutines:	with these parameters:
root_procs	inetd:1:1:inet:admin@addr sshd:1:99:ssh:admin@addr
disk_usage	/:90:admin@addr /usr:95:admin@addr
load	512:admin@addr
intruder_alert	security@addr

Figure 3: A simplified example of a host's Checksys configuration, showing that only certain subroutines are called, each requiring configuration.

Besides monitoring events, Checksys also manages them. Checksys will restart processes not running correctly -- either not running with specific arguments, or not running the correct number of copies, as specified in the configuration file. It will also remount local and remote filesystems. Checksys also notifies the host's admin, via e-mail, regarding events found and any actions taken. A message is also recorded in the host's system log, which is forwarded to a loghost and retrieved by Swatch for the archives. Checksys also alerts a display monitor to post a particular message, by creating a particular file for a particular event. This file, that acts as a flag for the display monitor, is described in Section 2.2 above. Being an active monitor, rather than a passive one, Checksys will become aware of an event's corrected state, and alert the display monitor to take down the warning message, by replacing the earlier file with a different one, which will signal an okay condition.

### 3 Monitoring a clustered system

The task of monitoring for possible events on thousands of clustered hosts working together as a single image, where one monitor must not burden the image with thousands of copies of itself, is a problem. It is a performance issue. Yet, the task is necessary. The more hosts that are clustered in order to image a single system, the more likely something will fail. The fact is, too, that an event monitor will detect, at times, many failures. Feasibly, it could take a small army of administrators to correct such failure conditions on a clustered system consisting of thousands of hosts. So, besides event monitoring, event management is also a necessary key to the successful

operation of a clustered system, and only virtual operators and virtual administrators can efficiently handle such a task.

Performance monitoring and performance management are also essential, and for similar reasons. A performance monitor collects data from the different system layer--the hardware, kernel, service and application layers - -for administrators and users who collect such data for diagnostic purposes in order to finely tune the performance of the different system layers. However, as explained earlier, in Section 1, performance monitors are coded with performance in mind.

The topic of performance monitoring is further discussed in the next section. For now, in staying with event monitoring, suffice it to suggest that, since a performance monitor collects data, it could also be used to collect the data needed for event monitoring. In this way, the system monitor could be reduced to its event-management function only, and the actual search for event failure would be handled by an efficient, performance-minded application, which must run on the cluster anyway. It is, of course, necessary that the performance monitor of choice avails itself to this type of modification, enabling it for the task of collecting the needed event-monitoring data.

### **3.1 Checksys on a clustered system**

On a clustered system, Checksys runs from cron at ten-minute intervals on every host in the cluster, just as it does on every other host at NCSA, with two exceptions. The first exception is that it runs beginning at two minutes past the hour rather than at one minute past (the importance of which becomes clear later), and the second exception is that the clustered hosts have no Checksys configuration file.

The Checksys master cannot run without a configuration file, since the configuration file presents the master with the subroutines to be run. This means that Checksys does not monitor anything on the clustered host; that is, not until an event is detected. But, if Checksys does not monitor for events, then how is an event detected? A further explanation of this is forthcoming, but, to explain briefly, the performance monitor is charged with the collection of monitoring data, and Checksys is charged with managing events that it detects as anomalies in that data.

One host belonging to the cluster is designated as an admin host, on which Checksys will run normally -- monitoring that host at ten-minute intervals beginning at one-minute past the hour. One of the subroutines this admin host will run is called `cluster_check`. This subroutine queries the performance monitor for information regarding processes, disk usage, load, local and remote mounts, and other event-monitoring data collected from all

clustered hosts.

The `cluster_check` subroutine compares the data collected by the performance monitor with a set of expected results. This is not so difficult a task since all hosts are identical replicas of each other. If an anomaly is detected in the data for a given host, the `cluster_check` subroutine builds a configuration file for that specific host for that specific event and, one minute later at two minutes past the hour, the problem host will run its copy of the Checksys master, which will see a configuration file for its host and perform its normal, albeit quite limited, duties -- monitoring for a specific event predetermined to be an anomaly, and thus verifying the event, and taking any related action such as restarting a process, e-mailing an admin, writing to the system log, and signaling a display monitor.

The Checksys master script on the cluster's problem host, upon the completion of its succinct monitoring task, will remove its host's configuration file placed there for it by the `cluster_check` subroutine that ran on the admin node one minute earlier, and will not run again until a new configuration file appears for its host. Checksys on the clustered hosts, therefore, runs on an as-needed basis only, integrating event monitoring of clustered systems into the general event-monitoring environment at NCSA with the help of a performance monitor, the details of which will now be discussed in Section 4.

## **4 The performance monitor**

The first step toward any type of monitoring on a Linux cluster is communication with front-end and controlling hosts. This communication is a necessary evil. It is necessary because constant contact with the hosts is needed for monitoring purposes, and evil because, with potentially thousands of hosts in a cluster, there are many size and scalability issues with which to negotiate.

### **4.1 Performance Co-Pilot**

At NCSA, Performance Co-Pilot (PCP), an open-source SGI product, is the monitoring mechanism chosen for the task of communicating between clustered hosts and their controlling hosts. PCP is a performance monitor designed with simplicity and extensibility in mind.

PCP defines data in terms of performance metrics -- individual pieces of data. Furthermore, these metrics can be split into what are called instances. See Figure 4.

```
% pminfo -f mem.freemem

mem.freemem
value 1368

% pminfo -f filesystem.full

filesystem.full
inst [0 or "/dev/root "] value 29.82345066876009
inst [1 or "/dev/sda2"] value 25.13607337247675
```

Figure 4: Using the PCP command, pminfo, a host returns performance metrics, or segments of information.

Instances are useful at times, for example, when a metric contains a volatile or unknown amount of information, such as the total number of processes running at any given time. All data returned by PCP must be encapsulated into a metric. This provides a standard way of retrieving any type of data from a monitored system, making PCP easy to understand and to work with. Any type of monitor can retrieve this data and act upon it.

The default PCP install for Linux can gather over 400 metrics. These metrics range from network statistics to process data. Adding another performance metric is made possible by writing a C program using the Performance Metric API (PMAPI). This added C program is referred to as a Performance Metric Domain Agent (PMDA), responsible for returning the new user-defined metric. It is installed as a dynamic object into the main PCP daemon. This feature gives PCP its quality of being extensible.

With this framework, it is possible to monitor any aspect of a computer system, from hardware to user programs. NCSA has taken advantage of this feature by developing two PMDAs. The first one, called mounts, returns a portion of the information in /proc/mounts. The second one, called process.running, returns a true/false value as to whether certain root processes are running. This information can satisfy a need for event data, required by the system monitor, as described in the previous two sections.

**4.2 Performance Monitor Design**

The performance monitor must offer information regarding jobs and queues. The source for job and queue information is PBS, which is currently the most popular Linux-cluster scheduler, and deployed at NCSA on clustered systems. In order to integrate information from PCP and PBS, a MySQL database is populated with data from both sources, providing a standard means of collecting and accessing this data.

Data handling must be fast in order to support other components of any database-centric schema, and the database must be updated constantly. To accomplish this, information from PBS and PCP is collected frequently, and immediately inserted into the MySQL database by an efficient C program, utilizing C libraries for PCP, PBS, and MySQL.

#### 4.2.1 Command-line interface

One such component of this schema, referring to Figure 1, is the system monitor that must pull information from the database. A command-line interface is necessary primarily for this purpose. Recalling that the system monitor, Checksys, is dependent upon the performance monitor for event data from clustered hosts, it is this interface, a Perl script called ccheck (for cluster check), that queries the database and retrieves the needed information for the event monitor. The ccheck script is called by the Checksys subroutine cluster\_check, which is described in Section 3.1.

```
% ccheck -h host_1,host_2 -m kernel.all.load,mem.freemem

host_1 kernel.all.load 01 2
host_1 kernel.all.load 05 1.99
host_1 kernel.all.load 15 1.91
host_1 mem.freemem      782260
host_2 kernel.all.load 01 2
host_2 kernel.all.load 05 2
host_2 kernel.all.load 15 1.91
host_2 mem.freemem      782604
```

Figure 5: An example of the command-line interface, ccheck, showing that any number of metrics can be retrieved for any number of hosts.

#### 4.2.2 Web interface

Another vital component that must be supported by the data-handling mechanism is the web interface, which Figure 1 illustrates as a User/Admin performance application. The web site is divided into several pages, each providing the user with a view from a different level of the cluster. At the highest level, there is a page for viewing the cluster's overall status. At the mid-level, there are separate pages for the viewing of hosts, queues, and jobs. At the lowest level, the view is specific to information about a single host, queue, or job. This lowest level of detail contains a lot of information. For example, it is possible to look at the disk space available on a particular host in the cluster, or the environment variables of an active job.

The web design is done in PHP [7]. It is set up in a very extensible fashion, in that sections of the pages are described within a configuration file. Many

other aspects of the web pages are also configurable, as there are hopes to package this for general use in the near future, following further development.

## **5 Concluding remarks**

NCSA's first attempt at merging system monitoring and performance monitoring in a cluster environment seems plausible in theory, although it must be stated that, at the time of this writing, only a few tests have been made for perturbing effects it might have on cluster performance. Still, based on the premise that monitoring is necessary, there are several reasons to believe that the schema depicted herein is an advantageous place from which to start.

A great advantage inherent to this monitoring schema is the use of a database. Besides the fact that all information is stored and retrieved in an efficient and standard way, applications centered around databases are usually easier to use, easier to develop, and, more importantly for clusters, easier to scale. Also, the amount of information available from PCP and PBS is seemingly enough to satisfy any need. It will be necessary to trim the amount of information collected and, very likely, provide the user with some means to configure the performance monitor so as to collect only that information of interest to the user, over and above the minimum amount necessarily collected for administrative purposes. This would allow the user to minimize the perturbing effect the monitor could have on the cluster's performance while running a job, giving the user control over perturbing effects, which one might have to consider a cost of doing business.

## **6 References**

- [1] Performance Co-Pilot Open Source Release: See <http://oss.sgi.com/projects/pcp/>
- [2] Portable Batch System: See <http://pbs.mrj.com/>
- [3] MySQL: See <http://www.mysql.com/>
- [4] Swatch: See <http://www.engineering.ucsb.edu/~eta/swatch/swatch.html>
- [5] Netsaint: See <http://netsaint.sourceforge.net/>
- [6] Alliance VMR: See <http://www.ncsa.uiuc.edu/SCD/Alliance/VMR/>
- [7] PHP: See <http://www.php.net/>