Supermon: High-performance monitoring for Linux clusters*

Ronald G. Minnich, LANL Karen Reid, U. Toronto

September 24, 2001

Abstract

At the ACL we are building tools that monitor a cluster, anticipate cluster node failure, and take action before the node fails. Actions include migrating the processes away from the failing node, deconfiguring the node, and initiating diagnostics on the node.

A key requirement for these tools is that they require efficient, frequent data collection from the nodes. Current tools for monitoring Linux systems do not provide enough information, and what information they do provide comes slowly and at a high cost in CPU resource consumption and network bandwidth. The resource consumption is an even more serious problem in a cluster, where consumption of network bandwidth is multiplied by the number of network nodes.

In this paper we describe Supermon, a new monitoring system for Linux clusters. Supermon functions as a server, and hence can supply data over many connections to many clients simultaneously. Supermon also replaces the SunRPC interface used by Linux status daemons with a very simple text-based command and response format. Supermon has proven to be very effective for many different types of clients, including Perl and Java programs. Our new system allows programs to gather cluster performance data at up to 1000 samples/second for all the nodes in a 128-node cluster, with no measureable impact on cluster node performance.

*Los Alamos National Laboratory, an affirmative action/equal opportunity employer, is operated by the University of California for the United States Department of Energy, under contract W-7405-ENG-36. LANL publication: LA-UR- 01-4673

Introduction

At the ACL, we have built clusters ranging in size from 64 to 512 nodes, in one case to a total of 6144 CPUs. An ongoing problem with these clusters has been the lack of efficient, high performance monitoring tools which provide comprehensive information about the state of the cluster nodes. The only tools available use the rstat daemon, `rstatd`. The standard Linux rstatd daemon provides limited information about cluster nodes and extracts the information in such an inefficient way that even at very low sample rates (10 hz) the daemon itself consumes at least 10% of the node's computing power.

There is no fundamental problem with the rstat protocol itself. The problem is with the specific implementation of the Linux rstatd and the way in which it extracts performance information from the kernel. To improve rstatd we have developed changes to the daemon and some additions to the sysctl capabilities in the kernel. The changes are minor, but reduce the load the daemon imposes on the system, with the result that users can measure system performance without perturbing system. Our changes require an addition to the sysctl tables in the Linux kernel, but the modifications are minimal and do not impact other functionality in the kernel. The changes make getting kernel status much more efficient, since they use the sysctl system call instead of opening, reading, and parsing files in /proc.

We have also written a program, *supermon*, that acts as a server for providing cluster performance data for multiple remote programs. We developed Supermon to handle the problems that crop up when many remote programs need to gather performance data from a cluster. Consider the case of 16 different users monitoring all the cluster nodes. In the standard approach, all the users would query all the nodes for performance information, resulting in a greatly increased load on the cluster nodes. For high enough sample rates, many of the queries would be redundant, in that they would return the same data to the different programs.

Supermon acts as a performance monitor server that provides an intermediary for the cluster nodes, gathering data used by remote programs but limiting the query rates to the nodes. Remote programs connect to the server, and as they make queries the Supermon server responds. Queries that come in at the same time or within some small time window (e.g. 100 microseconds) would return the same data for all the queries, and result in only one request to the nodes. Supermon throttles the rate of requests to the cluster nodes and keeps the rate at a manageable level.

Many programmers have difficulty dealing with SunRPC network conventions and data formats (known as XDR), especially in languages such as Java and Perl. Rather than force users to deal with the SunRPC and XDR issues, Supermon uses a simple text-based command and response format, making it easy for programmers to access the data via programs. In practice, programmers have written everything from Java-based GUI programs to Perl programs which gather data over long periods. Occasionally people connect directly to Supermon with telnet and type commands to the server, just to get an idea of what is going on with the cluster.

In this paper we will describe the performance issues with rstatd, our changes to the kernel to resolve these issues, our additions to rstatd to make the data more useful, and the results. Finally we will describe Supermon and how programs use it for gathering data. We will then discuss future directions for cluster performance monitoring, which we believe will support clusters of 4096 nodes.

Rstatd

Rstatd is a venerable Unix daemon, dating back to the 1980s. It provides status information to remote hosts via the Sun Remote Procedure Call protocol (SunRPC). SunRPC encodes RPC services using a program number, version number, and procedure number. Rstatd provides one program number for the service, three version numbers, and two procedures for each version (the standard NULL procedure and the actual status procedure). Each version

<table>
<tr><th>V1</th><th>V2</th><th>V3</th></tr>
<tr><td colspan="3">Cpu times (idle,sys,user, nice)</td></tr>
<tr><td colspan="3">Disk transfers</td></tr>
<tr><td colspan="3">Page in/out</td></tr>
<tr><td colspan="3">Swap in/out</td></tr>
<tr><td colspan="3">Interrupts (sum)</td></tr>
<tr><td colspan="3">Net interface: packets (in/out) errors (in/out) collisions</td></tr>
<tr><td></td><td colspan="2">Context Switch</td></tr>
<tr><td></td><td colspan="2">Avenrun</td></tr>
<tr><td></td><td colspan="2">Boot Time</td></tr>
<tr><td></td><td></td><td>Current Local Time</td></tr>
</table>

Table 1: Original RSTATD versions and content

provides similar statistics, as shown in Table 1.

Remote client programs (such as rsysinfo on Linux) can query rstatd servers for the information shown in the table.

Rstatd provides too little information

Rstatd does not provide enough information about remote hosts. The performance information is not sufficient in a cluster environment to figure out what is going on. For example, it does not provide information about how much memory a node has and how much is in use; it also does not provide information about swap usage – it only shows the *rate* of swap usage. Disk read and write statistics are not available. CPU statistics are shown as aggregate numbers, not individual CPUs (although, due to a bug in rstatd in Linux, not even these numbers are right).

In order to determine how parallel programs are impacting nodes, we need to extract these other statistics. Doing so requires changes to rstatd.

Rstatd on Linux is slow and inefficient

Rstatd on Linux is slow and inefficient. It takes too long to get the information, and the process of getting the information has so much overhead that it can slow the cluster nodes down when taking as few as 10 samples/second. Rstatd on Linux 2.2.10 on a 500 Mhz. PII takes on average 22 milliseconds for each RPC request. By comparison, rstatd on a 250 Mhz. R10000 running Irix 6.0 takes 1 millisecond for each RPC request. Linux rstatd is 22

Statistic	/proc/...	Time Overhead
Load Average	loadavg	.250 ms
CPU Status	stat	.740 ms
Network Status	net/dev	.500 ms
Numusers	/var/run/utmp	4 ms
Uptime	uptime	3 ms
Memory /Swap	meminfo	9 ms

Table 2: Rstat values and the time to extract them

<table>
<tr><th>V3</th><th>V6</th></tr>
<tr><td colspan="2">CPU Times (idle, sys, user, nice)</td></tr>
<tr><td colspan="2">Disk Transfers</td></tr>
<tr><td colspan="2">Page In/Out</td></tr>
<tr><td colspan="2">Swap In/Out</td></tr>
<tr><td colspan="2">Interrupts (Sum)</td></tr>
<tr><td colspan="2">Net Interface</td></tr>
<tr><td colspan="2">Context Switch</td></tr>
<tr><td colspan="2">Avenrun</td></tr>
<tr><td colspan="2">Boot Time</td></tr>
<tr><td colspan="2">Current Time</td></tr>
<tr><td></td><td>Mem: total, used, free, shared, buffers, cached</td></tr>
<tr><td></td><td>Swap: total, used, free</td></tr>
<tr><td></td><td>Disk: read IO, write IO, read block, write block</td></tr>
<tr><td></td><td></td></tr>
</table>

Table 3: New rstat information

times slower, on a faster machine. The source of rstatd's slowness is that it opens and reads files in /proc to get kernel statistics[1]. For example, *every* time rstatd needs to get CPU status, it has to open /proc/stat, read the file in, close the file, parse the data returned by the read, and use scanf to convert the strings to numeric data[2]. In the table below we enumerate some of the sources of overhead in rstatd.

The use of /proc to gather these statistics is inefficient in many ways. Rstatd requires binary data. When rstatd reads from these /proc files the kernel formats the data via printf. Rstatd has to turn this data back into binary information via scanf. The data conversion alone takes a significant amount of time, considering a target acquisition rate of 1,000 to 10,000 samples/second.

The use of /proc is problematic for other reasons. Programs which parse the output of files in /proc can get left behind by changes in the kernel. For example, the version of rstatd shipped with Redhat 6.0 provides incorrect statistics for cpu usage on all SMP machines. The reason is that when the Redhat 6.0 rstatd is reading the output of /proc/stat, it searches for cpu usage by looking for the string "cpu". On SMP machines this matching process will succeed for the strings: "cpu", "cpu0", and "cpu1". Each time the string is matched rstatd overwrites the current data. As a result, on SMP machines, rstatd always returns status information for cpu1, not the sum of cpu's 0 and 1 as one might expect. The program also erroneously returns the wrong disk information, for the same reason. The problem is easily fixed, of course, by changing the strings to "cpu " and "disk ", but it is illustrative of the kind of errors that crop up when a program is scanning printf output.

Fixing rstatd

Fixing rstatd requires changes in both the information it provides and the way in which it gets that information. We will first describe the new rstat protocol version and then describe the new higher-performance information-gathering code.

New rstat protocol version

In the table we show the TIME version of rstat (which currently provides the most information) and the new version.

Extensions to rstat to support these new statistics are trivial. If we use the current technique of scanning files in /proc then we also need to scan the output of /proc/meminfo, and check for additional output from /proc/stat, which is already open for other statistics. We also extended the rsysinfo command to print the additional information. Adding the new version took less than an hour.

[1] The very first rstatd for Linux used the traditional interface still used by SGI. It is interesting to note that converting rstatd to use /proc was seen as an improvement, while in fact it made rstatd worse in every possible way.

[2] Note that leaving the files open and simply re-reading them does not work.

```
typedef struct
{
  unsigned long avenrun[3];f
  int bdflush[9];
  freepages_t freepages;
  unsigned int jiffies;
  struct kernel_stat kstat;
  struct sysinfo sysinfo;
  unsigned int page_cache_size;
} SYSCTL_STATS;
```

Figure 1: The supermon sysctl structure

Improving rstatd performance

The performance of rstatd is very poor. It takes too long to get information from rstatd, 20 milliseconds at least. For that period of time the CPU is consumed with handling the request – 50 Hz. sampling would leave the CPU with no cycles for real work. Ten hz. sampling consumes at least 10% of the CPU[3]. Rstatd is not only slow and inefficient, but it has a major impact on the system it is measuring. Trying to gather samples at a high rate would only result in the measurement of rstatd.

Following a discussion on the FreeBSD hacker's list and a conversation with Jes Sorenson we decided to look into the use of sysctl for getting kernel statistics. Early measurements were encouraging: it takes about 36 microseconds to move 16 KB of data out of the kernel via the sysctl interface. This is almost 1000 times less overhead for 100 times as much data, i.e. a five order-of-magnitude reduction in overhead. We found that not all the statistics we needed were supported from sysctl, and we also found that multiple sysctl calls were required to get what we did need. To support our needs more effectively we created a sysctl entry that only requires one call to sysctl to pull all the needed information from the kernel. The design of the sysctl subsystem in Linux makes this extended sysctl very easy. The new sysctl call returns the structure shown below.

The structure includes the avenrun, bdflush, freepages, jiffies, and page_cache_size; and also includes two other structures, kstat and sysinfo. The kernel_status structure (not shown) provides all the information used by the current rstatd; the sysinfo structure (not shown) returns additional information memory, processes, swap, and load.

The total size of the returned data is about 1024 bytes. Copying this much data out of the kernel only takes 5-10 microseconds depending on the machine we are using.

We built a new rstatd that used the sysctl interface. The new rstatd was twice as fast and placed much less load on the system. The performance improved substantially but not as much as we had expected. The new limitation turned out to be the way Linux manages memory and swap statistics.

Performance limitations in the Linux kernel for memory info

We were curious as to why it took several milliseconds, instead of the expected 10 microseconds, to get the data we wanted from the kernel. Our search for a culprit soon centered on the meminfo statistics. Measurements showed that getting the memory and swap info was taking a full nine milliseconds. The first problem is in the meminfo statistics as they are gathered for 2.2 kernels[4]. This code scans the page structures for every page in the system to see if it is reserved, and then to see if it has a count > 1. If the page is reserved it is not counted. If the page count is higher than 1 then it is counted as a shared page.

Scanning all the page structures accounts for the high overhead of memory info statistics. Our choices are limited to fixing the code or not collecting the data. Rather than modify this part of the system we determined that these statistics were not essential for our use, given the cost of getting them. Performance improved by 6.5 milliseconds, but was still slower than we wished. It still took 2.5 milliseconds to get the status out of the kernel.

We next checked out the swap status code and found the same problem: information about swap blocks is scanned for each call to the kernel si_swapinfo() function. In this code the inner loop is over every block of swap. For our machine there are 4096 such blocks. Once again a simple bit of bookkeeping for each swap device would result in more efficient status information. We needed this in-

[3]The exact load on the 2.2 kernel varies with the size of memory and swap. Four nodes with 1 Gbyte memory and swap, the load is at least 10%.

[4]Note that in many cluster installations 2.2 is still the OS of choice, for driver and other compatibility reasons.

formation, so for now we are accepting the lower performance.

As of 2.4, the memory code has been fixed, but the swap info code has not.

Supermon, a tool for extracting data from clusters

The improved rstatd resolves problems with getting the data from the *individual* cluster nodes. There remains the problem of gathering the *aggregate* performance data from the nodes. Often many different users wish to see how well the cluster is (mis)behaving. To get this information, the users have to run programs that query all the nodes for performance data. Having many different programs continually poll all the cluster nodes for data can consume the network. We found we needed a tool that made it easy for different types of programs to gather the data and filter for only certain types of data. The tool also needs to function as a server, so as to combine requests from different client programs and thus reduce the rstad load on the cluster nodes.

Supermon is the tool we have built that extracts data from all the machines in a cluster simultaneously. Supermon uses vector RPC. A full discussion of vector RPC is outside the scope of this paper, see http://www.acl.lanl.gov/~rminnich for more information. Supermon has a very simple character-oriented command set and response. The emphasis is on a command set that any type of remote program (Java, Perl, etc.) can connect to and use. The commands are a single character, and do not require any transmission of newline or carriage-return characters in order to work. Return messages are in the form of newline-delimited lines, with two newlines indicating end of message. Characters (such as newline and carriage-return) not in the command set are ignored. The commands are:

#. Describe the status information available. The information is presented as one line per item of information, ending with two newlines. The format is: <Mask #> <name> <U> for information that is unbounded, such as uptime. For bounded information, such as memory usage, the format is <Mask #> <name> <B> <lower-bound> <upper-bound>. The mask number is used by applications to mask out data they don't need to see. This masking saves work for the application and network bandwidth.

S. Send the status. The status is sent as follows: <host base> <host number> <name> <value>. The host base is the base name of the host in the cluster, and the number is which host it is. For our lbp cluster, output for memfree looks like:

```
lbp 1 memfree 376119296
lbp 2 memfree 391376896
```

In this case, the the base is lbp, node numbers 1 and 2, with memfree values as shown.

a-f,A-F. Hex numbers are used to build a mask. The mask is delimited by any non-hex number, such as newline, space, or another command.

A sample session with supermon is show below. The first two lines always show the cluster name and the number of nodes. Since these variables are not presented by the S command the mask for them is zero. They are bounded and zero-based. To date, all bounded variables are based at zero but we do not want to assume that this is a universal rule for monitored information.

A Perl client for Supermon

For one type of performance monitoring we built a Perl client. We use the Perl client to collect traces of Supermon data. Collection is controlled by a user-specified parameter file, which describes the sampling rate and the list of variables that are to be traced. The Perl client makes a socket connection to Supermon, constructs the mask that it sends to Supermon to tell it which variables to send back, and then issues 'S' requests to Supermon at the specified sampling rate. Each sample is parsed and written to the trace file. One sample contains one line for each variable where a line has the current value of the variable reported by each of the processors. Another Perl script parses the output file into a format more useful for creating graphs and examining correlations between statistics or processors.

To get an accurate interval between samples, we use a high resolution version of usleep from the Perl module Time::HighRes. There may be some variance in the intervals if the Perl client is run on a heavily loaded machine. Since communication with Supermon is through a socket, the client does not need to run on the same machine as

Supermon. Therefore, data collection and trace processing do not interfere with the system being monitored, and the client adds no additional overhead to the system.

The figure below gives an example of how trace output might be used. The graphs illustrate the start up process of an MPI application. Each subgraph presents data from one node of the cluster. The solid lines show the number of interrupts measured each sample (left y axis), where a sample is taken every half second. The dotted lines are the cpu ticks attributed to system use each half second sample (right y axis). We notice a strong correlation between interrupts and system time (as we expect). We also clearly see how long it takes to start up each node. The first peak in the solid line occurs at 4 seconds in the bottom graph and at 20 seconds in the top graph, which is when the final node is initialized. Looking at kernel data gives us information that we might not otherwise see. For example, the bottom graph (node 1) shows that this node uses half the amount of system cpu time of the other nodes. The remaining cpu time turns out to be idle.

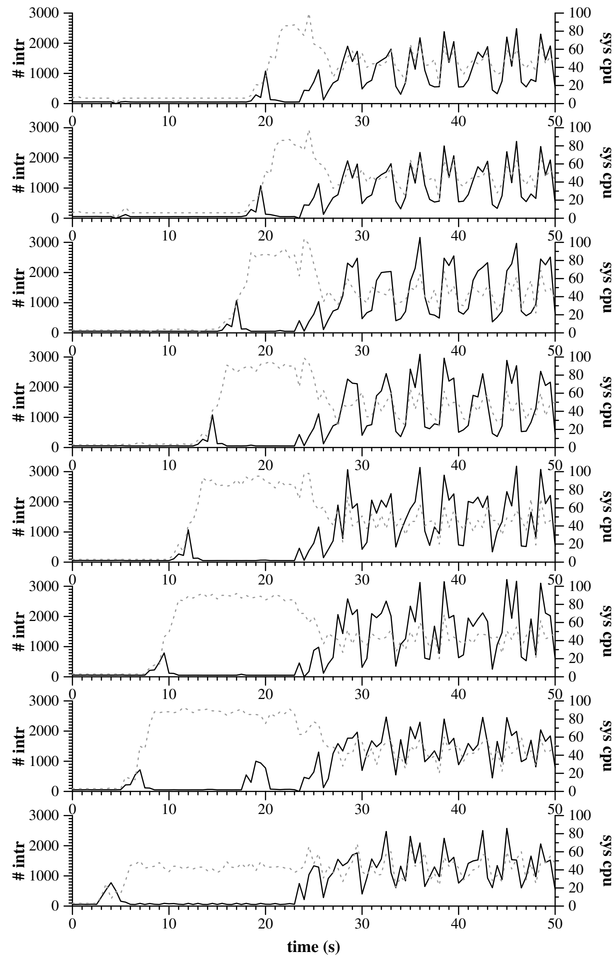

Figure 2: Supermon statistics for starting 8 cluster nodes

Performance

Performance of the new rstatd is very good. At 100 samples/second the impact of the new rstatd is not measurable, whereas the old /proc-based rstatd consumes at least 12% of the CPU. In other words, past a point the old rstatd was measuring its own impact on the CPU, since it was creating such a load on the system. The new rstatd has a much lower impact.

Supermon performance is correspondingly good. Supermon can easily monitor 100 hosts at 100 samples/second. The filtering has proven useful for both graphical front ends and perl scripts which gather data.

Next Steps

In an age when Java, Perl, and even Telnet users wish to connect to daemons and send queries it no longer makes sense to use SunRPC-based daemons for status monitoring. It is too hard for remote programs to deal with the complex encoding used by SunRPC for data, and it is more efficient to simply encode binary data as printable strings and send it over the wire. Also, the filtering that supermon does should be done directly at the source. As it is now, rstatd gives us a lot of information, much of which we simply throw away. Filtering should be done at the source, not at an intermediate point. We are building a new program, mon, which replaces rstatd. Mon will use the same command set as supermon and provide the same information. Supermon will be modified so that it gathers statistics from many remote mon programs.

Conclusions

We have described changes to rstatd that make it 6 times faster and far more efficient than the current /proc-based rstatd that is in common use on Linux. The changes are minimal (and available on our web page). We have also described a new program, Supermon, which allows for high-performance, low-overhead monitoring of hundreds of cluster nodes. Supermon is being used at the ACL to gather statistics from our clusters. All these tools are available under the GNU GPL from the ACL web site.