

Nodescape: A Machine Status Monitoring System

J. Frank Roberts
jfrobe2@engr.uky.edu
September 12, 2012

1 Introduction

Monitoring of machine health is important to the maintenance of any group of computers. Health monitoring allows administrators to detect and repair potential problems, preventing unplanned downtime. As our computing infrastructure continues to grow, it is becoming more difficult for administrators to quickly detect small problems. Our ability to collect and store data has scaled well with the size of our infrastructure, but our ability to analyze and present that data has not.

The intent of Nodescape is not to find better ways to collect and store data; good solutions exist for this problem. The motivation behind the development of Nodescape is to find new ways to analyze and present the vast amounts of data that we are now collecting. Nodescape will analyze and present data such that it is obvious to an administrator when something is wrong. Nodescape will make it possible to tell at a glance how healthy our machines are.

The underlying infrastructure for Nodescape has recently been rewritten to facilitate this goal. Though not ready for use in production environments, the infrastructure is adequate for collecting data to use for the development of new presentation tools.

2 Previous and Related Work

Nodescape is not the first attempt to build a scalable computer monitoring system. Several robust monitoring systems enjoy widespread use. Nodescape itself is not new; the work presented here is based on a previous version. Development of Nodescape has been, and will continue to be informed by other monitoring systems.

Nodescape The first version of Nodescape[1] was written by Dr. Hank Dietz, and is intended for use in a compute cluster. Dr. Dietz's version of Nodescape consists of a monitor and a collector. The monitor runs on each machine being monitored. It may run as a daemon, or it may be scheduled by an external program. The monitor is configured by the arguments passed when the program is run.

Dr. Dietz's collector receives status updates from the monitors and stores the data internally. The front-end is also integrated into the collector. The collector uses the status information to color an image of the cluster being monitored. Each node in the cluster is colored based on a single statistic (e.g. CPU temperature, load average). Whether a value is out of range is determined by how far it falls from the current and historical averages for that statistic.

Munin Munin[2] is written in Perl, and is compatible with most Unix/Linux derivatives. The Munin master collects data from running instances of the munin-node program. Users may write plugins for munin-node if they want to monitor additional properties of the computer system.

Unlike Nodescape, munin-node does not push data to the master. Instead, the master must be configured to pull data from the nodes. The master builds graphs from the data,

and presents these through a HTML web interface. Graphs are built for periods of one day, one week, one month, and one year.

Ganglia Ganglia[3] was designed specifically for use in high-performance computing environments. Like Munin and Nodescape, Ganglia uses the monitor-collector model for monitoring multiple machines. Ganglia is primarily used and configured through a web interface, but will also present data in a command-line environment. Ganglia supports a hierarchical monitoring structure, and scales to very large installations. Ganglia is in production use by many computing centers and corporations.

Cacti Cacti[4] is a front-end for rrdtool[5], a graphing and logging tool which is also utilized by Munin and Ganglia. Cacti does not support monitoring multiple machines directly, though the user may write scripts that collect information from other machines. Cacti provides a robust interface for modifying and creating graphs.

Pulsar Pulsar[6] is a tool for monitoring large Unix sites. Pulsar has three components: a presenter, a scheduler, and pulse monitors. The scheduler runs on each machine that is to be monitored. A configuration file determines which pulse monitors the scheduler is to run on each host. When a pulse monitor is run, it first invokes a command to gather data from the machine. The pulse monitor converts the result of the command (also called an alarm) to a comfort level, and reports that comfort level to the presenter. The presenter runs on the administrator's machine, and displays an icon for each alarm that it receives. The icons are colored based on the reported comfort levels. A comfort level of zero is not displayed at all. As the comfort level rises, the alarm icon is displayed as green, then yellow, then red. The administrator may get more information about an alarm by selecting the alarm.

3 Back-end Implementation

The Nodescape **back-end**, that is, the portion of Nodescape that collects and stores data, is broken into three pieces. The **monitor** extracts status information from the machines being monitored. The **collector** receives data from one or more monitors and inserts it into a MySQL database. The **database** makes the data available to front-end tools. In this context, a **front-end** is any tool that extracts status information from the database and presents it to a human, the **user**.

Nodescape's research value lies primarily in the development of front-end tools, so ease of development has been an important factor in writing the back-end. As a result, the back-end is simple and, in some ways, restricted. Distribution of Nodescape would require that the back-end be made more robust and fault-tolerant. This minimal implementation exists only to gather data for use in the development of front-end tools.

3.1 Monitor

The purpose of the monitor is to gather and tag status information, and to send the tagged information to the collector. The monitor runs on each machine from which we wish to

Option	Expected value	Description
host	IP address or hostname	The monitor will send information gathered from the status checks to the collector running on the specified host. The monitor exits with an error if this option is not set.
group	A group name	This option is not required. The monitor appends the name specified to the hostname before sending status information to the collector. The group name must not contain whitespace. If no group is specified, nothing is appended to the hostname.
port	UDP port number	The port that the collector is listening on. If this option is not set in the configuration file, the monitor uses port 45231.

Table 1: Monitor configuration options.

gather status information. The implementation of the monitor relies on other parts of the environment. The monitor does not gather status information directly. Instead, it relies on built-in or user-supplied utilities to extract information from hardware sensors, the operating system, or other pieces of software. The monitor is not a daemon, and does not run continuously. Rather, the monitor is run explicitly by the user, or it is scheduled by an external program, such as cron.

The monitor reads its configuration from the file passed as the first argument to the program. The configuration file contains a list of configuration options and a list of status checks to be run. Table 1 shows the options that may be set in the monitor.

A label and a shell command comprise a **status check**. The label describes the significance of the output from the shell command, and is used to tag the output in the database. The shell command may be a single command, or a series of commands connected by pipes. The monitor runs the shell command and sends the output to the collector.

Field	Contents
host	The name of the machine on which the monitor is running. This may include a group name specified in the configuration file. The format is hostname.group.
label	The label specified for this status check in the configuration file.
data	The output from the status check, represented as a string.
status	An integer describing how the status check command completed. This feature is not yet fully implemented.
when	The time recorded on the local machine immediately before running the status check command. The time is recorded as an integer representing the number of seconds since January 1, 1970.

Table 2: Status check package fields.

After reading the configuration file, the monitor runs the status checks one at a time. After running each status check, the monitor creates a **status package** containing:

1. the hostname of the local machine
2. the label of the status check
3. the output from the status check
4. a status code indicating whether the status check ran without error
5. the current date and time

The monitor does not process the output from the status checks in any way, except to remove leading and trailing whitespace. The monitor sends the package to the collector. Table 2 describes the status check package in detail. The monitor exits after it has run all of the status checks listed in the configuration file.

3.2 Collector

The collector receives status information from the monitors, and stores the information in a MySQL database. The collector runs as a daemon on a single machine; that machine may or may not also host the database server. Like the monitor, the collector configures itself based on the file passed as the first argument to the program. Table 3 shows the options that may be set in the collector.

Option	Expected Value	Description
port	UDP port number	The port to listen on. If this option is not set in the configuration file, the collector listens on port 45231.
user	A MySQL user name.	The name of the account to use when connecting to the MySQL server. This option is required.
dbhost	A hostname or IP address	The hostname of the machine hosting the MySQL database. This option is required.
passwd	The password associated with the MySQL account	This option is not required, but there is currently no other way to specify a password for connecting to MySQL.
dbname	The MySQL database to use	This option is required.
table	The name of the table to use	Specifies which table in the database to insert status information into. This option is required.

Table 3: Collector configuration options.

After reading the configuration file and connecting to the MySQL database, the collector blocks, listening for status packages from monitors. When the collector receives a status

package, it unpacks the information, inserts it as a single row into the database, and goes back to listening for status packages. Once the collector starts listening for status packages, it does not exit unless it is killed.

MySQL server allows remote connections, so moving the database functionality into the monitor would eliminate the need to have a collector. My design uses a discrete, central collector for two reasons. First, separating the insertion of the data into the database from the collection of the data follows the Unix design philosophy. That is, programs should be simple and do one thing well. Complex software systems should be built by chaining together many simple programs. Second, moving the database functionality into the monitor would introduce another dependency for the monitor. Any machine that we wished to monitor would need to have the MySQL client libraries installed.

3.3 Database

The collector inserts all of the status data into a single table in the database. The six columns in the table are: host, label, data, status, ctime, and mtime. The collector populates the host, label, data, status, and mtime fields from the data in the corresponding status package. MySQL places a timestamp in the ctime field as each row is inserted. Table 4 describes these fields more thoroughly.

Field	Populated from	MySQL type	Description
host	host status field	varchar(40)	Hostname of the machine this status was recorded on.
label	label status field	varchar(40)	Label specified in the status check for this status.
data	data status field	varchar(40)	Output from the shell command associated with this label.
status	status status field	int	The completion status for this shell command.
ctime	MySQL CURRENT_TIMESTAMP	timestamp	MySQL populates this field with the current time and date as the row is inserted.
mtime	when status field	timestamp	The MySQL from_unixtime() function is used to convert the integer number of seconds to a MySQL timestamp.

Table 4: Database table rows.

Handling status data as strings allows the implementation of the Nodescape back-end to be both simpler and more flexible compared to handling status data as numeric values. Handling strings is simpler because no type conversion is necessary. The output from the status check shell command is text, and remains so as it moves through the monitor and the collector. String representation is more flexible because the user may have status data that is non-numeric. Conversion of values is left to front-end tools.

Table 5 contains some sample data collected by the Nodescape back-end. These data are from the machine coreopsis. The monitor is configured to treat this machine as part of the **cs** group. Status checks are configured for memory usage, memory used as cache, load average over the last 1, 5, and 15 minutes, number of users logged in, and number of processes. The current version of the monitor simply sets status to zero. The monitor configuration on coreopsis is included below.

Listing 1: Coreopsis monitor configuration.

```
# IP
host example.com
port 45231
group cs
#: Label : Command
: used memory : /homes/jfrobe2/bin/usedm.sh
: cache memory : /homes/jfrobe2/bin/cachedm.sh
: loadavg : cat /proc/loadavg | cut -c 1-14
: user count : who | wc -l
: process count : ps aux | wc | cut -c 4-7
```

host	label	data	status	ctime	mtime
coreopsis.cs	used memory	704	0	2012-09-08 17:18:38	2012-09-08 17:23:01
coreopsis.cs	cache memory	2366	0	2012-09-08 17:18:38	2012-09-08 17:23:01
coreopsis.cs	loadavg	0.01 0.02 0.05	0	2012-09-08 17:18:38	2012-09-08 17:23:01
coreopsis.cs	user count	10	0	2012-09-08 17:18:38	2012-09-08 17:23:01
coreopsis.cs	process count	257	0	2012-09-08 17:18:38	2012-09-08 17:23:01

Table 5: Sample Data.

4 Front-end Ideas

Our primary objective in developing Nodescape is to look for ways to analyze and present large amounts of computer status information. Monitoring a single status check at a five minute interval on a sixteen node cluster over a twenty-four hour period will generate over 4600 status records. Deriving some value from this volume of data requires analysis. Our goal is that by using Nodescape, a user should be able to tell, at a glance, whether anything is awry with their machines. We intend to accomplish this goal by exploring a number of front-end designs.

Most of the presentation concepts that we are currently developing are based on pre-existing interface and presentation concepts. Our hope is that as we develop these concepts,

some truly novel ways of presenting the data will arise.

4.1 Per-node Alerts

This front-end presents an array of images or icons. Each image corresponds to a machine being monitored. If any of the status checks on that machine return a value that is out of range, the image is made conspicuous. This may be accomplished by using a different color or by animating the image. A value may be determined to be out of range by several methods. The front-end may use predefined values to define the range. It may also use statistics based on historical trends, or based on current values across a group of machines. Selecting a particular image presents more information about the corresponding machine.

4.2 Statistical Analysis

Present the user with descriptive statistics calculated from the status information collected. Some default set of statistics is calculated for each status check. These statistics may present a summary of the state across all machines, or across a group of machines. They may also summarize data across time. This front-end also should allow the user to configure new statistics to be calculated at any time.

4.3 Graphing

Graph the data. Graphs provide a visualization of long-term trends. In addition to visualizing trends, graphing may be used to visualize a correlation between two status checks. This front-end might work best if it was integrated with the statistical analysis front-end. The user should be allowed to request custom graphs.

4.4 Manual Analysis

The user should also be able to manually analyze the collected data. At least one front-end for Nodescape will attempt to minimize the difficulty of manual analysis. Robust searching and sorting capabilities will be required. While MySQL allows the user to construct sophisticated queries, its syntax is verbose and cumbersome. This front-end presents a set of tools that, while simple and straightforward, do not restrict the user's ability to filter the data.

This front-end presents the data in layers. A list of machines is presented at the top level, and the user is able to select any machine and get more detailed status information about that machine. The user may sort the list by the results of one or more status checks. Filters may be constructed to limit what information is shown.

5 Conclusion

Development of the new version of Nodescape is in the early stages. A simple back-end has been developed. Ideas for several presentation front-ends are being entertained, though they all have yet to fully mature. From this point, most development efforts will focus on

implementing some portion of the front-ends suggested here. In addition to developing new presentation concepts, some effort will be made to improve on the existing concepts implemented in other monitoring systems. Of course, other ideas for the analysis and presentation of machine status data are likely to surface as the development of front-ends begins.

References

- [1] The Aggregate's Nodescape Utility. <http://aggregate.org/NODESCAPE>. 2012.
- [2] Munin. <http://munin-monitoring.org>. 2012.
- [3] Ganglia Monitoring System. <http://ganglia.info>. 2012.
- [4] Cacti - The Complete RRDTool-based Graphing Solution. <http://www.cacti.net>. 2012.
- [5] RRDtool - About RRDtool. <http://oss.oetiker.ch/rrdtool/>. 2012.
- [6] Finkel, Raphael A. (1997), Pulsar: an extensible tool for monitoring large Unix sites. *Software: Practice and Experience*, 27: 1163-1136. doi: 10.1002/(SICI)1097-024X(199710)27:10<1163::AID-SPE124>3.0.CO;2-N