**Scalable Systems Software Enabling Technology Center
Plugable Scalable Systems Monitoring Component**

Final Technical Report

Date of Issuance: April 2009

Author:
Michael T. Showerman
120 W. Clark
Urbana, IL 61801

Originating Research Organization:
The Board of Trustees of The University of Illinois at Urbana Champaign
801 S. Wight St, 109 Cobel Hall
Champaign, IL 61820

Project Summary

NCSA's role in the SCIDAC Scalable Systems Software (SSS) project was to develop interfaces and communication mechanisms for systems monitoring, and to implement a prototype demonstrating those standards. The Scalable Systems Monitoring component of the SSS suite was designed to provide a large volume of both static and dynamic systems data to the components within the SSS infrastructure as well as external data consumers.

Design Goals

Scalability is central to the design of this component. The number of devices in high performance computing systems has been dramatically increasing for large installations over the last few years. Concurrently, the availability of high quality data at the device level has expanded significantly as well. Network switches, interconnect switches, power controllers, host adapters, storage systems and many other devices incorporate not only performance information, but other data that is useful in predictive failure analysis: such as temperature, voltage, transient memory/CPU errors and fan speed information.

In traditional systems, there are multiple separate and often overlapping infrastructures to gather and interpret this information without a common interface to provide the data to its consumers. Resource managers, performance monitors, and administration/health monitors often use independent mechanisms for collecting their own information without any aggregation or sharing. This project addressed a common interface specification for the collection and distribution of device data in an extensible and scalable manner.

Accomplishments

This software component had three phases of the development cycle. Stage one involves designing a monitoring system prototype that collects the necessary data for the other Scalable Systems Software components it must interact with, and to define an extensible XML interface. In addition, this interface was designed and tested to provide a framework that accommodates the expansion of new data types and devices to be monitored. Existing software used for the collection and visualization of system performance data were adapted to integrate with this new communication mechanism, and new applications were tested to demonstrate the flexibility of the component interface. The data was viewed graphically using existing tools developed locally. In addition to the collection of system performance data, an application had been developed to view the registration and communications between the software systems contained within the SciDAC Scalable Systems Software project. This application was helpful in the debugging phase of component interactions, and was also is a visual aid in demonstrating the communication paths within the entire scalable systems software stack.

The second phase of development was the design and implementation of the software foundation for a scalable extensible monitoring hierarchy named Warehouse. This first involves daemons to monitor hardware/software systems. The design of this software incorporates the use of an abstraction layer that partitions the software between platform/system specific sections, and infrastructure object software. This is partially accomplished by developing portable software objects for the internal data storage and communication mechanisms, while defining a functional interface for the collection and querying of the data. The implementation of this functional interface is provided in dynamically linked shared libraries. This allows a single daemon to collect and export different or new information based on the implementation of libraries linked at run time. This model requires the use of internal data stores that are flexible in the manner they retain their data.

This design also requires additional isolation of the data content from the processing at the middle layers of the collection hierarchy. In this phase, research into the scalability/ data quality/ extensibility tradeoffs were required to tune the software layer used to aggregate the data and export it via the XML interface. Light weight semi-intelligent protocols were developed for communication between the devices producing performance/state data and the aggregation components. Variations in the protocol were used to adjust tradeoffs between communication overhead and data processing overhead.

Warehouse was designed to be topology-agnostic and type-agnostic. A warehouse information gathering system (a "tree" of warehouse processes") can be configured in an arbitrarily complicated way that fits the needs of systems it is monitoring. Information can be handed off an arbitrarily large number of times, and collected at any point in the system. Each warehouse is required only to know where it is getting information. Trees of warehouse processes can be created with no practical limit to the number of warehouses in the tree, or the arrangement of the processes in relation to each other (thus "topology agnostic").

Warehouse is also type-agnostic. The information that passes through the tree for devices is one of three types, integer, floating-point, or string. Each piece of information has a name by which it is identified. The only reason all information isn't strings is so that comparisons can be made efficiently. (A sample request might be "give me all compute nodes with percent CPU used less than 20".)

The communication design was eventually changed after initial scaling tests to a priority thread pool system specific to warehouse to handle the potentially large number of client sockets. Using this model, connections are categorized and handled by separate pools of threads according to their response behavior. This method is instrumental in maintaining responsiveness in large scale systems where multiple complex failure modes are present. When this system was tested, we were able to test up to 1000 nodes from a single warehouse layer in the hierarchy. As warehouse applications grow, and the number of nodes monitored becomes large, data messages will be split across network packets. To support correct reconstruction of packets, a warehouse protocol was created that frames messages and allows reconstruction by the receiver.

Using a thread pool scheme for processing network connections exposed many other problems with the network code in warehouse. Any connection behaving badly now has the potential to delay the monitoring of other connections. Improving the network robustness has been an ongoing task throughout the project. Monitoring a cluster for tests at ORNL under real workloads use proved useful for

demonstrating several challenges in the design. We were able to demonstrate a wide variety of network and device abnormal states while maintaining overall system responsiveness.

A student was hired to enhance the design of the information store structure to improve scaling and efficiency. Secondary effects of the information store changes involve enhancements to begin supporting meta-monitoring tasks that aggregate data for operations such as job based statistics. The warehouse networking components were also transitioned to expand the portability to new platforms. The software was then packaged and made available as a source distribution as well as a package for the OSCAR-SSS deployment suite.

In the third phase, new visualization software, performance archiving and meta-monitoring components were developed to demonstrate and test the functionality of this system. For an example of a meta-monitor, a software component can be used to gather job information from the queue manager, process information from the process manager, and correlate the real-time performance data from the system monitoring component to provide job-based monitoring. This could provide standardized metrics like actual memory or CPU usage at the job level, or provide new types of job based data as a result of site specific implementations of the data collection shared libraries. In the case of a cluster, this may be current bytes/sec of the interconnect for a specific job to help understand performance bottlenecks of a cluster.

Also in the third phase, new functionality to improve administration and scheduling capabilities were added by providing query engine to the performance database. This could be used to provide an XML response to devices that meet a certain criteria. A scheduler may frequently query online hosts with available CPU, memory, disk, rather than gathering all of the necessary scheduling parameters for suitable and unsuitable hosts to arrive at the same conclusion. Visualization techniques for choosing and displaying meaningful data for very large scale systems were also be developed. This involves both dense data representation, and the creation of metrics that provide sufficient contrast to illuminate the important differences between devices and jobs.

We then began the development of a tool to begin displaying the data collected by warehouse that we called NodeView. The first section involves the creation of the communication routines for connection to warehouse. The initial design uses the assumption that later stages will use differential protocols, so the entire available dataset is currently collected by NodeView.

The second section was to parse out the node properties and display them in some sort of tree structure. I am using a QListView, which is similar to the left panel you would find in the Windows Explorer. Each line entry corresponds to one of the node properties in the XML response string. There is a “+/-” box to expand/collapse the tree at each leaf. See the image below.

Warehouse	Value	
Node		
NodeId	oscarnode1	
NodeState	up	
Configured		
Memory	501	
Swap	1019	
Processors	4	
Available		
Processors	4.000	
Memory	444	
Swap	1019	
Node		
NodeId	oscarnode2	
NodeState	up	
Configured		
Memory	501	
Swap	1019	
Processors	4	

Expand All
Collapse All

Hide Unselected

The Warehouse List View Pane

The third part of the process was in regard to the ability to select the nodes to be displayed/graphed based on a set of filters. The “SSSRMAP Message Format” specification allows for the requesting to particular data using the “Where” element. The NodeView GUI has something called “Selection Filters”. These Selection Filters allow a user to create filters for selecting nodes based on their properties and values. When the tree is created, every node property is saved to a list. This property list will appear in a drop-down box which can be combined with conditionals and values to create a filter. The nodes selected from this process will be the ones that are graphed. The next image shows the Manage Selection Filter dialog box which allows the user to create new filters, or copy, edit, rename, or delete old ones. The value in the large list box is the “short” name of the filter. The box at the bottom would display the actual filter which was created by the user.

The Manage Selection Filters Window

The next image shows the actual viewer/editor for Selection Filters. In this window, you will be able to string together a list of filters using AND or OR, remove individual lines with the “X” box, and move the lines around with the UP and DOWN arrows.

The Selection Filter Edit/View Window

The fourth part of the fourth is to select what data is plotted, i.e. the y-axis. Since the type of data available from the warehouse is not known beforehand, we need a way to specify what values are actually plotted for those nodes selected by the Selection Filter. A kind of equation editor was implemented where

the user can 'write' simple equations using the existing properties and a small set of operators such as +, -, *, /, COUNT, SUM, AVG, MEAN, etc. As with the Selection Filters, these user-defined equations could be saved so that one (or perhaps more at a time) could be selected for graphing. This part has not yet been implemented other than creating the placeholder for invoking the data plot equation editor.

The fifth part of the project is the actual graph. Many different styles of graphs could be used. For the first graph style, We want to give the user two ways to view a large number of nodes along the x-axis. The first (and most basic way) is to allow the user zoom in/out (in effect determining how many nodes can be displayed in the window at a time) and then present the user with a scrollbar so as to view only a subset of the nodes at a time, but be able to scroll to view others. The second (and more abstract way) is to allow the user to enter a 'clumping' number, which defines how many nodes are grouped into a single bar. For example, if you have 1000 nodes to be plotted, but you wanted to see them all in a single window, you could clump 50 nodes into a single bar, giving a total of 20 bars to plot. When clumping nodes together, the user could select from several algorithms for calculating the value that is plotted for a bar. For example one could plot the average of all values of nodes within a clump, the minimum/maximum of all values, maximum minus average, etc. This methodology could enable several calculations which would allow a user to quickly see when there are aberrant nodes within a clump. The user could then click on that clump and 'zoom in' to those nodes by creating smaller clumps. Using these two methods of viewing, a 2-D bar graph was able satisfy all needs for cluster visualization as an initial implementation.

Main Node View Window

Results and Conclusions

The design and development of this component resulted in a prototype implementation that was sufficient for data collection for the components within the Scalable Systems Software project as well as for providing monitoring data to external consumers. The resulting protocol and software designs for networking served as a basis for future data gathering and aggregation projects. Internally at NCSA we developed multiple follow on monitoring systems and visualization applications for performance information, airflow distribution, and interconnect routing analysis/monitoring. In addition to the continued technological evolution from this project, collaboration with nearly all the groups of this project has remained strong and contain ongoing joint development activities.