Time-Synchronized Visualization of Arbitrary Data Streams[*]

Paul Z. Kolano
NASA Advanced Supercomputing Division, NASA Ames Research Center
M/S 258-6, Moffett Field, CA 94035 U.S.A.
paul.kolano@nasa.gov

ABSTRACT

Savors is a visualization framework that supports the ingestion of data streams created by arbitrary command pipelines. Multiple data streams can be shown synchronized by time in the same or different views, which can be arranged in any layout. These capabilities combined with a powerful parallelization mechanism and interaction models already familiar to administrators allows Savors to display complex visualizations of data streamed from many different systems with minimal effort. This paper presents the design and implementation of Savors and provides example use cases that illustrate many of the supported visualization types.

Keywords: visualization, time synchronization, parallel data streams, data analysis, system monitoring

1. INTRODUCTION

Large computer installations involve huge numbers of interacting components that are subject to a multitude of hardware failures, transient errors, software bugs, and misconfiguration. Monitoring the health, utilization, security, and/or configuration of such installations is a challenging task. While various frameworks are available to assist with these tasks at a high level, administrators must more often than not revert to using command line tools on individual systems to get a low-level understanding of system behavior. The output from such tools can be difficult to grasp on even a single system, so quickly becomes overwhelming when taken across many hosts.

A variety of visualization tools and techniques have been proposed to increase the amount of information that can be processed by humans at once. Existing tools, however, do not provide the flexibility, scalability, or usability needed to assist with all the varied information streams possible in large installations. In particular, these tools often require data in a specific format and/or in a specific location with interfaces that have little relation to the underlying commands from which the data originates.

This paper presents a new visualization framework called Savors, **S**ynchronization **A**nd **V**isualization **O**f a**R**bitrary **S**treams. The goal of Savors is to supercharge the command-line tools already used by administrators with powerful visualizations that help them understand the output much more rapidly and with far greater scalability across systems. Savors not only supports the output of existing commands, but does so in a manner consistent with those commands by combining the line-editing capabilities of vi, the rapid window manipulation of GNU screen, the power and compactness of perl expressions, and the elegance of Unix pipelines. Savors was designed to support *impromptu visualization*, where the user can simply feed in the commands they were already using to create alternate views with optional on-the-fly aggregation of information across many systems. In this way, visualization becomes part of the administrator's standard monitoring and analysis process with no need for a priori aggregation of data at a centralized resource or conversion of the data into a predefined format.

Savors supports of all four combinations of single/multiple data streams and single/multiple views. That is, Savors can show any number of data streams either consolidated in the same view or spread out across multiple views. In multi-data scenarios, data streams can be synchronized by time allowing even distributed data streams to be viewed in the same temporal context. In single-data multi-view scenarios, views are updated in lockstep fashion so they show the same data at the same time. Together with its integrated parallelization capabilities, this allows Savors to easily show meaningful results from across even very large installations.

[*] This work is supported by the NASA Advanced Supercomputing Division under Task Number ARC-013 (Contract NNA07CA29C) with Computer Sciences Corporation

This paper is organized as follows. Section 2 presents related work. Section 3 discusses the Savors console. Section 4 details Savors data handling. Section 5 describes the Savors view components. Section 6 covers example use cases and provides samples of available visualizations. Finally, Section 7 presents conclusions and related work.

2. RELATED WORK

A previous incarnation of Savors, dubbed the **S**calable **A**ural-**V**isual **O**perations **R**oom for **S**ecurity, was discussed in previous work,[12] but had several notable limitations including fixed views that were only applicable to specifically formatted security data, no notion of synchronization between multiple views of even the same data, and difficulty in manipulating the tools on large multi-monitor displays. An inspiration for the current version was VisTextFlow-IP,[23] which combines text and visual interfaces to allow administrators to get the benefit of visualization while still interacting with the familiar text interface. While the goals are similar, VisTextFlow-IP is limited to the one specific command that can be represented within its visualizations, limiting its generality.

Some tools are designed to assist administrators in visualizing information from specific types of services. Dos Santos et al. visualize NFS servers, network data, and web servers using various 3D metaphors[5] including cities, cone trees, solar systems, pyramids, landscapes, and libraries. ENAVis[14] shows the connectivity between hosts, users, and applications using graphs of data derived from standard system tools including `netstat`, `ps`, and `lsof`. Theia[9] is a tool to diagnose problems in Hadoop clusters using heatmaps to show anomalies associated with jobs over time. Hochheiser et al. use scatter plots to show various relations between URLs, hosts, and times of web server logs,[11] with radius and color representing request frequency and HTTP return status. While each is useful for its specific domain, Savors was designed to be applicable across many different domains.

A number of techniques have been developed to visualize general log files. SeeLog[6] shows entire log files on a single screen using multiple columns of lines with length proportional to a single log message and colored by message type. Frei et al. use a histogram matrix to analyze mail server logs,[8] where the radius of the circles in the grid shows the frequency of log message word count and the color indicates frequency changes between intervals. LogView[16] shows the contents of log file event clusters using a treemap colored by event type severity. Girardin et al. describe an approach for real-time monitoring of logs[10] using spring graphs, self-organizing maps, and parallel coordinate plots. MieLog[21] visualizes log files using four regions consisting of a heatmap of message types, a time region depicting message frequency over a frequency timeline, an outline region similar to that of a column of SeeLog, and the actual log messages in the highlighted area. Several of these views are useful and may be incorporated into Savors in the future, but on their own do not support multiple views of the same data.

A variety of tools support the single-data multi-view model to show different perspectives of the same data. RUMINT provides various coordinated security visualizations with DVR-like controls, which include binary rainfall displays[3] that map varying numbers of bits to pixels, parallel coordinate and scatter plots[2] between various protocol fields, and ASCII packet information. RUMINT was a primary inspiration for the previous incarnation of Savors and several of its views and pause/step capabilities were emulated. The Visual Firewall[13] shows four views of network traffic including parallel coordinate and glyph-based scatter plots between ports and IP addresses, a network throughput graph, and a quad-axis diagram mapping IP address, subnet, IDS alert type, and time. VIAssist[4] show multiple views of network flow data including parallel coordinate plots, various charts, a table lens, and summarized dashboard view. All of these tools provide useful visualizations but can only process a single data stream at once so cannot correlate views across multiple files or hosts.

A limited number of visualization tools support the multi-data multi-view model. VisTrails[1] allows the user to create a matrix of related visualizations using a common sequence of visualization steps applied to different data sets or different portions of the same data set. Lourenco et al. present a prototype visualization tool[15] that can show scatter plots, pie charts, treemaps, and parallel coordinate plots of multiple data sets with matching of data columns to allow common filtering and colorization. Snap-Together Visualization[17] allows users to combine different views of a relational database and specify various coordinating actions between views such as loading data in one view when items are selected in another. A similar tool is Improvise,[22] which allows sophisticated coordinations of actions such as selection, loading, and visualization of data to be specified via coordinated query graphs. None of these tools are designed to handle real-time monitoring, however, where the data is in constant flux and the complexities in their coordination models do not lend themselves well to impromptu visualization.

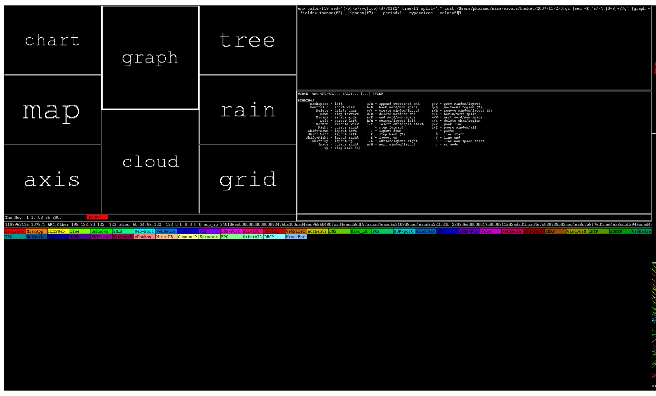

Figure 1. Savors console

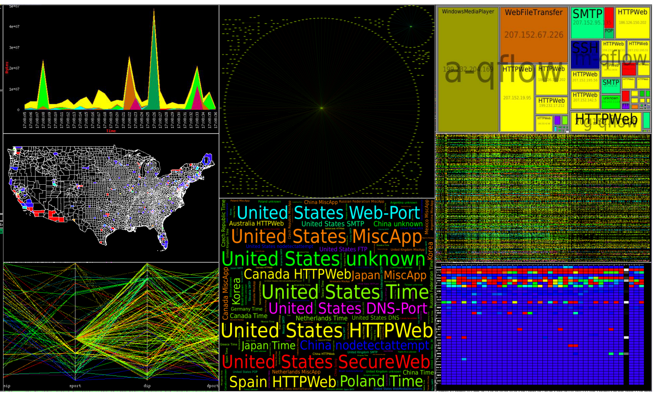

Figure 2. Display corresponding to console of Figure 1

Data analytics tools such as Splunk[20] and the Elasticsearch ELK stack[7] can ingest data streams from a variety of sources into a searchable centralized database, which can be visualized in configurable web dashboards. The expressiveness of database queries and flexibility of dashboard panels allows for full support of all combinations of single/multiple data/views. While these systems offer powerful capabilities, their push-based data model does not always scale and they require installation of additional software on each data producer when the data stream does not support a standard forwarding mechanism such as syslog or SNMP. Savors instead uses a pull-based data model with information retrieved from the source on demand and aggregated on-the-fly, allowing data to be kept at its original location until needed and retrieved without additional software beyond existing ssh daemons.

3. CONSOLE

Savors consists of three components: a *console*, one or more *data servers*, and one or more *views*. The console is responsible for user interaction, spawning data servers and views according to given command pipelines, and controlling synchronization between data streams. The data servers are responsible for spawning and interacting with the commands that generate data, manipulating the data as specified, and sending the data to the console and views. Finally, the views are responsible for visualizing the data as specified on one or more displays.

Figures 1 and 2 show a sample console window and its associated views. The console contains the current view layout in the upper left, a vi-like line editor and context-sensitive help in the upper right, and information about the active view such as color mappings and data time/contents at the bottom. Savors is based on a model similar to that of GNU screen where the display consists of one or more layouts that each contain one or more regions, which themselves each house one or more windows, with each window corresponding to a single view. Multiple windows can be created in each region with shuffling between them. Regions may be split both horizontally and vertically allowing arbitrary layouts to be created. Like windows, multiple layouts can be created with shuffling between them allowing a completely new set of windows to be swapped in with a single keystroke.

When a view is initialized, the console is responsible for spawning the view, finding and/or spawning the requested data servers, and sending the appropriate messages to all involved entities to connect the view with the data server(s). If a data server already exists that has the same specification as one given, the new view will be hooked into that server midstream instead of spawning a new server. This both conserves resources as the same data stream can be used by multiple views, as well as provides the synchronization for the single-data multi-view model as all views with the same data specification will get their data from the same source at the same time.

The final responsibility of the console is to provide synchronization in both the multi-data single-view and multi-data multi-view models. In these models, multiple concurrent processes generate arbitrary data asynchronously at unknown intervals so the data servers must be explicitly directed when to provide their data to subscribed views. Savors uses time as the synchronization medium so data servers are allowed to progress according to the chronological order of their data. Data servers send every data line to the console first and only send the data to subscribed views after the console has sent a ready notification to the server with the oldest data. The user may specify different synchronization groups, allowing multiple time periods to be running at once.

4. DATA SERVERS

Each data server is responsible for spawning a given command pipeline (minus the view portion), transforming its output according to a set of *data directives*, and sending the resulting data lines to the console and any subscribed views. Views expect data lines formatted as fields separated by whitespace with the zeroth field containing certain items parsed out by the data server such as the time. Fields are referenced using the shorthands *fi* (e.g. `f2`) for a single field, *fi-fj* (e.g. `f3-f5`) for a range of fields, or *fi,fj* (e.g. `f7,f8`) for a list of fields. Since the command output ingested by a data server may be in any format, user-specified data directives guide the server on how to transform output. These directives are specified using an *env* command at the beginning of the pipeline with the type of each directive as the variable name and the contents of the directive as the value (e.g. `env time=f1-f3`).

To gain an understanding of the directives that might be needed to natively handle the output of arbitrary commands, a number of commonly used status commands were examined for common output patterns. Four main characteristics of interest were identified including how time is associated with the output (per line: 34%, per loop: 13%, untimed: 53%), whether headers were included from which labels could be parsed (yes: 25%, no: 75%), whether fields were fixed within every data line (78%) or whether they could vary in location and/or content (22%), and whether the command runs once and exits (47%) or (optionally) loops continuously (53%).

To support time variations, a *time_grep* directive can be used to identify the line in each loop with the time in it, which the *time* directive is then evaluated against before being parsed into a Unix time. To support header labels, a *label_grep* directive can be used to identify the line with the labels in it. A *label* directive may optionally be specified to indicate the mapping between labels and fields when not an ordered one-to-one mapping. Since headers and other output extraneous to the main data lines may occur in output, *grep* and *grep_v* directives may be used to include and exclude lines, respectively. To support varying field locations/contents, a *sed* directive can be used to rewrite lines as needed and a *split* directive can be used to split up lines with differing field separators. Any parsing not possible with built-in directives can be done with an external program in the command pipeline.

To support commands that only execute once, a *repeat* directive can be used to rerun the command at a given frequency. When repeat is used with untimed commands, all data within the same invocation is associated with the same time. This avoids the problem with looping untimed commands where the data stream may block waiting for an older stream resulting in inaccurate times given to data lines still in the pipe. Savors supports the save and playback of data streams using the *tee*/*tee_a* and *replay* directives, which write/append to a save file and replay such files at a given speed multiplier, respectively. Save files differ from the original data streams in that all data transformations have already been carried out in the saved data. This is required to reproduce some contextual information only found in the raw data streams such as timings associated with untimed commands.

Multi-data single-view data streams can be specified in two different ways. The *data* directive can be used to create any number of interleaved parallel data streams with the special variable *fD* replaced with each value wherever specified (e.g. "`data=1-n ssh hostfD`" can be used to create similar data streams on n hosts). A second construct "`(cmd`$_1$` & cmd`$_2$` & ... & cmd`$_n$`)`" can also be used to specify multiple parallel data streams where each cmd$_i$ may be a full command pipeline with its own directives. This allows completely different commands to be fed into the same view. To better support this model, a *cut* directive can be used to selectively splice and reorder the list of fields to normalize the field contents across differing commands. The "data" directive and parallelization construct are actually carried out within the console, which generates and spawns the full set of data servers required. Also carried out within the console, is the *sync* directive, which allows synchronization over multiple periods of time and/or disabling of synchronization. Data servers are only synchronized with other servers in the same sync group allowing both real-time and historical data to be visualized concurrently.

5. VIEWS

The final command in each Savors pipeline describes how to view the data stream(s) created by the data server(s). It consists of a view type for the command name and any desired options applicable to that view. Savors currently supports eight primary view types (axis, chart, cloud, graph, grid, map, rain, and tree) with most primary types also having various subtypes. Each view is a separate process that communicates with one or more data servers and the console. Separating the views from the console allows for support of multi-host displays such as the

the NASA Advanced Supercomputing (NAS) Division's hyperwall[18] by allowing views to be spawned on the backing systems. Separating the views also provides greater extensibility by allowing views to be implemented in languages other than default (Perl/Tk) as long as they implement a simple TCP protocol directing them to change size/iconification, connect to a server, receive data and labels, and exit.

The options accepted by each view differ according to type, but the main options common to all of them are *--fields*, which specifies the list of expressions to visualize, *--color*, which specifies how the resulting objects should be colored, *--ctype*, which specifies alternative colorization schemes such as a common hash, *--grep*, which allows views to show different items from a common data stream, and *--period*, which specifies how much time (within the data) to wait until visualizing the next batch of data lines. Savors follows the methodology that simple things should be easy to specify while complex things should be possible. In this way, the fields and color options can be built from either simple field terms (i.e. `--fields=f2-f4`) or full perl expressions (e.g. `--fields='use File::Basename;dirname(f9)'`), which allows significant flexibility while supporting the more common simple case easily. Various convenience functions are supported within the perl expressions for IP address geolocation, host resolution, and anonymization of IP addresses, host names, and user names.

Savors supports multi-host displays allowing single views to be split arbitrarily across physical displays of differing resolutions. To support such displays, each component host computes all views of which it is a part at full relative resolution (i.e. the resolution of the full-size view if all displays were the same as the local display). The host then uses an appropriate X11 `-geometry` option to offset the origin of the view such that it will only display the portion for which it is responsible. While this results in inefficient CPU and memory utilization, it is a general-purpose solution that was easily implemented without the need to modify each individual view type.

To more easily spawn multiple related views across the same or different displays, two additional view-related data directives are available. The *view* directive is similar to the "data" directive in that it can be used to create any number of views with the special variable fV replaced with each value wherever specified. These views are treated as a unit within the console and arranged according to the value of the *layout* directive using vertical splits (e.g. `layout=1-1`), horizontal splits (e.g. `layout='1|1'`), and grids (e.g. `layout=2x3`) with arbitrary nesting. This feature allows parameter studies across different variables to be trivially created. For example, the construct "`view=1-8 layout=4x2 ssh hostfV`" can be used to create a 4x2 grid of views that each show the same visualization of different hosts (such as the bottom of Figure 7). These directives also support "console-less" operation, where a given layout of specific views can be invoked in their own windows without the interactive console window.

6. EXAMPLE USE CASES

Savors is a general-purpose framework that has been designed to operate with many forms of data in many different domains. This section illustrates just a few of the possible use cases of Savors in the domain of system administration with emphasis on its multiple data stream parallelization and synchronization capabilities.

6.1 Configuration Validation

Configuration management systems greatly reduce the burden of maintaining large computer installations but cannot guarantee that running configurations are as expected due to run-time changes by administrators, errors in the management configuration itself, and/or accidental omissions of particular systems. With its ability to easily spawn parallel commands and automatically aggregate the results, Savors can assist in validating running configurations across many hosts. For example, consider the file systems mounted on a host, which, even with the same static configuration file, may differ due manual mounts/unmounts by an administrator, mount options changed in response to errors, inability to mount properly due to network failures or file server misconfigurations, etc. Figure 3 shows a sparse adjacency matrix between the mounted file systems and their mount options colored by file system across eight cluster front-ends that are meant to be equivalent. When the mount configuration is consistent, each column would have eight blocks of the same color indicating that the option represented by the column is present on the file system represented by the row and that the file system is mounted on all eight hosts. In this case, differences can be seen in four of the file systems. This same technique can be used to quickly spot other configuration differences across large numbers of hosts.

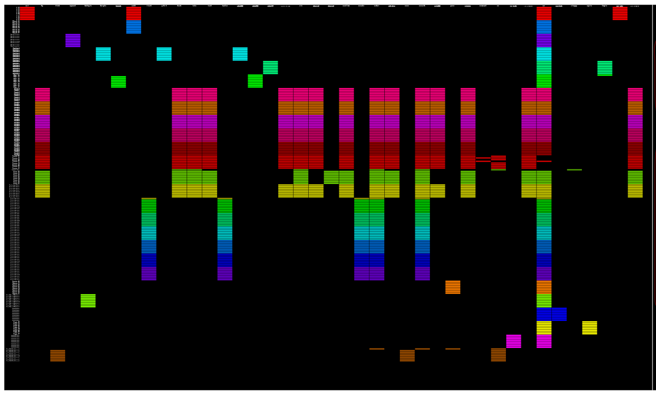
Figure 3. Grid of mount options colored by file system

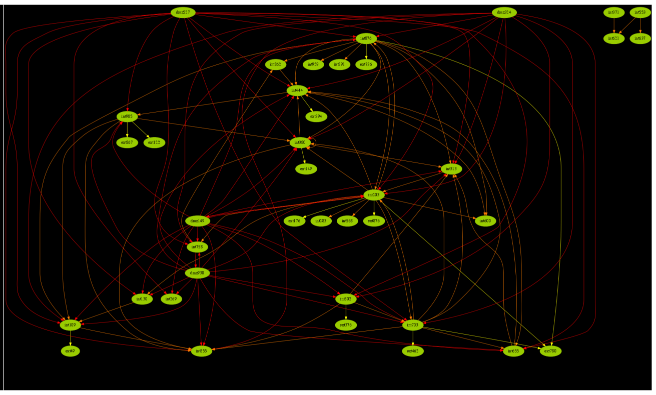
Figure 4. Graph of ssh connectivity

Even when a configuration is consistent between systems, it may not necessarily implement its desired intent. For example, consider an ssh access policy where internal hosts may connect with any internal, DMZ, or external host, but external hosts must first pass through a DMZ bastion to access internal hosts. Truly validating that the relevant configuration files across all systems and devices comply with this policy may take a security expert considerable time. A lightweight sanity check of this policy against the current network state, however, is easily achieved by Savors in a few seconds. Figure 4 shows a graph of ssh connections between the three types of systems based on output from the `ss` command across 14 cluster front-ends. The graph is directed from the origin of each ssh connection to its destination and colored red if it involves a DMZ bastion, orange if it involves two internal hosts, and yellow if it involves an external host. While it does not provide true assurance that the policy is correctly implemented, it does suggest that internal hosts are able to access all types of hosts, connections can, for the most part, be traced back to a DMZ bastion, and, for this instant at least, no external host is directly accessing an internal host.

6.2 Historical Log Analysis

Finding the root cause of a failure in a large installation may involve scouring the log files of many different systems in the relevant timeframe to try to identify the problem that occurred first. Savors can greatly simplify this process because it can pull log data directly from any of the involved hosts/devices and display events in the exact temporal order that they occurred. For example, an error accessing a file on a Lustre file system may be caused by issues such as a crash of the local client, a failure of the interconnect to the servers, or a bug on the server. Figure 5 shows three text rainfalls of Lustre errors in historical syslog files (seven hosts interleaved on top and two hosts on their own on bottom). By pausing the data streams and stepping through them in the order of occurrence, finding the first indication of a problem becomes a trivial exercise. Unlike other approaches where logs must be consolidated into a single database, which may not always be feasible due to scalability and/or reliability concerns, the pull-based model of Savors allows logs to be streamed directly from the source and aggregated dynamically on-the-fly.

Besides root cause analysis, logs are also valuable in identifying various patterns and/or anomalies to assist with items such as capacity planning and intrusion detection. Humans can, in many cases, spots patterns and anomalies more easily than machines as long as the data is presented properly. Because of its time synchronization capabilities, Savors is particularly adept at visually aligning logs collected over fixed intervals for finding patterns across time periods. Figure 6 shows seven stacked line charts, each of which depicts the network I/O rate (from the `collectl` utility) over the same portion of different days shifted into the same context to allow regions of interest such as those with similar shapes or periods of repeated high utilization to be quickly spotted. From the figure, regions of similar activity start around 2am and 3am, which correspond to daily backups that occur at the same time every morning on the transfer bastion on which these logs were collected. This type of view would not be possible without time synchronization as the underlying `collectl` streams may run at different rates depending on the amount of activity in the logs being replayed. The automatic synchronization of Savors, however, allows this same technique is be easily applied to any logs over multiple time periods.

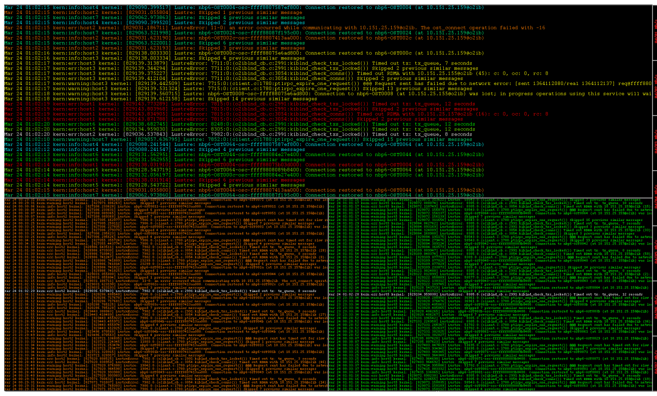
Figure 5. Rainfall of Lustre syslog errors colored by host

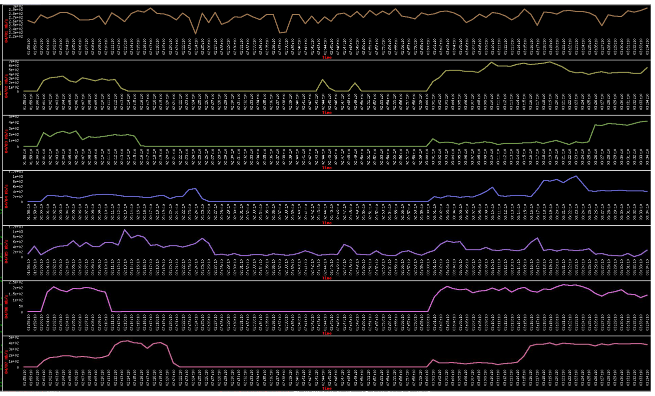
Figure 6. Charts of network activity per day

6.3 Real-Time Monitoring

Monitoring the state of large installations is essential for determining how resources are being used, predicting resource exhaustion, and resolving resource conflicts between users. For example, monitoring CPU and memory utilization on front-end systems can help administrators determine if users are performing work on front-end systems that should be done on back-end compute resources, if the amount of installed memory is sufficient for common tasks, how to better balance the load between high utilization users, etc. This kind of information does not typically reside in files, but is instead queried from kernel memory using standard system tools such as `top` and `ps`, which is ideal for the stream-based model of Savors. Figure 7 shows two stacked views representing different levels of detail about CPU and memory utilization across eight systems generated from `ps` data streams. The top view is a word cloud of user/command pairs colored by command and sized by cumulative CPU time. This view quickly shows which processes are consuming the most CPU time across all systems (`MATLAB`, `idl`, and `shiftc`). The bottom view is a composite hive plot showing low-level stats about the individual processes on each of the systems, which makes it possible to see outliers and clumping within the data.

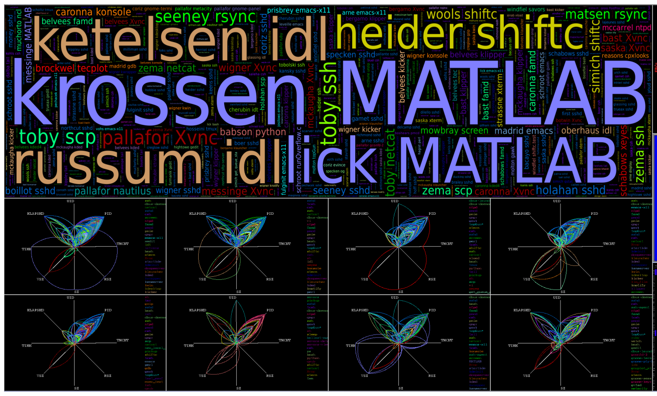

Figure 7. Word cloud/hive plots of CPU/memory usage

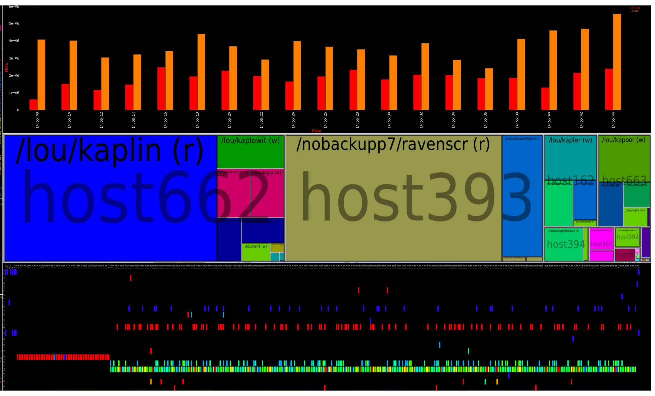

Figure 8. Chart, treemap, and grid of file system utilization

Even more critical than monitoring CPU and memory utilization is monitoring distributed file systems shared by large numbers of resources since a file system that is full, down, or overutilized may cause CPU cycles to be wasted when job results cannot be written to disk or overall job performance is decreased due to I/O contention. Many file systems have no way of determining global load besides monitoring the I/O of all the individual systems that mount them, which is a task that Savors can easily handle. Figure 8 shows three stacked views representing different levels of detail about file system activity generated from two NAS-developed utilities called `fstop` and `ltop`, which show file system activity on one host and Lustre activity across all hosts, respectively. The top view is a bar chart of front-end I/O generated from user commands compared to total I/O of a single Lustre file system, incorporating data from `fstop` on 10 hosts and `ltop` on one host that was normalized into a form consistent across both and aggregated into a single view not possible otherwise. The middle view is a treemap showing the directories generating the most front-end I/O activity across each host, which provides an understanding of the relative I/O of each host, the file systems with the most activity, and the users generating that activity. The bottom view is a sparse heatmap showing contention between the disks of all Lustre file systems by user, where multiple blocks colored in the same column indicate contention for the corresponding disk.

7. CONCLUSIONS AND FUTURE WORK

This paper has described a new framework called Savors, **S**ynchronization **A**nd **V**isualization **O**f a**R**bitrary **S**treams. Savors allows the normal command pipelines used by administrators to be visualized in a variety of ways. Commands can be trivially parallelized across multiple hosts with the resulting data streams synchronized by time within the same or different views. Savors was designed to support an impromptu model of visualization where visualization becomes just another standard tool in the administrator's workflow of monitoring and analyzing system components without the need to centralize or reformat data streams ahead of time. Savors is open source and available for download.[19]

There are a number of directions for future research. The console should be given the ability to save/restore layouts as is already possible for individual views. Likewise, the ability to step backward should be added as is already possible in the forward direction. Also needed is a better mechanism to display static data, which can currently be emulated using a repeat directive, but should be a direct capability. There are a variety of additional views that could be incorporated including dendrograms, voronoi diagrams, particle simulations, and bubble charts. Support for a large graph library and a faster canvas should also be investigated.

REFERENCES

1. L. Bavoil, S.P. Callahan, P.J. Crossno, J. Freire, C.E. Scheidegger, C.T. Silva, H.T. Vo: VisTrails: Enabling Interactive Multiple-View Visualizations. 16th IEEE Conf. on Visualization, Oct. 2005.
2. G. Conti, K. Abdullah: Passive Visual Fingerprinting of Network Attack Tools. 1st ACM Wkshp. on Visualization and Data Mining for Computer Security, Oct. 2004.
3. G. Conti, J. Grizzard, M. Ahamad, H. Owen: Visual Exploration of Malicious Network Objects Using Semantic Zoom, Interactive Encoding and Dynamic Queries. 2nd IEEE Intl. Wkshp. on Visualization for Computer Security, Oct. 2005.
4. A.D. D'Amico, J.R. Goodall, D.R. Tesone, J.K. Kopylec: Visual Discovery in Computer Network Defense. IEEE Computer Graphics and Applications, 27(5), Sept./Oct. 2007.
5. C.R. Dos Santos, P. Gros: Multiple Views in 3D Metaphoric Information Visualization. 6th IEEE Intl. Conf. on Information Visualization, Jul. 2002.
6. S.G. Eick, M.C. Nelson, J.D. Schmidt: Graphical Analysis of Computer Log Files. Comm. of the ACM, 37(12), Dec. 1994.
7. Elasticsearch ELK stack. `http://elasticsearch.org/overview`.
8. A. Frei, M. Rennhard: Histogram Matrix: Log File Visualization for Anomaly Detection. 3rd IEEE Intl. Conf. on Availability, Reliability, and Security, Mar. 2008.
9. E. Garduno, S.P. Kavulya, J. Tan, R. Gandhi, P. Narasimhan: Theia: Visual Signatures for Problem Diagnosis in Large Hadoop Clusters. 26th USENIX Large Installation System Administration Conf., Dec. 2012.
10. L. Girardin, D. Brodbeck: A Visual Approach for Monitoring Logs. 12th USENIX Large Installation System Administration Conf., Dec. 1998.
11. H. Hochheiser, B. Shneiderman: Using Interactive Visualizations of WWW Log Data to Characterize Access Patterns and Inform Site Design. Jour. of the American Society for Information Science and Technology, 52(4), Jan. 2001.
12. P.Z. Kolano: A Scalable Aural-Visual Environment for Security Event Monitoring, Analysis, and Response. 3rd Intl. Symp. on Visual Computing, Nov. 2007.
13. C. Lee, J. Trost, N. Gibbs, R. Beyah, J.A. Copeland: Visual Firewall: Real-time Network Security Monitor. 2nd IEEE Intl. Wkshp. on Visualization for Computer Security, Oct. 2005.
14. Q. Liao, A. Blaich, A. Striegel, D. Thain: ENAVis: Enterprise Network Activities Visualization. 22nd USENIX Large Installation System Administration Conf., Nov. 2008.
15. R.A.d.M Lourenco, R.V. Guimaraes, N.J.S. Carneiro, et al.: Exploring, Comparing and Coordinating Multiple Datasets in an Information Visualization Tool. 16th IEEE Intl. Conf. on Information Visualization, Jul. 2012.
16. A. Makanju, S. Brooks, A.N. Zincir-Heywood, E.E. Milios: Logview: Visualizing Event Log Clusters. 6th IEEE Conf. on Privacy, Security, and Trust, Oct. 2008.
17. C. North, B. Shneiderman: Snap-Together Visualization: A User Interface for Coordinating Visualizations via Relational Schemata. 5th Intl. ACM Working Conf. on Advanced Visual Interfaces, May 2000.
18. T.A. Sandstrom, C. Henze, C. Levit: The hyperwall. 1st Intl. Conf. on Coordinated and Multiple Views in Exploratory Visualization, Jul. 2003.
19. Savors. `http://savors.sf.net`.
20. Splunk. `http://www.splunk.com`.
21. T. Takada, H. Koike: MieLog: A Highly Interactive Visual Log Browser Using Information Visualization and Statistical Analysis. 16th USENIX Large Installation System Administration Conf., Nov. 2002.
22. C. Weaver: Building Highly-Coordinated Visualizations in Improvise. 10th IEEE Symp. on Information Visualization, Oct. 2004.
23. W. Yurcik, R.S. Thompson, M.B. Twidale, E.M. Rantanen: If You Can't Beat 'Em, Join 'Em: Combining Text and Visual Interfaces for Security-System Administration. ACM Interactions, 14(1), Jan./Feb. 2007.