Using Visualization in System and Network Administration

Doug Hughes
Auburn University

Using Visualization in System and Network Administration

Doug Hughes – Auburn University

ABSTRACT

Unix systems have numerous tools that generate copious amounts of data about performance, security, process status, networking and sundry other things; this usually results in the output of raw numbers or text. Visualization of this data can lead to useful insights. Examples include: graphing performance data, customizing tools to make a complex operation easier to understand, and correlating log events according to user-defined rules or patterns. Using visualization in daily activity can help the brain recognize normal and aberrant behavior, make complex tasks easier, and improve workplace efficiency.

The goal of this paper is to detail how simple rapid-prototyping, GUI tools can be used to quickly develop applications to visualize data. This paper also describes the application of visualization to areas of system and network administration. I will attempt to illustrate my own results and experiences developing visualization tools in Tcl/Tk [1] (with various extensions). The design and implementation of four such tools will be discussed and used as examples of visualization applied toward system and network administration. Some have surpassed their original design criteria and provided unanticipated side-effects that enhance their utility.

Introduction

Though the precise definition of visualization is often debated, it is customarily used as a way to express information in two or three dimensions. The advantage of visualization is that it takes raw data and manipulates it such that recognizable patterns begin to emerge, or presents it such that a certain task becomes easier to understand or more efficient. Visualization can also be used for prediction and intuitive troubleshooting, once sufficient data has been collected. Through the use of rapid prototyping GUI tools like Tcl/Tk, Perl [2]/Tk, Python, etc. one can develop simple visualization tools that can help in various system and network administration tasks.

By nature, visualization works best with a GUI interface. According to Jakob Nielsen [3], there are five aspects to usability of a GUI interface: easy to learn, efficient, easy to remember, relatively error free or error forgiving, and pleasant to use. Because of the user-definable nature of the tools that I will be describing, and the target audience (myself, my co-workers, and others in my technical field), I will be concentrating on the first three aspects. This may cause the tools at times to be plain in appearance and unforgiving when used in unexpected ways, but future iterations and extensions will work to correct any deficiencies. All these tools have been designed in a language which makes individual tailoring quite easy.

Currently Auburn University College of Engineering uses several tools written in Tcl/Tk to indicate health and performance of networks and machines, to make certain tasks more manageable, and to troubleshoot when problems occur. The tools I will be focusing on include a security analysis tool, a server CPU pie-chart tool, a SPARCstorage Array disk visualization, placement, and optimization tool, and briefly, a hub analysis tool.

Motivation

These tools have all been designed to make our jobs easier and more productive. Without these tools, perusing the raw data streams would be a time consuming and frustrating process. They help us to always be aware of events as they happen. Being a part of a support organization, it's important to at least give the illusion to your users that you know about a problem before they report it. If only the phone would answer itself when a server went down...

tklogger

We use TCP Wrappers [4], klaxon, tocsin [5], and other customized daemons and programs to log security events to a centralized, secure, limited-access machine via syslog. Originally, all of our syslog data was simply posted on the console of a machine as it arrived, without any organization or correlation possible. This resulted in one of two undesirable situations. First, if the console window was on top, a portion of the screen space was being used; if nothing was happening, that screen space was wasted. Second, if the console was in the back, the information was largely lost behind another window.

Tklogger was developed as a real-time security analysis tool. The goal was to have a way to correlate high priority and low priority traffic based on syslog data files and regular expression matches. It was determined that events should also be displayed in multiple colors to facilitate their interpretation at a glance and without having to scrutinize the text (color event classification). Subsequently, a quick glance at the window could determine the security status of the hosts and the network. It was also decided that when important events occurred, the window would automatically come to the front. This way the administrator using the machine (myself) would not always have to keep a third eye on the console, nor would useful screen space be wasted. Other features such as search capability, scrollback, and easy configuration were added as needed. It was originally constructed a few hours a day over the course of a week. Little modifications were made for the first few months, but the general functionality remained relatively unchanged.

cpupie

Cpupie was inspired by an object oriented Tcl/Tk pie-chart extension called tkpiechart [6]. Out of this simple demo came the idea for representing the CPU states of our servers as pies. The already widespread practice of breaking up a CPU into four components (idle, wait, system, and user) made its use in this capacity immediately apparent. Thus, displaying the CPU trends of our servers was imagined (and since proven) to be a useful symptom of server health. Mr. Fontaine (the author of the pie-chart widget) and I collaborated on the features of tkpiechart to get it to its present state. Excluding that work, the initial design phase for cpupie was approximately 16 hours.

ssa

Ssa was born in an afternoon of frustration. We have two SPARCstorage Arrays [7] attached to two servers. At the time, each had 12 disks that were almost completely filled with approximately 14 file systems on RAID-5 volumes. Also at the time, three new disks had come in for each array to add more file system capability to our home directory servers.

Since the array model that we own is already divided into six SCSI controllers of up to five devices per controller (Figure 1), it is convenient to purchase disks in groups of six to maximize the advantages of a six disk RAID-5 stripe. After we exhausted the capacity of our existing disks we were left in a bit of a conundrum: how do we take three disks, add them to the array, and arrange them such that no two stripes share a disk, busy stripes do not share a controller, and all disks are evenly used without overburdening the busy stripes? (The astute reader may note that with three new disks it is probably impossible to not have at least one volume that has two stripes on the same controller using a six disk stripe given that all disks will be evenly utilized.)

Without this tool, we would have had to write all configurations on a white-board in text with volume sizes and layout then manually arrange things by erasing, moving, and updating all totals without being able to see the final results until implementation. This approach was fraught with peril, was essentially brute-force and may have taken us hours for each array. The ssa tool was written to speed up this process for this and all future instances of new disk arrival, removal, or failure. The first drag-and-drop, user-hostile version was made available in four hours. Since the creation, our decreased time spent calculating movements has more than made up for this design time.

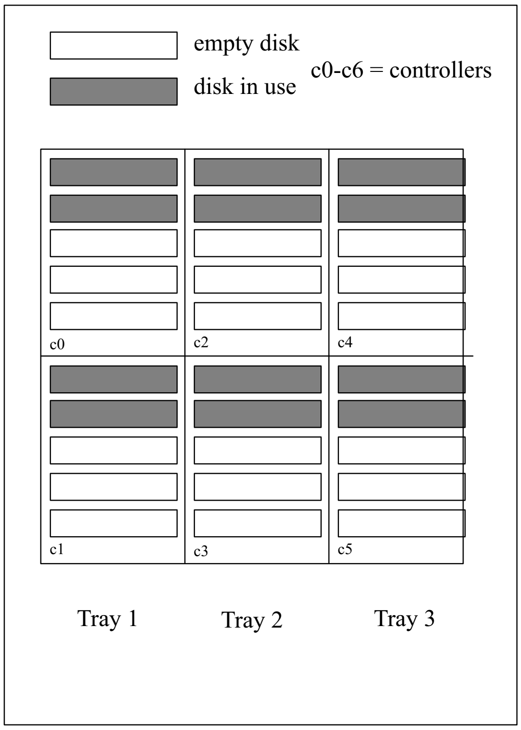

Figure 1: Array physical layout

hphubwatch

Finally, hphubwatch was written at a time when slowness problems were being experienced on several of our 30 networks. We required tools to monitor and analyze our HP hubs, but our budget was extremely tight. This tool uses SNMP [8] to gather information from HP AdvanceStack hubs and display it graphically in real time. It was written in a few hours and is fairly small and simple.

Program Implementation and Usage

tklogger

As already mentioned, the output of tklogger is contained in two windows. The latter contains high priority events and the former contains low priority events. An example window layout is in Figure 2. Normally, each text line would be displayed in the color associated with that event. An event is represented in user-defined colors to enable easily recognizing the type of event when it occurs without actually reading the corresponding text. In this way, a user can tell at a glance when something bad has happened or when an unusual pattern occurs. Further investigation may be warranted, particularly if an alarm color is displayed (anything matching *red* is high priority).

The idea for using colors to represent events occurred to me one day while experimenting with a Tcl/Tk text widget and seeing how easily it could be configured to display lines of text in multiple colors and/or fonts. The idea for using a Tcl/Tk text widget to display time sensitive logging information in colors blossomed out of that.

Augmenting the visualization is a search capability that allows one to highlight matches in place or display them in a separate window. Menus are available for users to adjust scrollback capability, save events to a file, reload the configuration file, adjust the search options, and perform other actions. One can also pause the polling to examine an event more closely before resuming.

When an alarm event occurs the window immediately de-iconifies itself if necessary and rises above all other windows on the screen. A recent addition has allowed it to also perform other actions such as sending mail, paging, ringing the keyboard bell, or executing a user defined function written as a Tcl procedure. We have tklogger running at all times monitoring and displaying various events.

With proper configuration, tklogger can be used to monitor many log files concurrently. Each file may be given a base priority for all records. These base priorities can then be overridden with regular expressions. A Sample configuration file is shown in Figure 3. **File** directives specify the log file to poll. **Color** directives give a priority (high or low – based on color choice) that is the base priority for events in that file. All events appended to the end of a file will have this base priority color. Files that do not have a **color** directive will be examined for **match** expressions. Regular or fixed expression **match** directives are used to override any applicable base priority, or to execute a command. **Ignore** directives are used to elide base priority information that may not be necessary (e.g., debugging information from sendmail).

```
file auth /var/log/authlog
file daemon /var/log/daemon
file local0 /var/log/local0.info
file local1 /var/log/local1.note
file local2 /var/log/local2.warn
file local3 /var/log/local3.note
file maillog /var/log/maillog
color local0 forestgreen
color local1 lightseagreen
color local2 magenta
color local3 red1
color auth red2
ignore NOQUEUE
match dlam red1
match mooneje {email page-doug
    {.cshrc accessed}}
match help {playsound
    /home/ens/doug/sounds/chord.au
    orange}
match {LOGIN FAILURE} mediumvioletred
match (pgcntd|refused) red4
match portwatcher red3
match (vrfy|expn) violetred
```

Figure 3: tklogger rules

I would be remiss if I did not compare the usage of tklogger with two other popular log analysis tools. Contool [9] provides a way to perform actions when certain messages appear on the console, but lacks in areas such as event grouping and searching capability. Swatch [10] is another extremely useful and extensible

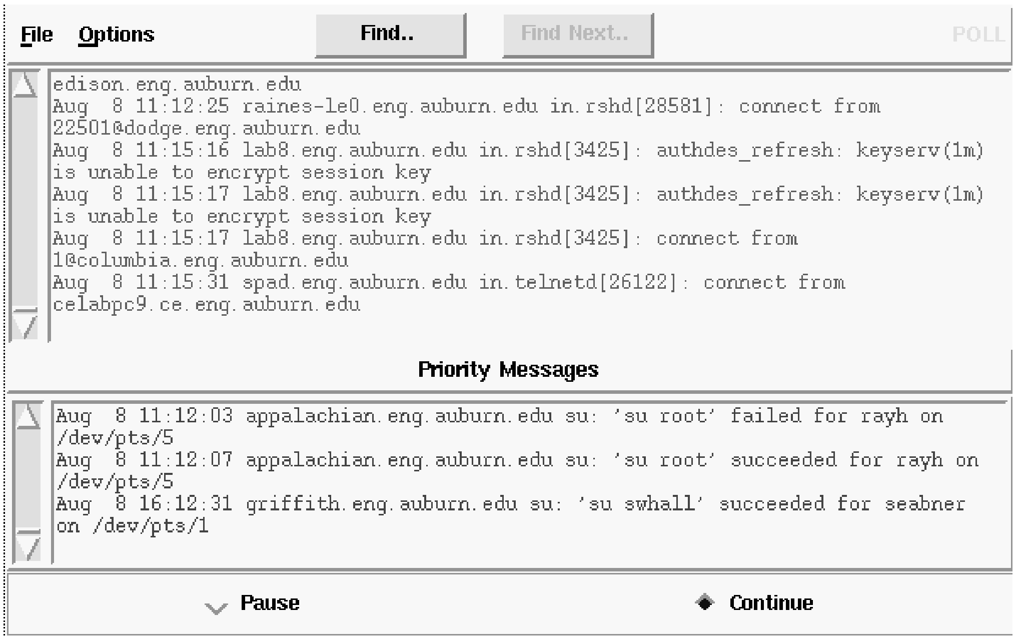

Figure 2: tklogger

log analysis tool written in Perl, which gives it all the power of that language. However, swatch is meant to be run in the background and I was looking for something more visual. Both tools are meant to process one input source at a time.

cpupie

To analyze our server CPU states, cpupie (see Figure 4) uses the *rstat*(3) portion of the scotty [11] Tcl/Tk extension. Since all Unix platforms of which I am aware support *rstat*(3), the operating system independent nature of this tool was immediately attractive. The scotty extension also gave us the added advantage of client/server sockets so that only one machine was required to do the polling. Any other machines wanting to view CPU status of any subset of the servers could connect to the master machine for updates via a TCP/IP socket.

The cpupie program was designed with several simplistic features that we have found useful. The most commonly used features are available via buttons along the top of the window. It takes advantage of the native PostScript generation capabilities of the Tk canvas widget to output the current CPU states to color or black and white PostScript printers at the touch of a key. It has the capability to average the CPU states of all the servers over a user defined time interval. Finally, the polling interval and listening socket (for client/server operation) are user configurable. Cpupie is constantly being run by several people on two-headed workstations to monitor our servers.

ssa

Like cpupie, the ssa tool (Figure 5) is also implemented with a canvas widget for easily printing the current array layout. The most basic unit of operation on the canvas is a color filled rectangle. The disks are large rectangles, and each stripe (subdisk) of a disk is a smaller color filled rectangle. The stripes of a particular volume are all the same color across all disks. Mirrors of a volume are also the same color, but are stippled with a bitmap to indicate that they are mirrors and not the primary disk(s). Log disks have the same stipple as the mirrors do, but are usually invisible because of their tiny size, so a feature was added to artificially increase their size to identify disk location.

In order to understand the use of the ssa tool, one must also understand some of the nomenclature specific to the Veritas software that drives the SPARCstorage Array. Each physical disk is mapped to a VM disk which is divided into up to 16 subdisks via a virtual table of contents (VTOC). Subdisks are then amalgamated into a unit called a plex. This can be a RAID-3, RAID-5, or striped plex (in fact the basic functionality of a plex is analogous to a stripe in the conventional sense). These plexes can then be mirrored, concatenated, or combined with a log into a volume. In our case a file system goes onto the volume, though some places use them for raw databases partitions. The relationship between disks, subdisks and plexes is illustrated in Figure 6.

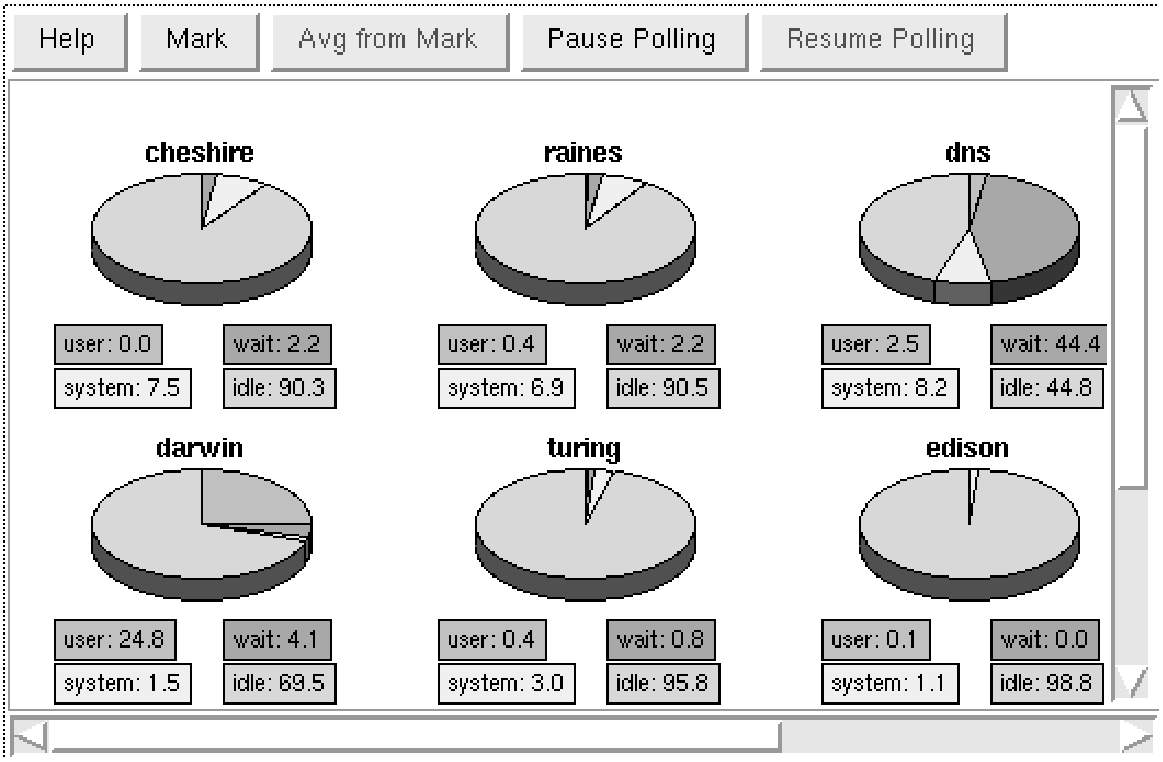

Figure 4: cpupie

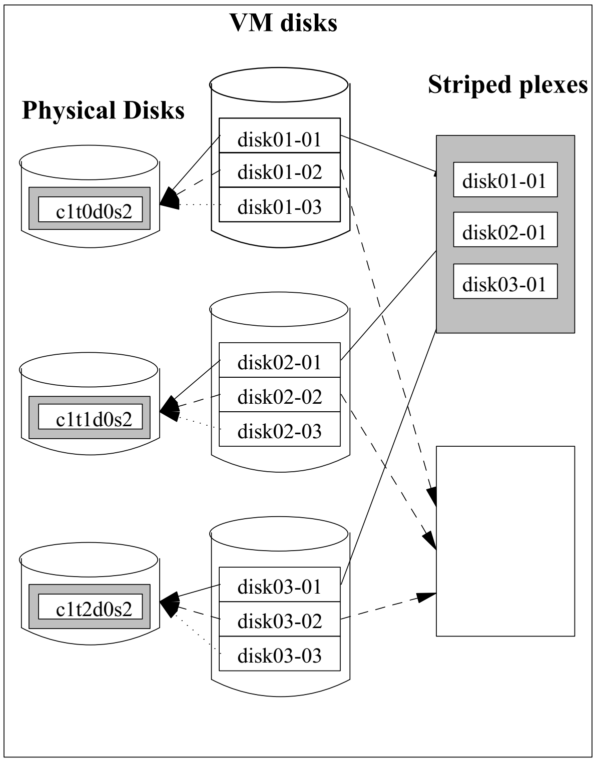

Figure 6: Volume Manager terminology

For people who don't use the Veritas Volume Manager (VxVM) GUI interface, a button also displays the sequence of commands necessary to configure the array when it has been arranged as desired. This tool will not be valuable to people who do not have the Veritas software installed, but it is capable of running without an array.

The use of this tool is a bit more complex then the previously mentioned ones. Subdisks can be dragged from one disk onto another. New empty disks can be created with the click of a button to simulate the addition of a disk to the array. Also, there is an undo function, a way to determine the size of a subdisk by clicking on it, and a way to determine the used and free space on a physical disk by clicking on it with mouse button 2. This tool is very VxVM specific, but it has served us quite well by allowing us to fine-tune placement of file systems on our SPARCstorage array and experiment with different configurations before implementation. It serves as a useful example of using visualization to simplify complex tasks.

hphubwatch

The hub watching tool is also implemented using scotty and, like tklogger, makes use of the color capabilities of the text widget. A portion of the window is shown in Figure 7. It currently assumes one machine is connected per port. The three columns following the MAC address correspond to administrative status, operational status, and media status respectively. The rest of the columns are explained in the legend of the figure. Information that has changed is highlighted in yellow. In the figure, the number of frames has changed on ports 5, 7, 8, and 9 (yellow) and the percentage of collisions to frames on port 8 and 9 is greater than 0 (red). Information that is deemed

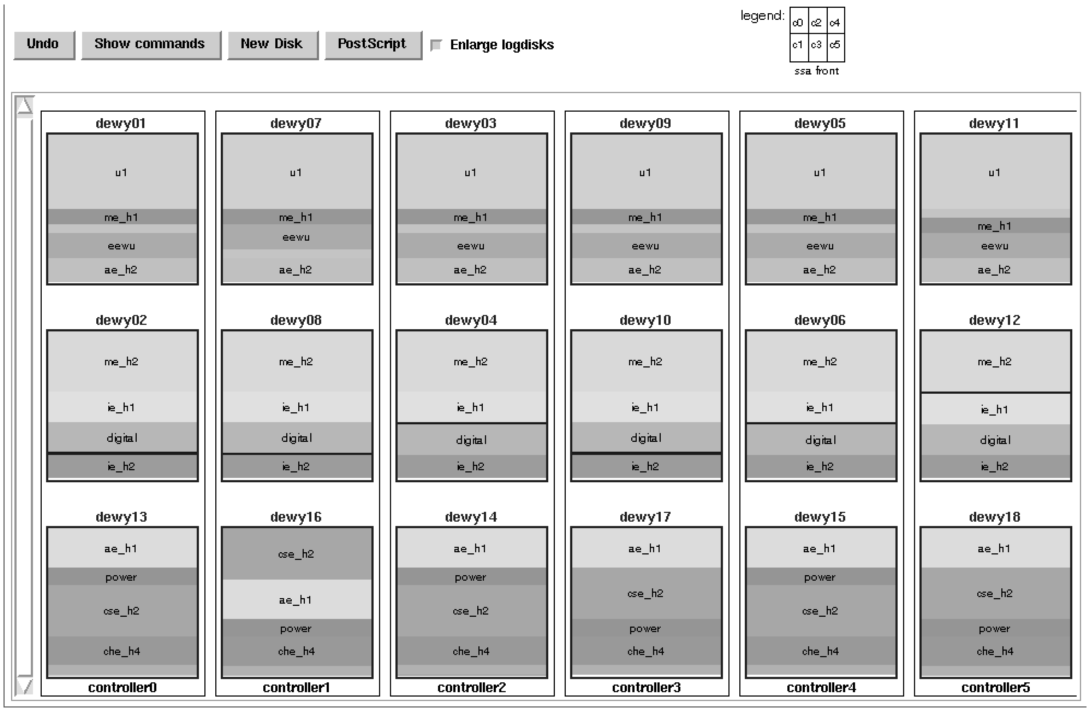

Figure 5: ssa

significant is highlighted in red. This information includes giants, jabbers, alignment errors, and other events of this nature which might indicate problems. The polling interval is user specified and the highlighted regions change after each polling interval. It is a very simple tool with narrowly defined operating parameters. We use it to troubleshoot networks through their hubs when problems occur.

Visualization Results

All of the tools have lived up to my expectations of their design. There have been serendipitous surprises as well. The paragraphs below will attempt to outline the expected results as well as describing how some of our expectations have been surpassed.

tklogger

Tklogger has already proved its usefulness at detecting many different types of events in progress including mail spam, port scanning, user account cracking, and misconfigured daemons. When a high priority event occurs, it has been beneficial to search back to find low priority events that may provide correlative information. We even have people using it for different purposes. I use it to monitor security while another person uses it to monitor WWW logs.

The multiple input file functionality has proven to be the most useful (non-visualization related) feature by allowing syslog generated events to be organized in multiple files by facility and priority. Tools that generate syslog information have the information forwarded to the secure log-host where it is stored in files according to simple syslog configuration rules. This facilitates searching and archival of logs based on administrator defined event groupings. This has been considerably easier than dealing with one large file.

cpupie

Cpupie began as a fun project to do with Tcl/Tk that could tell us generally how busy our servers were. It has since proven to be a useful forensic tool for diagnosing potential server problems before they happen. The CPU states convey information about other pieces of the system indirectly such as memory and disk usage.

The color green on any CPU is intuitively obvious. Our brains have been trained from childhood that green indicates that all is well. In this case green represents idle time. A green CPU tells us the machine has room to grow.

Yellow corresponds to system activity. Typically, yellow on our CPU's is 5-25% of the pie. When yellow gets higher than this, it is usually an indication to us that there is either a lot of process activity occurring (a daemon possibly gone amok) or a lot of memory in use for some reason. Our license server typically has 15% of its single CPU in yellow.

When a CPU exceeds 15% in the red state, we investigate. This usually indicates that there is a large amount of disk activity occurring. On our mail server this can be normal if a large mail list is being processed. On any other servers this usually means the machine is involved in some swapping or paging. Subsequent investigation leads to a culprit or the resignation that more memory is needed.

There is an interesting distinction between the user CPU state (displayed in blue) and the other three states. While resource usage typically balances across all CPU's, CPU time spent in user is easier to visualize on one processor of a machine. Therefore, on a machine with four CPU's, a CPU with 50% blue should be interpreted as 100% of two CPU's in use. On our quad-CPU machines the affect of this

```
A = Media Available                    N = Not Available/Not Operational
O = Operational                        E = Administratively Enabled

Med = Media Errors                     Al  = Alignment Errors
Gi  = Giant Packets                    Rn  = Alignment Errors
Pt  = Partitional Port                 JP  = Past Jabbers
P   = Port Polarity                    I   = Intruder
FPS = Frames per second                %C  = Percentage of collisions/frames
```

	Mac Address				Frames	Octets	Med	Al	Tot	G	R	Collis	Pt	JP	P	I	FPS	%C
1	00:00:1D:04:A3:C4	E	O	A	96717	12320617	0	0	0	0	0	23481	0	0	0	0	0	0
2	00:00:00:00:00:00	E	N	N	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	00:00:00:00:00:00	E	N	N	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	00:00:00:00:00:00	E	N	N	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	00:00:C0:05:88:15	E	O	A	3689803	682703787	2	0	2	2	0	447600	0	0	0	0	1	0
6	00:00:1D:02:16:A3	E	O	A	57633	5160102	0	0	0	0	0	25559	0	0	0	0	0	0
7	00:00:F4:C8:BF:B1	E	O	A	199516	18354766	0	0	0	0	0	71680	0	0	0	0	9	0
8	00:00:F4:D0:32:62	E	O	A	536727	96050741	149	0	149	0	19	27493	0	0	0	0	1	1
9	00:00:F4:CA:6F:9D	E	O	A	55620	4788085	0	0	0	0	0	22812	0	0	0	0	10	1
10	00:00:00:00:00:00	E	N	N	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	00:00:00:00:00:00	E	N	N	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	00:00:00:00:00:00	E	N	N	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 7: : hphubwatch

demarcation is readily apparent. The blue slice of the pie is almost always at one of five levels: 0, 25, 50, 75, or 100. We can tell at a glance which compute servers would be candidates for new jobs. As our users write code to take advantage of multi-threading in modern operating systems, these distinctions may be less visible. In Figure 4, the machine **darwin** is a four CPU compute server with one CPU servicing a CPU-bound process and **dns** is a mail server (processing a mail list as it would happen).

ssa

Since our first addition of disks, we have learned that it is much easier to buy disks in groups of six to add to the arrays. However, we still use the tool when new disks arrive. In order to not overburden any controller or disk, we try to distribute things as evenly as possible. This means that when new disks come in we need to move subdisks and adjust free space to provide balanced access across controllers and disks. It has gone through several small enhancements since its creation in November of 1995 and is now under general release to help other users of Veritas Software around the world.

The SPARCstorage array tool makes our jobs a lot easier by allowing us to see at a glance where space is available. VxVM is very flexible at letting you customize your views of the array, but is not designed to allow one to experiment with new disk/volume layouts. It is a production tool. Once you commit to an action, you must wait until it is completed. With the storage array tool we have been able to reduce our configuration time from hours to minutes, save a lot of white-board space, print out (graphically) a picture of the current array layout, and see the actual commands that will be executed by VxVM when we are done.

hphubwatch

We have already identified several high traffic machines in work-groups on particular subnets and endeavored to further segment these networks, or install switches when required. In addition the tool has discovered, via high numbers of media errors, several old, flat, gray-satin cables that people had used to plug their computers into the network. It has also discovered the occasional improperly punched-down house wiring that has reverse polarity. In the hubs where security is enabled, it lets us track intruders on ports where undergrads may be trying to plug in their laptops or otherwise access the network in an unauthorized manner.

Conclusions

In my experience the genesis of new visualization tools has been 50% need and 50% inspiration. One of our most useful performance tools (cpupie) was inspired by a simple demo. Another (tklogger) was created as an experiment in log analysis. A third was created to meet a specific goal (ssa). The remaining tool presented in this paper (hphubwatch) was created as an experiment and out of the desire to manage our hubs. All four tools have been in use for 6-18 months.

I have found that using different colors is a convenient way to express desired characteristics. The brain can distinguish between thousands if not millions of colors. More is not necessarily better, though. Experiment with your color choices and use those that work best on your application in your environment. I have experimented with fonts in different pitches and styles (e.g., italic vs. bold), but they do not have the versatility of color. Combining font changes with color changes has worked well. When you do use colors, try to use consistent strategies among tools. (i.e green is good, yellow may need investigating, red should be checked on immediately). Lastly, try to arrange contrasting colors between similar colors (stick a purple between your light seagreen and your medium seagreen).

Visualization tools help the system and network administrator to make sense out of complex data. Most any program that generates statistical or performance data can be visualized. Using visualization can provide a revelation out of an otherwise rudimentary set of unprocessed data. Patterns may begin to emerge which make management of resources, and trend analysis easier.

Visualization also can make complex tasks (e.g., file system rearrangement on a group of disks) easier by displaying the problem in an easier to understand or more intuitive format. Tcl/Tk has been particularly suited to these tasks by providing an easy to learn interpreted language with excellent publicly available contributions, and extensibility via C and C++.

Acknowledgments

I would like to thank my boss, Steve Henderson, for giving me the freedom to construct the tools about which this paper is written. I would also like to thank all the contributors of feedback and enhancements for their input. Finally, I would like to extend my deepest appreciation to all of the writers of the extensions to Tcl/Tk that made these tools possible, particularly Jean-Luc Fontaine, Karl Lehenbauer, and Jüergen Schöenwälder for contributing tkpiechart, TclX, and scotty respectively.

Availability

All tools will run on any Unix platform where Tcl/Tk and the relevant extensions can be compiled (which is most of them). Tklogger is self sufficient and should be able to run on Windows and Macintosh platforms as well (with minor modifications regarding file naming conventions). All tools are available for anonymous FTP at ftp.eng.auburn.edu in the pub/doug directory. A sample configuration file for tklogger (.tkloggerrc) is available on this server as well.

Tklogger is also available at the Coast security archives (http://www.cs.purdue.edu/coast). Documentation for each tool is included inside the file near the top.

Author Information

Doug Hughes got started administering Sun systems while attending Penn State University. He graduated in 1991 with a BE in Computer Engineering. From there he spent three years in GE Aerospace before and after the merger with Martin Marietta (pre-Lockheed, post-RCA) doing everything from software development to networking, systems administration, and internal consulting for client-server projects. From there he went to Auburn University College of Engineering where he now resides as the Senior Network Engineer. His interests include writing programs in scripting language to ostensibly make his job easier. He can be reached via U.S. Mail at 103 L building, Auburn, AL 36849, or via electronic mail at Doug.Hughes@eng.auburn.edu .

References

[1] John K. Ousterhout, *Tcl and the Tk Toolkit*, Addison Wesley, Reading, Mass, April 1994.

[2] Larry Wall, Randal Schwartz, *Programming Perl*, O'Reilly and Associates, Sebastopol, CA. 1991.

[3] Jakob Nielsen, "Iterative User-Interface Design", IEEE Computer Society, *Computer*, Vol 27, n7, July 1994, pp. 32-41.

[4] Wietse Venema, "TCP Wrapper: Network Monitoring, Access Control and Booby Traps", *Proc. 1992 USENIX UNIX Security Symposium*, pp 85-92.

[5] Doug Hughes, *klaxon and tocsin – detecting port scanning*, 1995-96, unpublished tools, available via FTP at ftp.eng.auburn.edu:/pub/doug/ .

[6] Jean-Luc Fontaine, *a Tcl/Tk pie utility*, 1996, available via ftp at ftp.neosoft.com in /pub/tcl /alcatel/code/tkpiechart-*.tar.gz .

[7] Sun Microsystems Inc., *Understanding Disk Arrays*, white paper, 1994.

[8] Marshall T. Rose, *The Simple Book, 2nd edition*, Prentice Hall, Inc., Englewood Cliffs, New Jersey, 1994.

[9] Chuck Musciano, *contool*, 1994, available via ftp at ftp.x.org:/R5contrib/ .

[10] Stephen E. Hansen, E. Todd Atkins, "Automated System Monitoring and Notification With Swatch", *Proc. Nov. 1993 USENIX LISA*, pp 145-155.

[11] Jüergen Schöenwälder, "scotty – a Tcl interpreter with TCP/IP extensions", *Proc. 3rd USENIX Tcl/Tk Workshop*, Jul. 1995.