The following paper was originally published
in the Proceedings of the
Tenth USENIX System Administration Conference (LISA X)
Chicago, IL, USA, Sept. 29 - Oct. 4, 1996

Visualizing Huge Tracefiles with Xscal

Alva L. Couch
Tufts Universit

Visualizing Huge Tracefiles with Xscal

Alva L. Couch – Tufts University

ABSTRACT

Xscal is a tool for visualizing the behavior of large numbers of comparable entities such as computers, routers, processes, users, etc. The input to Xscal is a table of numbers and strings, which can represent a system snapshot at an instant of time or an event trace of system activity over time. Xscal depicts the global distribution of values in one or two columns of that data. The user can then annotate the visualization to denote categories of behavior, after which Xscal can depict detailed information on entities in each category. Originally developed to study massively parallel computation, Xscal is applicable to a wide variety of system analysis tasks, from visualizing the process table or accounting statistics for a large server to studying the performance of a large network.

Introduction

Very often there is a problem with a large system or network that requires scanning a large amount of data for the answer. This data may be console output, process statistics, accounting data, or network performance measurements. While statistics on the data may be useful in isolated cases, in practice one must actually read the detailed data in order to reach a conclusion. Although very little of this data needs to be read, it is usually mixed in with much other data of little relevance. Our problem is to locate that data and interpret it to reach a conclusion, both quickly and reliably. Usually, this involves locating unusual behavior or occurrences in a sea of normal behavior and occurrences. Visualization is a particularly powerful technique for locating such unusual behavior.

There are many existing approaches to visualizing log files, all of which attempt to preserve the character of the log file as a stream of events. The paired tools `tcpdump` and `tcpview` [8] allow the analyst to dynamically filter the stream of output from a network monitor to eliminate packet records of little interest. The user interactively decides what events to keep in the event trace. `SeeLog`, part of AT&T's software visualization suite `SeeSoft`, produces several depictions of a program log file. Events are colored in accordance with their type, so that the user can see at a glance where interesting events might be recorded. There are many other tools like `tcpview` and `SeeLog`, all of which view a logfile as a series of events and utilize color mapping, filtering, and zooming to navigate the event stream. But all of these tools are relatively specific to particular problem domains, such as networking,

Several very powerful and general-purpose visualization tools can be adapted to analyze networks if one is willing to interface system data to them. My parallel computation analyzer, Seeplex [2], and its relatives, Paragraph [4] and Pablo [7], can generate a variety of views of system performance for a network. However, one must first translate system data into a form compatible with their data formats. In the case of Seeplex and Pablo, this is possible but quite involved. Paragraph is too domain-specific to parallel computers to generate much useful information. Survey data shows [6] that many intended clients for these tools consider their use too involved to be worth the effort of learning, even users working in the tools' original problem domain. Clearly, system administrators need something easier to use.

Xscal

Xscal is a visualization program that can display and analyze data for thousands of entities such as machines, networking hardware, or phone switches. Originally designed for analyzing massively parallel computations on networks of thousands of cooperating processors, Xscal is equally useful when we need to comprehend very large snapshots or trace files from a server or network. Xscal graphically depicts the global distribution of data, allows the user to group data by behavior, and exhibits original data by group. There is no limit to input file size except the physical memory of the host system, the speed of the processor, and the patience of the user. I have visualized 50000-line and larger files on a 128-megabyte Sun SparcStation-10 using this tool. Xscal is an X11 tool written using the freely available Openwin X11 toolkit, and is portable to a majority of X11 systems.

Xscal differs from its predecessors in several ways. Xscal emphasizes data views that remain usable regardless of the size of the data file. Scale-sensitive views that change in usability as the number of entries increase are de-emphasized. Xscal reads the simplest possible input format, unlike other tools with moderately to extremely complex requirements for input format. While the emphasis in other tools is simply upon depicting the data, Xscal allows the user to graphically annotate data to group or separate behaviors and can depict further views based upon that annotation. In modern parlance, Xscal is a 'highly interactive' data browsing interface.

Using Xscal

Input to Xscal is a table of columns of data delimited by spaces and tabs, where each column entry is either a number or an ASCII string. Depending upon how Xscal is invoked, the data file is interpreted either as a snapshot at an instant of time or as an event trace over time. In a snapshot file, each line of data describes everything known about a particular entity, where each column is assumed to contain commensurate measurements for the same kind of data. A trace file embodies many snapshots by describing each entity at several instants of time. Each trace file must have two columns, designated by the user, that denote which entity is being described and what time the data was recorded. All other columns are assumed to contain measurements or state information that become valid at the stated time and may be changed by the next entry (in time) for the same entity. Lines in the trace file are sorted into time order by Xscal and need not be sorted into any particular order beforehand.

Unfortunately, Xscal only knows how to read integers and floating point numbers, so that it cannot parse times expressed in date format. Entity descriptions, however, can be any text, including non-numeric strings. Below we will discuss ways to adapt output of common UNIX tools to Xscal's format.

Xscal's trace format is complex enough that a simple example may clarify it. The event trace file:

```
" time user processes
   1  couch 20
   7  sofa  10
  10  couch 10
  15  sofa   5
```

means the following:

- Before time 1 nothing is known about either `couch` or `sofa`.
- From time 1 to before 10, `couch` has 20 processes.
- From time 7 to before time 15, `sofa` has 10 processes.
- From time 10 to the end of time, `couch` has 10 processes.
- From time 15 to the end of time, `sofa` has 5 processes.

In other words, the input file above corresponds to the following sequence of actual values, where values in each line are valid until the next time:

time	couch procs	sofa procs
1	20	-
7	20	10
10	10	10
15	10	5

This format may seem complex, but it was derived from much experience designing and parsing much more complex forms [3]. I believe it to be the simplest possible form to which any event data can be converted.

Xscal's Views

In either snapshot or trace mode, Xscal obeys a consistent strategy in displaying data. First it depicts a global representation of all measurements either at an instant or over all time and awaits user input. The user may then choose colors and textures to designate particular ranges of values. Xscal can then generate subsequent views depicting statistics and detailed data on selected ranges. In particular, if a range of values indicates undesirable behavior, one can immediately obtain full details on the conditions surrounding that behavior.

Viewing Snapshots

In snapshot mode, every line of the input file represents a summary of measurements for a distinct entity. For a chosen column of the data, Xscal produces a depiction of both values and frequencies of data in that column (Figure 1). The vertical axis represents value while horizontal axis represents individual entities. On the vertical axis, there is a hash mark for each distinct data value. On the horizontal axis, entities are sorted into order of increasing value from left to right, and there is a hash mark between each two groups of entities with identical data values, so that whitespace between hash marks indicates the frequency of a particular value.

These two axes are combined into an X-Y plot that represents the distribution of values for an entity. This plot is created by plotting a point for each entity with horizontal offset equal to the entity's position on the horizontal axis and with vertical offset corresponding to its value on the vertical axis, filling in gaps so that these points become a horizontal line whose vertical offset represents a value and whose horizontal extent represents the value's frequency. These horizontal lines (of entity measurements) are joined end-to-end by vertical lines whenever values change, so that each vertical segment represents the difference between two adjacent values in sorted order. The line thus drawn always proceeds from bottom left, simply increasing, to top right of the diagram.

The shape of this 'characteristic line' indicates the global distribution of values for all entities.

1. A diagonal proceeding from bottom left to upper right denotes that values for all entities are uniformly distributed (but perhaps randomly distributed) over a given range of possible values.
2. A 'step' which begins at bottom left, proceeds horizontally to the middle, then vertically to top middle, then along the top to the right, denotes that there are two distinct values that occur with roughly equal frequency, one small and one larger.

3. A horizontal line denotes that all values are the same.
4. A line that rises quickly at the left and then levels off denotes that most values are relatively large, because the frequency of larger values is greater than that for smaller ones.
5. A line that stays low and rises near the right denotes that most values are relatively small.

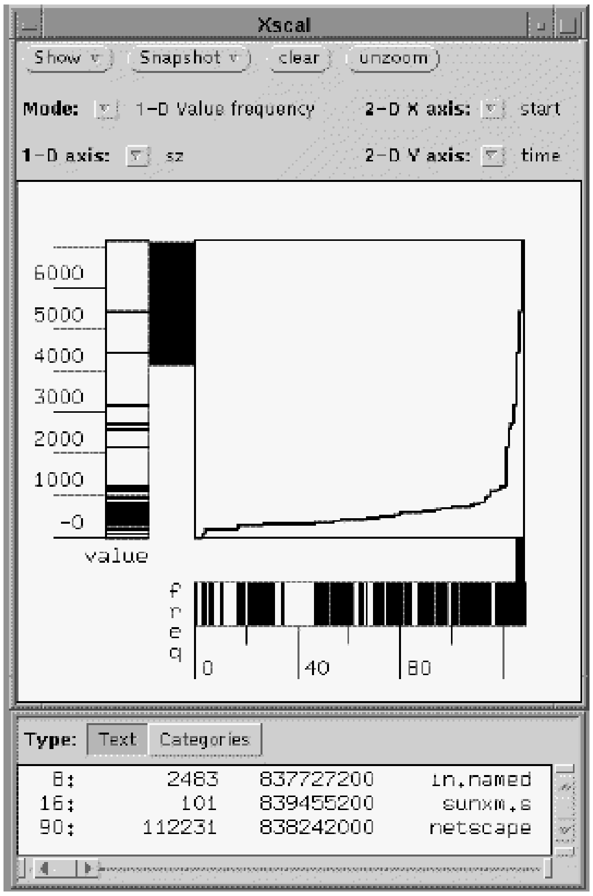

Figure 1: A snapshot of server memory usage based on ps command output.

The 'characteristic line' shows data distribution regardless of the structure of input values. If entities assume a small number of discrete states, it looks like a staircase where the width of each stair indicates the frequency of that state. If entities assume many distinct values, the line looks like a continuous curve.

Be not deceived, however: this depiction is very different from a normal X-Y plot. In particular, a characteristic line for a linear sequence of values looks almost identical to a characteristic line for white noise. By sorting measurements, the snapshot view effectively suppresses our knowledge of noisiness in the data, in order to coherently and understandably depict the global distribution of values. Any view that depicted that noise (such as a standard X-Y plot without sorting) would obscure the data distribution.

This depiction is a 'figure of merit' [5] for one system parameter, in the sense that one can tell, at a glance, whether the distribution of the parameter is normal or exceptional. To fully understand this, consider what happens when each entity is a process on a timesharing server and the depicted value is memory utilization.

1. A diagonal proceeding from bottom left to upper right means that a given process is equally likely to have any memory utilization from small to large.
2. A 'step' which begins at bottom left and vertically up the middle means that half the processes are memory hogs (with the same memory utilization(!)) while the other half are low-impact.
3. A horizontal line means that all processes have the same memory utilization (and something is very wrong with your server).
4. A line that rises quickly at the left and then levels off means that almost all processes consume a lot of memory.
5. A line that stays low and rises near the right means that a few processes are hogs and the rest relatively low-impact. This is also the characteristic appearance of a statistically normal distribution of values with a small standard deviation.

From this diagram, one can tell at a glance what action to take to improve system performance if memory is in short supply.

- In cases 1 and 4, there is very little that can be done, because a large number of processes are contributing to the problem. It is time to buy more memory or servers.
- In cases 2 and 3, something is probably seriously wrong with the server and it is time to reboot. Perhaps there is a runaway self-replicating process.
- In case 4, very few processes are causing the problem so it is perhaps time to ask a few users to be more courteous.

Of course, one could determine this from looking at the output of `ps` or `top`, but scanning this diagram shows much more than a printout of numbers.

Note that since the impact of multiple processes is the sum of their memory utilizations, the *area under any curve segment* has units of memory utilization × frequency, which is cumulative memory utilization. This is proportional to the total impact upon the server for that segment.

Figure 1 depicts an actual Xscal depiction of memory utilization for all processes on a UNIX server. This graph is a trough curving upward from the base, implying that a few processes taking most of the memory (which is a fairly normal state of affairs). To determine what these processes are, we distinguish them by selecting a texture from a texture palette and painting that texture onto the region of the Y axis for which we wish detailed data. The figure shows the resulting black rectangle. Then we ask for a textual view of the data, shown below the global view in the figure. This view shows only the offending processes we selected, sorted by memory consumption. The text view identifies several memory hogs.

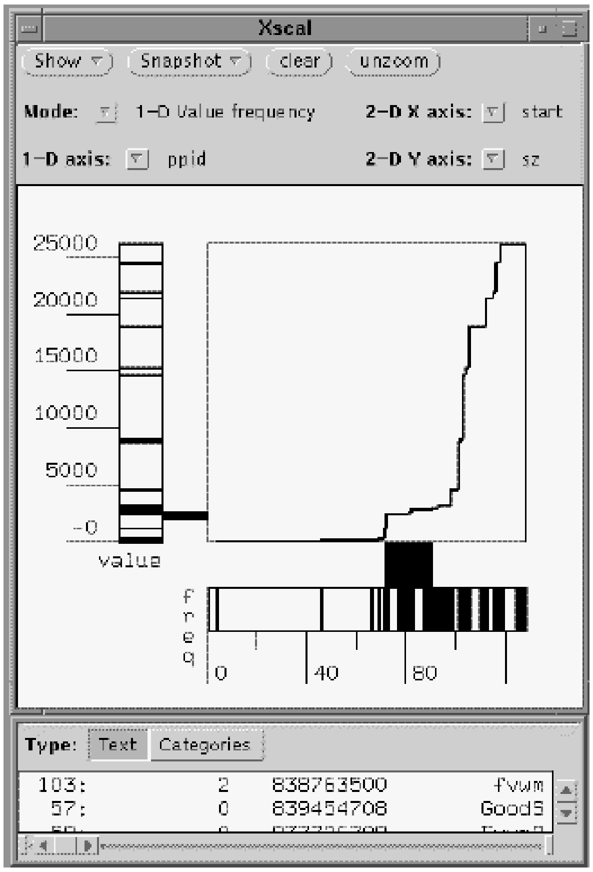

Figure 2: Displaying the distribution of parent process identifiers for all processes gives surprising insight into usage patterns for a server.

Figure 2 shows the distribution of parent process identifiers for all processes on a server. This distribution has an unusual shape consisting of multiple steps. Closer examination shows that each step represents an interactive session for one user, because all the user's processes are children of the same login process. Large steps are X11 sessions and small steps are telnets. Thus a datum that usually indicates little of interest when listed in a `ps` command, when shown in this way, tells one a great deal about how the server is being utilized.

The user can zoom in on details in the global view by selecting a rectangle of the view to magnify with the mouse. There is no limit to the detail of zooming, to the limit of floating point resolution. While zooming, scroll bubbles on the top and right hand sides of the view show one's relative position within the view.

Studying Correlations

Xscal can also be used to study the relationship or correlation between kinds of measurements. It does this using a standard two-dimensional scatterplot, where the user can choose the columns representing each axis. For example, Figure 3 represents a comparison of memory utilization and CPU time utilization for all processes, where memory utilization is depicted on the Y axis and CPU utilization on the X axis. Two-dimensional scatterplots can be explored just like one-dimensional distributions, except that one must select a rectangle of points for detailed study.

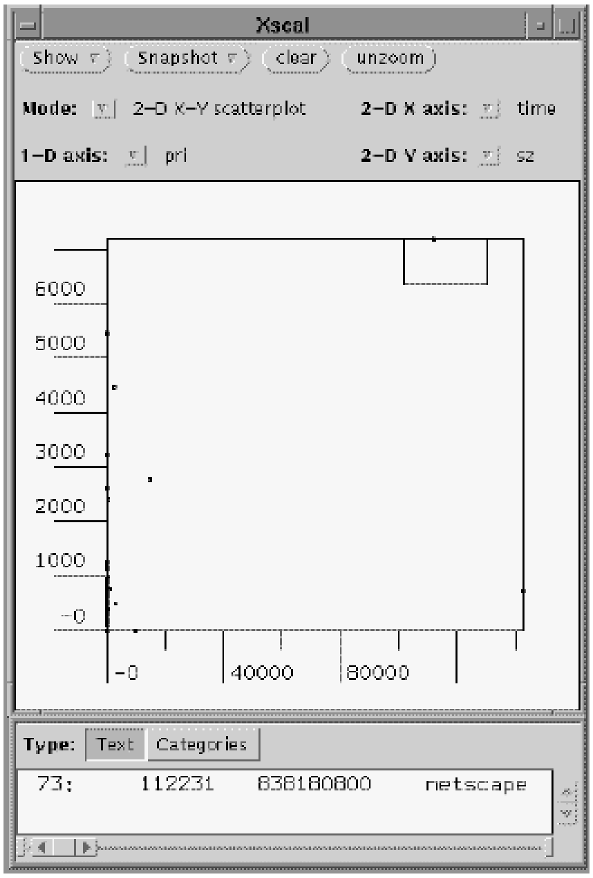

Figure 3: A two-dimensional scatterplot of memory utilization (Y) versus cumulative CPU utilization (X) for all processes on a timesharing server shows two runaway processes I would perhaps have missed otherwise.

When I casually generated this figure in writing this paper, I discovered that my server contained two runaway processes. The data point at the top right represents a process with extremely large memory utilization and total CPU utilization. I selected it by drawing a rectangle around it, so that detailed data was shown for it. Closer examination showed that it had been running for weeks. The data point at bottom right is also a runaway, but its impact is solely upon CPU usage, not memory.

Displaying Time Series

Xscal's second form enables display of time series data by interpreting input lines as events rather than states. Each line has a time when it occurred, an entity describing where it was measured, and a series of measurements for that time and place. There are no other restrictions; Xscal re-sorts input internally so it can be in any order, even backward. A measurement for an entity is valid from the time stated in its line to the time of the next measurement for the same entity.

Suppose we invoke Xscal on a login history, where entities are people (uids), time is wall clock time, and state is average memory consumption (from `acctcom`). Since the output of `acctcom` is not suitable for Xscal, I first convert it using a Perl script as described below. Of course, one can write the conversion script in any general-purpose programming language.

When invoked on a time series, Xscal first provides a scatter-plot of values versus time (Figure 4). The plot represents our total memory usage on a single server between 1 am and 11 pm on a single day. The vertical axis is again memory consumption, while the horizontal represents time. Entities are particular users. Each dot represents how much memory was being used, on average, by a particular user at a particular time.

To learn more about this plot, we click on a 'busy' time in the scatter-plot to reconstruct textual information and the value-frequency plot for the chosen time. These work just as if we were studying a single snapshot, but only show that subset of the event trace that determined measurements for the chosen time (events before and including that time). Using this view we mark two ranges of measurements for further analysis: we use the mouse to associate 'relatively high' memory utilization (in the upper third of the Y axis) with a dark texture, and 'moderate' memory usage (in the middle third) with a lighter one. These textures will then be used by Xscal to depict the behavior of each entity, both at an instant and over time.

Using this categorization, Xscal can show a frequency histogram summary of entity behavior over time (Figure 5, top right). In this histogram the horizontal axis is time and the vertical axis is frequency. The height of a textural element at a given time represents the number of entities having behavior corresponding to that texture, i.e., moderate or high memory utilization. While the global depiction of behavior seems to indicate a busy day, the thin dark line in the histogram shows that only at most three users are using a really large amount of memory at any time.

To exhibit behavior of particular users, Xscal can also depict a timeline for each entity (Figure 5, bottom

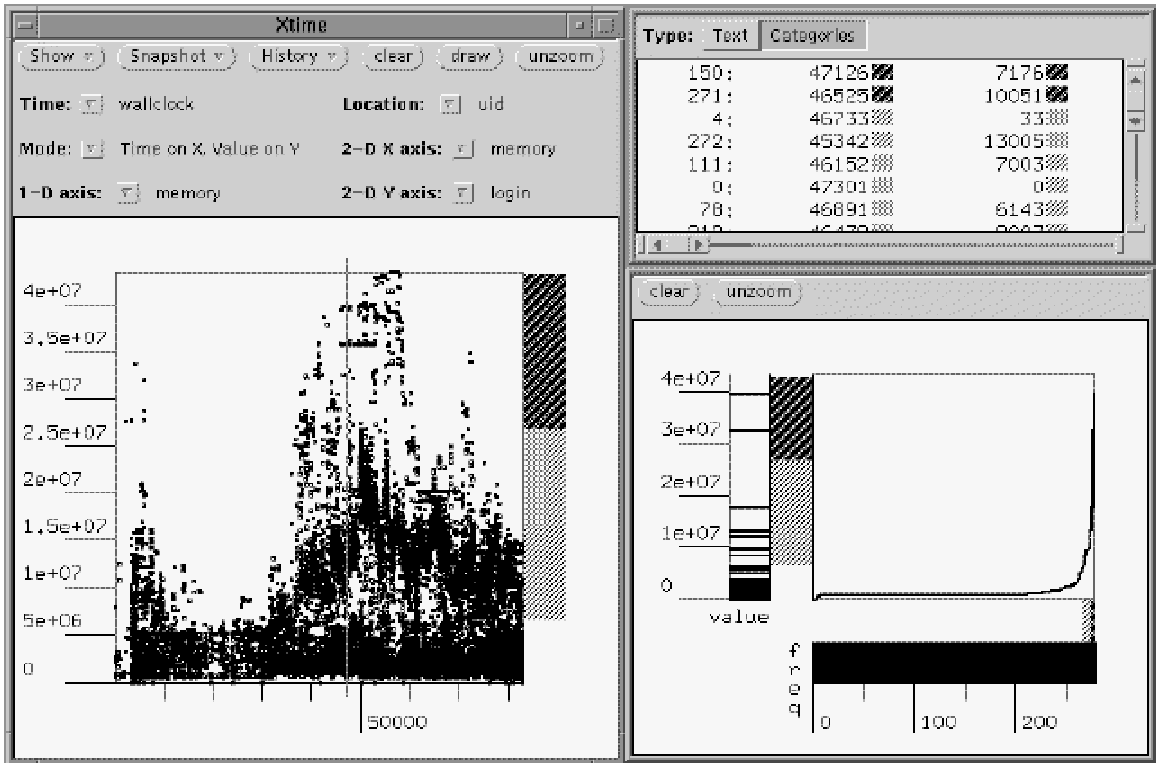

Figure 4: Average memory usage for each user on our server between 1 AM and 11 PM on a single day. At left is a plot of time on the X-axis against user memory usage on the Y-axis. Selecting a time reconstructs a snapshot of conditions at that time, shown at right in textual form and as a distribution.

right). This shows that although we might be concerned about the global graph, our users actually individually utilized a lot of memory for relatively short periods. It is perhaps time to buy a new server.

The strength of all of Xscal's views is that it is easy to distinguish relatively normal behavior from exceptional behavior and quickly detailed information on the exceptions. Using these views, we can very quickly answer the two relevant questions about a resource conflict problem: when and who!

Theory of Xscal

Xscal's views are based on principles discovered in my study of how to visualize the behavior of massively parallel programs [2]. One can create an infinitude of Xscal-like views by following several basic principles.

A *scalable view* is one whose form, size, meaning, and use do not vary with the amount of data it depicts. A view whose size changes is instead a *scrollable view*, because it eventually becomes larger than screen size and must be viewed in parts by scrolling. Any view that depicts distinct entities is by nature scrollable, so that scalable views must instead depict data distributions.

Using three basic building blocks, the time axis, the value axis, and the frequency axis (as depicted in the snapshot global view), one can depict many distinct scalable depictions. The value and frequency axes are typically both necessary and somewhat dual to one another. For a datum that takes only a few distinct values, frequency usually indicates more than value, while for a datum that takes many values, value shows more than frequency. Both are present in the basic global parameter view in order to effectively represent all possible data distributions.

While Xscal more or less exhaustively covers the possibilities for two-dimensional scalable views, these principles give rise to many useful three-dimensional views that Xscal does not yet provide. The snapshot views of a datum, viewed over time, form a surface where one axis is time, another value, and a third is frequency. The evolution of the parameter can be seen at a glance in this view. Another example is a two-dimensional scatterplot of values depicted over time.

Converting File Formats

Unfortunately, Xscal was designed to analyze traces of parallel algorithms, which can be made more consistent in format than standard system log files. Fortunately, Xscal can take input from a UNIX pipe, so that Perl scripts can be used to translate between UNIX output and Xscal's desires. Since Xscal does not limit the user to a particular number of columns of

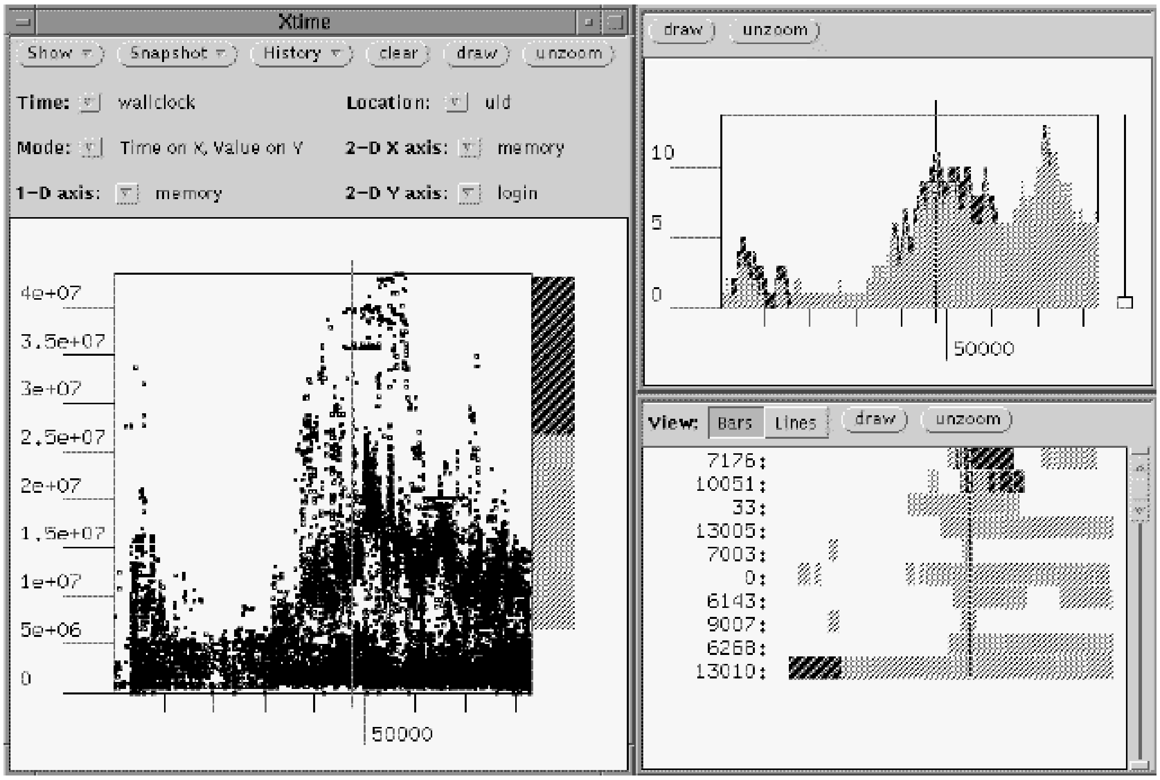

Figure 5: Given categorizations of memory usages as high (dark texture) and moderate (medium texture), Xscal can show category frequency over time (top right) or timelines for particular users (bottom right). We see, not at all surprisingly, that most memory utilization is due to the activities of three users.

input, a timely strategy is to tack new columns onto the beginning of each line that contain Xscal-readable input. In the future, I plan to make domain-specific modifications to Xscal itself that will allow it to parse such things as UNIX date strings for syslog visualization.

There are two translation problems, one of which is easy and the other of which is quite difficult. If the input file already has a consistent columnar form in which each column has a consistent meaning, translation is relatively straightforward. This is true for process data, accounting data, filtered network traces containing only one protocol, etc. If input is amorphous in form, with different meanings to the same column in different lines, more extreme measures are needed to translate it into depictable form. In many cases depiction is impossible due to the nature of the data.

Translating a columnar form into an Xscal input requires a relatively straightforward, if ugly, Perl script. To generate the `ps` examples above, I added fields at the beginning of each line representing the numeric values of strings that Xscal cannot interpret as numbers. Xscal reads dates and times as simple strings, so I translate all times into seconds since the Epoch (Jan 1 1970, 0:00am GMT), using the Perl5 library module `Time::Local`. It is also necessary to translate strings like '`10%`' into their decimal equivalents (0.10) because Xscal will interpret the former as a string. I also place placeholders in empty fields, and change spaces embedded in fields to underscores so that fields will not be split up by Xscal; '`Jul 20`' becomes '`Jul_20`'.

The `acctcom` example was much more difficult to generate. I had to write a script that computes the global history of memory utilization for each user and represents that history in Xscal form. This required translating process statistics provided by `acctcom` into user statistics. For each elapsed time reference, a user's average memory consumption is the sum of the average memory utilizations for all processes that user is executing at that time. This may sound easy to compute, but in practice naive methods take too much CPU time to execute and the only efficient way to compute this requires advanced data structures such as balanced binary search trees.

Examples of format translation scripts used in preparing this paper may be obtained at http://www.cs.tufts.edu/pub/xscal/lisa-96 .

Xscal Limitations

Xscal executes relatively slowly on large data files because it must initially sort and otherwise index the data. Its internal memory needs are roughly five times the size of the data file itself, in order to arrange for it to execute as quickly as possible. The input file to the last example contains 20000 lines of data and takes 20 CPU-seconds to read and render on an UltraSparc containing 128 megabytes of main memory. Once input is read, rendering becomes much faster; repainting the screen takes about five seconds after each user request.

It takes time to tell Xscal what to do. As indicated above, translating an output format into Xscal's input format can be very involved, or even impossible. Once the file is translated, one still must tell Xscal which columns to interpret as what. This may seem like a small price to pay, except when a pressing systems problem must be solved. Then it takes too long to set up.

Xscal does not treat string input as well as it could. A string is just another was to indicate system state, and Xscal should treat it as such. One way is to construct a value axis for all strings, in which offset of a string on the axis corresponds to the ordinal of the string in a sorted list of all strings. Xscal could easily do this, but does not at this point. Xscal's internal assumption that all strings are at most 10 characters long is also a serious problem. Xscal should treat each line of data as an integral unit in textual displays, and keep the format the same in those displays as in an editor.

Conclusions

In any search for abnormality in a complex system, the most difficult problem is to define normality. Normality is easier to observe, feel, and experience than to define. In turn, a tool that allows one to compare experiences, normal and abnormal, is usually more effective in analyzing problems than any attempt to define normality rigorously. Of course, one has to choose views that by their nature look different in normal and abnormal situations. This is the driving philosophy and motivation behind all visualization of complex systems.

Several times, Xscal has saved me time in looking at systems data. I initially tried it on accounting data to settle a dispute about server planning. One faction claimed that the University's mail server was slow because some students were abusing their privileges. Xscal showed immediately that not only were the students all using roughly the same amount of resources, but also that resource problems stemmed from use of the system for administrative tasks.

Xscal allows several kinds of comparative analysis to be done easily and quickly. However, there are absolute limits to the utility of any general purpose tool in this problem domain. To be utilized regularly, a visualization strategy has to be *really easy* to employ, and Xscal, while very powerful, is still too difficult to use regularly. Data conversion is relatively difficult, setup takes too much time, and there is no provision for heterogeneous data sets where measurements for differing entities are not necessarily comparable.

This last problem makes it quite difficult for one to use Xscal or its relatives to analyze a network. Sure, one can understand the function of 1000 X-terminals (of the same kind(!)) but what if one has 20 Suns, 10 SGI's, and 20 Dec workstations? Data for these stations is not necessarily comparable. Effective techniques for visualizing and understanding the performance of scalable heterogeneous networks are as yet undiscovered.

Xscal's principles, however, can be easily applied to domain-specific tools for viewing particular kinds of system data. For example, I can envision a process table lister (like top) based on Xscal, where several value-frequency views show the distribution of CPU consumption, memory consumption, etc. Xscal's coloring and texturing operations are too powerful; all one needs is an ability to move a scroll bubble around on the entity axis and examine entities in the order specified by the values. I can envision a special-purpose accounting viewer with the ability to reconstruct the distribution at a particular time, scroll through it using the value-frequency view as a scrollbar, and construct frequency histograms according to some automatically defined value range scheme. In retrospect, Xscal's tolerance for user-defined ranges may well be a weakness, because of the time it takes a user to set up such a color mapping scheme.

This makes it quite clear that Xscal should change from being a tool to being a library of visualization routines one can apply to different problem domains. I plan to work on this problem, and would welcome input on problem domains to which Xscal techniques could be applied, as well as opinions about the appropriate software platform to use in developing that library.

Availability

Xscal is freely available from ftp://ftp.cs.tufts.edu/pub/xscal . The current revision is 2.0Beta, which supports floating point. Revision 1.2 is limited to integer input. Source code for filters utilized in preparing this talk are available from ftp://ftp.cs.tufts.edu/pub/xscal/lisa-96 .

Author Information

Alva L. Couch was born in Winston-Salem, North Carolina where he attended the North Carolina School of the Arts as a high school major in bassoon and contrabassoon performance. He received an S.B. in Architecture from MIT in 1978, after which he worked for four years as a systems analyst and administrator at Harvard Medical School. Returning to school, he received an M.S. in Mathematics from Tufts in 1987, and a Ph.D. in Mathematics from Tufts in 1988. He became a member of the faculty of Tufts Department of Computer Science in the fall of 1988, and is currently an Associate Professor of Electrical Engineering and Computer Science at Tufts. In 1996 he received the Leibner Award for excellence in teaching and advising from Tufts. He has assisted in maintaining the Tufts computer systems for Computer Science teaching and research since 1985, when he was a Ph.D. student, and is currently responsible for maintaining the largest independent departmental computer network at Tufts. He can be reached by surface mail at the Department of Electrical Engineering and Computer Science, 161 College Avenue, Tufts University, Medford, MA 02155. He can be reached via electronic mail as `couch@cs.tufts.edu`. His work phone is (617)627-3674.

References

[1] Alva Couch, *Graphical Representations of Program Performance on Hypercube Message-Passing Multiprocessors*, Tufts University Department of Computer Science Technical Report 88-4, April 1988.

[2] Alva Couch, ‘‘Categories and Context in Scalable Execution Visualization’’, *Journal of Parallel and Distributed Computing* 18, August 1993, pp. 195-204.

[3] Alva Couch and David Krumme, ‘‘Portable execution traces for debugging and performance visualization’’. *Proc. Scalable High Perf. Computing Conference*, IEEE Computer Society Press, 1991.

[4] M. Heath and J. Etheridge, *Visualizing Performance of Parallel Programs*, ORNL Tech. Rep. ORNL-11813, Oak Ridge National Lab., Oak Ridge, TN, 1991.

[5] Michael F. Morris, ‘‘Kiviat Graphs – Conventions and ‘Figures of Merit’ ’’, *Performance Evaluation Review*, Vol 3, No 3, pp. 2-8, Oct. 1974.

[6] C. M. Pancake and C. Cook, ‘‘What Users Need in Parallel Tool Support: Survey Results and Analysis," *Proc. Scalable High Performance Computing Conf.*, 1994, pp. 40-47. http://www.nero.net/pancake/surveys/94-80-3.html .

[7] Dan Reed, R. Olson, R. Aydt, T. Madhyastha, T. Birkett, D. Jensen, B. Nazief, and B. Totty, ‘‘Scalable Performance Environments for Parallel Systems", *Proc. Sixth Distrib. Memory Computing Conf.*, IEEE Computer Society Press, 1992.

[8] Tcpview: the tcpdump network trace viewer, available in ftp://ftp.ac.washington.edu/pub/tcpview .