Toward Scalable Performance Visualization with Jumpshot*

Omer Zaki, Ewing Lusk, William Gropp, and *Deborah Swider*
Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, IL 60439
`{zaki,lusk,gropp,swider}@mcs.anl.gov`

Abstract

Jumpshot is a graphical tool for understanding the performance of parallel programs. It is in the tradition of the upshot tool, but contains a number of extensions and enhancements that make it suitable for large-scale parallel computations. Jumpshot takes as input a new, more flexible logfile format, and comes with a library for generating such logfiles. An MPI profiling library is also included, enabling the automatic generation of such logfiles from MPI programs. Jumpshot is written in Java, and can easily be integrated as an applet into browser-based computing environments. The most novel feature of Jumpshot is its automatic detection of anomalous durations, drawing the user's attention to problem areas in a parallel execution. This capability is particularly useful in large-scale parallel computations containing very many events.

1 Introduction

Jumpshot is a graphical tool for investigating the behavior of parallel programs. It is a "post-mortem" analyzer, taking as input a file of time-stamped events, which we call here a *logfile*. The file is written by the companion package CLOG, also described here. Jumpshot and CLOG are only loosely coupled by the format of the logfile and may be used independently.

Jumpshot can present multiple views of logfile data. The primary view is a series of timelines, one for each process, showing with colored bars the state of each process at each time. This view can be zoomed and scrolled for close examination of specific times. Other views include histograms of state durations and a "mountain range" view showing the aggregate number of processes in each state at each time.

Jumpshot is implemented in Java. Advantages of this approach include portability and network capability (the ability to run as a applet in a Web browser). The choice of Java was also an experiment in exploring the features and capabilities of current Java environments, and we report here on our experiences.

Logfile-based tools similar to Jumpshot have a rich history. Commercial tools include Time-Scan [2] and Vampir [3], and academic tools that are distributed for use by others include Para-

---

*This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Computational and Technology Research, U.S. Department of Energy, under Contract W-31-109-Eng-38.

Graph [12, 11], TraceView [17], XPVM [15], XMPI [4], and Pablo [19]. Each of these shares some features with Jumpshot, but it is unique in its particular combination of features.

Jumpshot is freely available and is packaged with the MPICH portable implementation of MPI. MPICH can be obtained from `http://www.mcs.anl.gov/mpi/mpich`. The distribution includes the CLOG logging facility and an MPI profiling library for use with MPI programs. Jumpshot is particularly easy to use with MPICH's compiling and linking scripts [6].

This paper is organized as follows. In Section 2 we give a synopsis of the history of Jumpshot and the requirements that drove its development. In Section 3 we present a walkthrough of Jumpshot features. Section 4 presents the logging package that accompanies Jumpshot and writes the files Jumpshot reads. We present Java-related issues in Section 5. Section 6 shows a few interesting specific logfiles and illustrates the kinds of insights that can be obtained with such a tool. We conclude with a discussion of opportunities for future research in this area.

2 Background

In this section we give a brief history of the developments that led to Jumpshot, culminating in a list of requirements.

2.1 History

Jumpshot traces its history back to the original version of gist, a program that was delivered with, and only ran on, the BBN Butterfly parallel computer. Gist is still available as part of the TotalView© debugging environment, although it is now known as TimeScan. The original version of upshot [13] was written to provide the most useful feature of gist, its zoomable and scrollable timeline window, in a color version (gist was black-and-white only) that was non-proprietary and would run on any workstation supporting X-windows. (Upshot gets its name from gist, for which it is a synonym [5].) Over the years upshot expanded to include a number of other features as well [14]. The enhancements were motivated by the development of an application whose highly irregular and input-dependent behavior made an analysis tool of this type indispensable [16]. The logfile generation library developed along with upshot was called ALOG, for Argonne logging.

Upshot originally was written in X (with the Athena widget set). Because it was cumbersome to expand and maintain, we completely rewrote it in the Tcl scripting language, using the Tk companion graphics package [18]. This approach made for extremely rapid development, but the interpreted nature of the language made the graphics component slow when logfiles were large. To deal with this problem, we rewrote the main graphics component of upshot in C, using Tcl's C interface. This rewrite solved the performance problem but introduced a dependency on an unstable part of Tcl itself, as the C interface changed right after the publication of [18]. This version, called nupshot (for new upshot), can still be used, however, if one obtains an earlier version of Tcl/Tk. Until recently, nupshot has been our main performance analysis tool. Upshot, in pure Tcl/Tk, is also still available, but also requires an old version of Tcl. This instability of Tcl/Tk encouraged us to find an alternative.

A few years ago, we rewrote the logging library to improve in multiple ways on ALOG. There was also an intermediate library called BLOG. For historical continuity, therefore, the new package had to be called CLOG (pronounced see-log). It is described in detail in Section 4.

2.2 Requirements for a New System

More than ten years of experience with similar tools gave us a clear set of requirements that our next-generation performance analysis tool should meet.

- It is expected to have a long lifetime and to be the foundation for our future research in performance analysis. It should therefore be implemented in a stable and widely supported programming environment.
- The tool must be portable to all the types of workstations that parallel programmers would want to use. Although the use of X made previous tools portable within the Unix universe, portability to Microsoft-based environments is now required.
- It must be able to deal with large logfiles in a scalable way. While filtering of *very* large logfiles can be expected to take place either within the application program itself or in an intermediate preprocessing step, in order to be a practical tool, performance needs to be maintained in the presence of many thousands of graphical objects being displayed, zoomed, and scrolled.
- The display package (the Jumpshot part) must not be too tightly integrated with the logging package (the CLOG part), for the sake of flexibility within each.
- Support for nested and overlapping states is needed, although nested states occur far more often than overlapping ones, because of the natural nesting of subroutine calls in programs.
- Support for MPI concepts, such as communicators, is required. At the same time, it is overly restrictive to tie the tool to the message-passing model of parallel computation.
- It must be relatively easy to manipulate the display, turning specific states on and off, changing colors, and so forth.
- The ability to create printable versions of displays is crucial, for inclusion in documents.
- A desirable feature is to help the user find the "interesting" parts of what might be a large and confusing display. This is a research topic we are only beginning to explore.
- Multiple types of display are useful for looking at the behavior of a program from many points of view:
  - process timelines with zooming and scrolling
  - histograms of state durations and message data
  - "mountain ranges" for aggregating state data
- It must be possible for the user to define states that are meaningful in the context of the application, not just the system.
- It must be possible to query specific states and events for details.
- The logfile format needs to be flexible, to account for unanticipated types of events and states, summary data, and entirely new concepts.
- It is desirable that an event can be connected back to the source code that generated it.

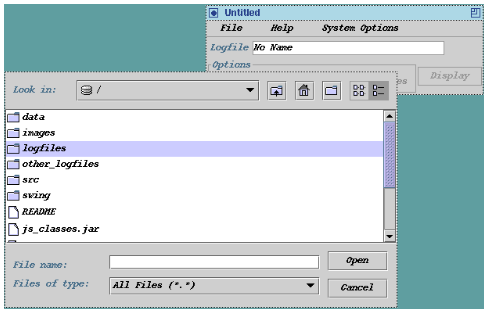
Jumpshot currently makes some compromises in attempting to meet these requirements. These compromises are expected to be temporary.

3 Using Jumpshot

In this section we give a walkthrough of using Jumpshot to view a very simple parallel program. We defer to Section 6 the viewing of more interesting examples. Jumpshot can be used either in stand-alone mode by running the `jumpshot` command or, when properly installed, from a Web browser, in which case it will run as an applet. In either case it begins with a simple opening window, as shown in Figure 1. The "look and feel" of Jumpshot is by default "metal", the default appearance of the Swingset class library. In the `System Options` menu it can be changed to have a Motif, Windows95, or MacOS appearance. Figure 2 shows the difference between Metal and Motif looks.

Logfile: No Name

Figure 1: Initial Jumpshot screen



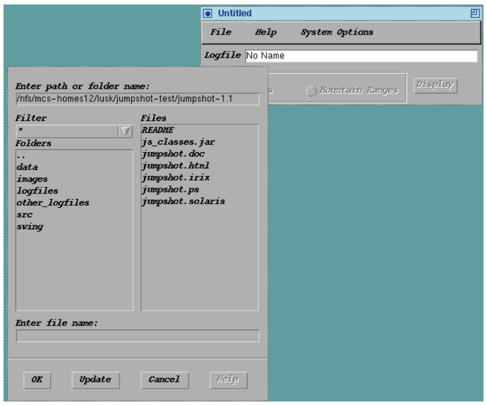
Logfile: No Name

Look in: /

data
images
logfiles
other_logfiles
src
swing
README
js_classes.jar

File name:

Files of type: All Files (*.*)



Logfile: No Name

Enter path or folder name: /nfs/mcs-homes12/lusk/jumpshot-test/jumpshot-1.1

Filter: *

Folders:
..
data
images
logfiles
other_logfiles
src
swing

Files:
README
js_classes.jar
jumpshot.doc
jumpshot.html
jumpshot.irix
jumpshot.ps
jumpshot.solaris

Enter file name: