ABSTRACT:
The NetLogger Methodology for High Performance Distributed Systems Performance Analysis

Brian Tierney, William Johnston, Jason Lee, Gary Hoo, Chris Brooks, Dan Gunter

Computing Sciences Division
Lawrence Berkeley National Laboratory[1]
University of California, Berkeley, CA, 94720

1.0 Introduction

Unexpected performance problems, such as low throughput or high latency, often arise in distributed systems, and are of particular concern in high-performance distributed systems. Finding the reasons for these problems is challenging because the nature of the systems tends to multiply the number of possible points of failure. While post-hoc diagnosis of performance problems is valuable for systemic problems, with operational problems users will have already suffered through a period of degraded performance. The ability to recognize, and possibly predict, operational problems will enable elements of the distributed system to use this information to adapt to operational conditions, thereby minimizing the impact on users.

We have developed a methodology known as *NetLogger*, for monitoring, under realistic operating conditions, the behavior of all the elements of the application-to-application communications path in order to determine exactly what is happening within a complex system. Applications and some operating system components are modified to perform precision timestamping and logging of "interesting" events, using a common log format, at every critical point in the distributed system. Software agents collect and filter this event-based performance information, turn on or off various monitoring options to adapt the monitoring to the current system state, and manage the large amounts of log data that are generated.

The goal of the performance characterization work is to produce predictable, high-speed components that can be used as building blocks for high-performance applications, rather than having to "tune" the applications top-to-bottom as is all too common today.

2.0 NetLogger

NetLogger consists of *event logs* that represent the raw information about system performance, and the *NetLogger Toolkit* that generates, manipulates, analyzes, and visualizes the logs.

1. The work described in this paper is supported by ARPA, Computer Systems Technology Office (http://ftp.arpa.mil/ResearchAreas.html) and the U. S. Dept. of Energy, Office of Energy Research, Office of Computational and Technology Research, Mathematical, Information, and Computational Sciences Division (http://www.er.doe.gov/production/octr/mics), under contract DE-AC03-76SF00098 with the University of California. Authors: bltierney@lbl.gov, Lawrence Berkeley National Laboratory, mail stop: B50B-2239, Berkeley, CA, 94720, http://www-itg.lbl.gov). This is report no. LBNL-NNNNN.

2.1 Events and Event Logs

Event logs contain high resolution, synchronized timestamps taken before and after *events* (activities of interest). Events may include application-, operating system-, or network-level activities; the latter two sets of events are monitored and logged using versions of the Unix utilities *vmstat* and *netstat* that have been modified to support NetLogger. The hosts comprising a NetLogger-monitored high performance distributed system must operate with all host system's clocks synchronized to about 0.25 - 0.5 ms. This can be achieved by using the Network Time Protocol (NTP)[2] [5] on networks where there are an appropriate number of precision time sources (e.g. GPS based NTP servers).

All events, including application-specific events, are logged in the NetLogger common logging format[3]: a unique *keyword* for each event, a time stamp, the host IP address, and event-specific information. Event log entries for a request to a server, for example, might include the request's identity so that its logical path through the server and the distributed system may be traced in isolation from other requests.

2.2 NetLogger Toolkit

The NetLogger Toolkit consists of three components: a C/C++ library of routines to simplify the generation of event logs, a set of tools for managing and filtering the log files, and a set of tools to visualize and to analyze the log files.

Event Log Generation Library

The C/C++ event log generation library contains routines to open, to write, and to close log files. Events can be written to a local file, to the Unix *syslog* daemon, or to a given port on a given network host. To add NetLogger event logs to an existing application it is best to have access to the application source code. However, it is possible do some monitoring without the source code by writing wrappers for various system components.

Agent-based Log File Management and Filtering

Log file management and filtering is performed by software agents. For this discussion, an *agent* is an autonomous, adaptable entity that is capable of collecting information about the state and history of a system component (e.g., a host). Agents provide the standard interface for monitoring CPU load, interrupt rate, TCP retransmissions, TCP windows size, etc. Some agents can independently perform various administrative tasks, such as restarting servers or monitor processes. Collectively, a distributed system's agents also maintain a continually updated view of the global state of the system.

NetLogger's agents are written in Java and use the KQML communication language [1]. They are designed to be able to "learn" new behavior dynamically, i.e., without the interruptions required for recompilation and restarting of, e.g., Unix daemons. Agents also communicate with one another to share new behavior and data.

We define a *broker* to be a special type of agent that manages system state information, filters this information for clients, or performs some action on behalf of a client. The broker controls agents on

2. See http://www.eecis.udel.edu/~ntp/.

3. NetLogger's common logging format was developed prior to the Internet Engineering Task Force's (IETF) current work on a standard logging format (see ftp://ds.internic.net/internet-drafts/draft-abela-ulm-01.txt) and NetLogger will likely be modified to support the IETF's standard when it has been officially issued.

each host, telling them which tasks to perform. The broker also collects event logs from each agent, and merges them together, sorted by timestamp, for use by the event log visualization tools.

The agent architecture is a crucial component for NetLogger because of the immense volume of log event data that can be generated: without the ability to filter this data, real-time analysis would be infeasible. Agents also provide a mechanism for real-time feedback permitting high performance distributed applications to adapt to current conditions.

We use Web applets to interface to the brokers. For example, one applet is used to trigger what events are logged, while another applet is used to collect and filter event logs for analysis or display.

Event Log Analysis Tools

NetLogger tools to analyze log files include *perl* scripts[4] to extract information from log files and write data files in a format suitable for *gnuplot*[5] to graph the results. These tools were used to generate the graphs below.

In order to provide real-time monitoring, the NetLogger Toolkit includes the Real Time Visualization (*rtv*) program (see: http://www-itg.lbl.gov/DPSS/userguide/real.html). *rtv* generates graphs similar to gnuplot, but can do so in real time, from a file or from the network. *rtv* provides the ability to play, pause, rewind, slow motion, zoom in/out, and so on.

3.0 Sample NetLogger Results

To illustrate the NetLogger approach, we describe its use in a high performance distributed system deployed in the DARPA-funded MAGIC gigabit testbed network[6] [2]. MAGIC has a complex, widely distributed application running over a high-speed ATM network. This is an example of a environment where a complete monitoring methodology is critical to obtaining expected performance.

3.1 The Distributed Parallel Storage System

A real-time terrain visualization application uses the MAGIC network to access multi-gigabit image data sets from the Distributed-Parallel Storage System (DPSS). The DPSS provides an economical, high-performance, widely distributed, and highly scalable architecture for caching large amounts of data that can potentially be used by many different users. It allows real-time recording of, and random access to, very large data sets. In the MAGIC testbed, DPSS system components are distributed across several sites separated by more than 2600 Km of high speed network, using IP over ATM as the network protocol.

The terrain visualization application (*TerraVision*) lets a user explore/navigate a “real” landscape represented in 3D by using ortho-corrected, one meter per pixel images and digital elevation models (see [4]). TerraVision requests from the DPSS, in real time, the sub-images (“tiles”) needed to provide a

4. For more information see: http://www.metronet.com/perlinfo/perl5.html

5. For more information see: http://www.cs.dartmouth.edu/gnuplot_info.html

6. MAGIC (Multidimensional Applications and Gigabit Internetwork Consortium) is a gigabit network testbed that was established in June 1992 by the U. S. Government’s Advanced Research Projects Agency (ARPA). The testbed is a collaboration between LBNL, Minnesota Supercomputer Center, SRI, Univ. of Kansas, Lawrence, KS, USGS - EROS Data Center, CNRI, Sprint, U. S. West, Southwest Bell, and Splitrock Telecom. More information about MAGIC may be found on the WWW at: http://www.magic.net/

view of a landscape for an autonomously "moving" user. Typical use requires aggregated data rates as high as 100 to 200 Mbits/sec. The DPSS is easily able to supply these data rates.

The DPSS is essentially a "logical block" server whose functional components are distributed across a wide-area network. It consists of multiple low-cost, medium-speed disk servers which use the network to aggregate multiple server outputs for high performance applications. To achieve high performance all levels of parallelism are exploited, including those available at the level of the disks, controllers, processors/memory banks, servers, and the network.

Other papers describing the DPSS, including one describing the implementation in detail [7], are available at http://www-itg.lbl.gov/DPSS/papers.html.

DPSS Timing Facility

The DPSS and several of its data-requesting clients, including TerraVision, have been instrumented to collect time stamps at all important events for each requested data block. When applications request data blocks from the DPSS, the requests take the following path (see Figure 1). A *request list* (list of

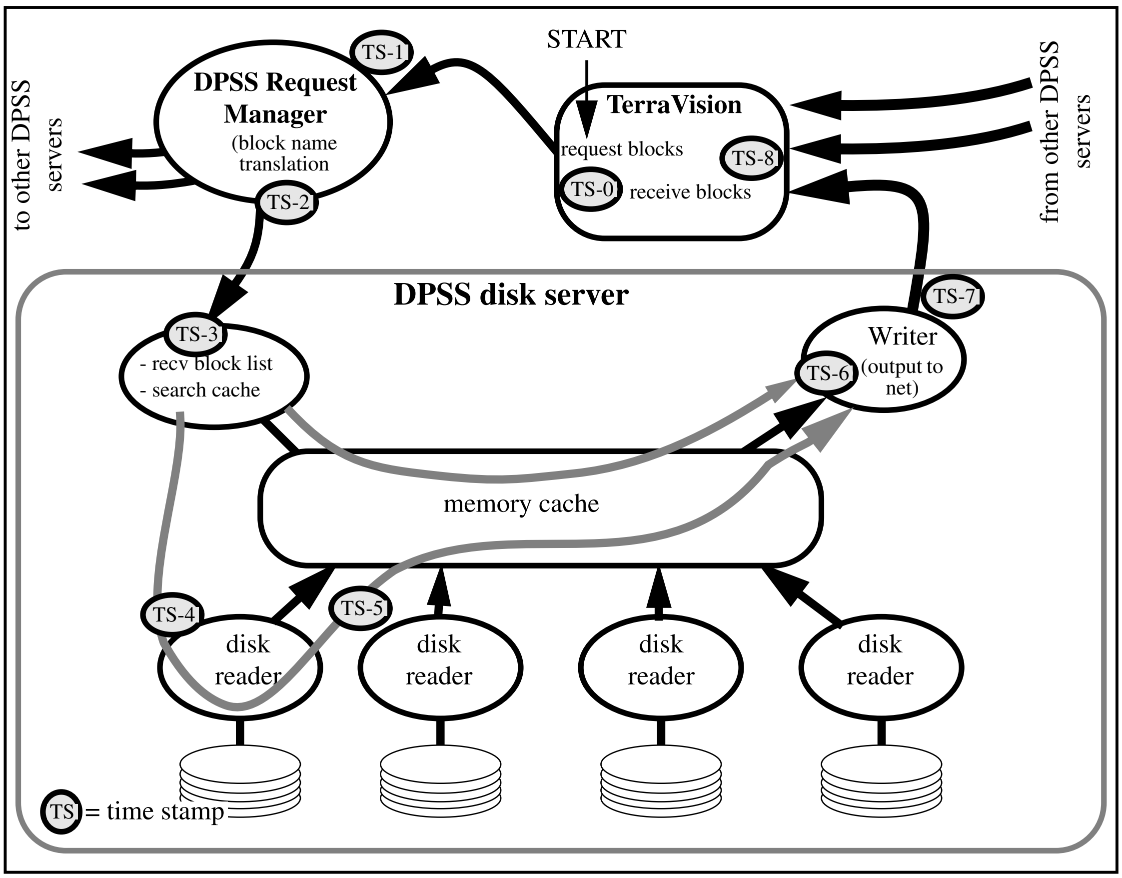

Figure 1 DPSS Performance Characterization Points and Optimal Average Timings

data blocks) is sent from the application to the *name server* ("START"), where the logical block names are translated to physical addresses (`server:disk:disk_offset`), then the individual requests are forwarded to the appropriate disk servers. At the disk servers, the data is read from disk into local cache, and then sent independently to the application (which has connections to all the relevant servers). Timestamps are gathered for each data block request before and after each major DPSS function, such as name translation, disk read, and network send. The DPSS-specific timestamps are sent with the data block back to the requesting application, where logging can be performed using the DPSS client library. Related events such as TCP retransmits, TCP window size, server CPU load and interrupt rates are logged independently and correlated to the request-response behavior in order to build up a complete picture.

NetLogger Analysis of the DPSS

To date, the single most useful analysis technique for NetLogger data has been to examine the data paths through the various system components, correlated in time with related events. These paths (lifelines) are characterized by the time of the actions, events, or operations in all of the system components. So, for example, DPSS events, together with TCP retransmit events, are plotted vertically and the corresponding times horizontally in Figure 2. Each "single" line corresponds to the life of a data request, from application request to application receipt—that is, to its lifeline. These lifelines have characteristic shapes and relationships that represent how the various components and algorithms in the system operate and interact in the face of different environmental conditions (e.g., network congestion).

Figure 2 illustrates the general operational characteristics of the DPSS[7]. Each line style in the graphs represents data from a different DPSS disk server. The graph plots "real time" (elapsed time since the beginning of the session) on the horizontal axis, and the chosen monitoring points on the vertical axis.

Referring to Figure 2, TerraVision sends a list of data block requests every 200 ms, as shown by the nearly vertical lines starting at the *app_send* monitor points. The initially single lifelines fan out at the *server_in* monitor point as the request lists are resolved into requests for individual data blocks. Each block request is first represented individually in the read queue (*start_read*).

Using these lifeline graphs it is possible to glean detailed information about individual operations within the disk servers. For example, when two lifelines cross in the area between *start_read* and *end_read*, this indicates that a read from one disk was faster that a read from another disk. (This phenomenon is clearly illustrated for the server represented by the crossing solid lines in Figure 2 at A.) Similarly, the parts of Figure 2 denoted by "B," "C," "D," "E," "F," and "G" denote performance measurements or unusual behavior that are immediately apparent from a visual inspection of the graphed data—measurements and behavior that would be far more difficult to identify from even close inspection of the voluminous logs of raw data.

Other Results

NetLogger has also been used to successfully analyze several other high performance distributed applications, such as a High Energy Nuclear Physics data analysis system designed to process one Terabyte of data per day[3]. NetLogger was also used to find an ATM switch buffer problem in the MAGIC network that triggered large numbers of TCP transmissions [6]. The final version of this paper will include results from these applications. The final version will also include more details and screendumps of the real-time event visualization tool *rtv,* and of the applets for interfacing to the brokers, described above.

7. Results in Figure 2 are from a four year old DPSS server platform (a 60 MHz Sun SS10). Current servers and disks are more than twice as fast; typical throughput is 10 MBytes/sec per DPSS disk server)

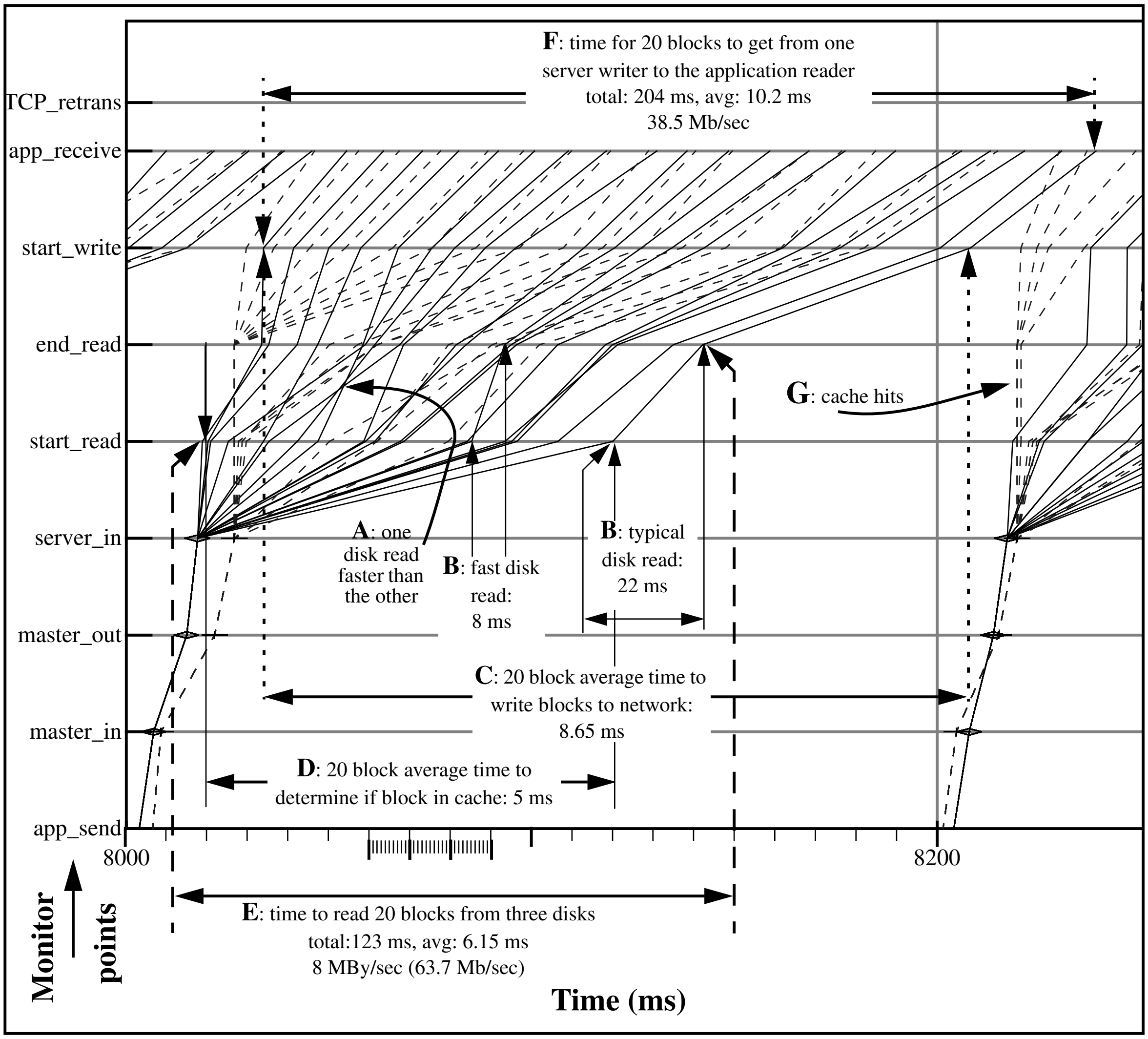

Figure 2 **Detail From a Two-Server, LAN, Experiment**

4.0 Related Work

There are several research projects addressing network performance analysis. For example, see the list of projects and tools on the Cooperative Association for Internet Data Analysis (CAIDA) Web site (http://www.caida.org/Tools/taxonomy.html). However, we believe that the NetLogger approach is unique in combining application, system, and network -level monitoring together.

5.0 Conclusions

We envision that monitoring top-to-bottom (i.e., including all relevant aspects of the application and its supporting infrastructure) and end-to-end (i.e., including all of the software and hardware components in the various paths used by widely distributed applications) will be a critical element in building reliable high-performance distributed systems. Moreover, agents that can adapt their behavior as needed can provide the level of control, response, and organization of the monitoring that is needed for applications to adapt to varying environments.

6.0 References

[1] Finin, T. et. al., DRAFT Specification of the KQML Agent Communication Language, unpublished draft, 1993. (http://www.cs.umbc.edu/kqml/kqmlspec/spec.html)

[2] Fuller, B., I. Richer "The MAGIC Project: From Vision to Reality," IEEE Network, May, 1996, Vol. 10, no. 3.

[3] Johnston, W.E., W. Greiman, G. Hoo, J. Lee, B. Tierney, C. Tull, D. Olson, "High-Speed Distributed Data Handling for On-Line Instrumentation Systems." Proceedings of IEEE/ACM Supercomputing 97. (http://www-itg.lbl.gov/DPSS/papers.html)

[4] Lau, S, and Y. Leclerc, "TerraVision: a Terrain Visualization System,", Technical Note 540, SRI International, Menlo Park, CA, Mar. 1994. (http://www.ai.sri.com/~magic/terravision.html)

[5] Mills, D., "Simple Network Time Protocol (SNTP)", RFC 1769, University of Delaware, March 1995. (http://www.eecis.udel.edu/~ntp/)

[6] Tierney, B., W. Johnston, G. Hoo, J. Lee, "Performance Analysis in High-Speed Wide-Area ATM Networks: Top-to-Bottom End-to-End Monitoring", IEEE Network, May, 1996, Vol. 10, no. 3. LBL Report 38246, 1996. (Also see http://www-itg.lbl.gov/DPSS/papers.html.)

[7] Tierney, B., Johnston, W., Chen, L.T., Herzog, H., Hoo, G., Jin, G., Lee, J., "Using High Speed Networks to Enable Distributed Parallel Image Server Systems", Proceedings of Supercomputing '94, Nov. 1994, LBL-35437. Available from http://www-itg.lbl.gov/DPSS/papers.html.