Abstract

In this paper, we explored where does the time of a system spend. Previous research in this area generally focus on CPU performance, with the increasing gap between CPU performance and I/O performance, I/O subsystem has played more and more important role in system performance. Our project focus on the I/O behavior of three typical applications, database, netscape and shell to see how much time a process will be stalled to wait for I/O. We use the existing tracing tool, and build our annualizing tools to get the results.

1. Introduction

At the last SOSP conference, there were two papers that purported to tell us "where does the time go" by periodically interrupting the CPU to see which instruction it was executing and thereby develop a long-term profile of CPU execution. Unfortunately, this methodology only measures where the CPU spends its time. It does not tell us when a process is stalling for I/O. Unfortunately, as many of us have experienced, it is common to spend as much time waiting for I/O as waiting for the CPU, and technology trends suggest that I/O delays will become more dominant in the future, since the gap between CPU performance and I/O performance are more noticeable. We use system call tracing to determine how much time processors spend waiting for I/O and what types of I/O are causing delays. The tracing program is based Janus project[6] , and we build analysis tool to get the result.

2. Background

The tracing tool we used in our project is originally developed for system security and utilize solaris built-in process tracing mechanism - proc[6]. It is modified to keep track with all the system calls and record the time spend on each system call. The I/O behavior of the system thus will be captured.

3. Applications

Different applications will have quite different behavior in how to use the system resources. In the large scope of applications, we focus on tracing database, netscape and shell. For each type, we traced user behavior for a certain period of time. The reason we choose these three types of applications are obvious. Currently, database are very important for commercial application, and for an ordinary user, browsing the web may constitute a large portion of the activities, also for a unix user, running shell to execute common commands are almost inevitable. The database is a minimal database called QUEL, with basic SQL ability and recovery ability. The netscape we used is netscape communicator 4.51. And the shell is tcshell in SunOS 5.5.

4. Methodology

During the tracing process, we break the system calls into two categories, I/O related and others. The I/O related system calls includes sys_read, sys_write, sys_open, sys_close, sys_creat, sys_lseek, sys_poll,sys_mmap, sys_munmap, sys_fcntl, sys_ioctl, sys_readv, sys_writev, sys_pread, sys_pwrite. Other system calls are like sys_fork, sys_time, sys_context, etc. In our project, we answer the question where does time go by answering which category a process would spend more time.

The available tracing tools have some limitations. First, the log file generated by the tracing program is very huge. Running a few minutes netscape will generate very large log files. If we want to monitor a user’s behavior in running netscape for a day, then the requirement of storage space would be a lot. There seem two ways to solve it. We can either analyze the data on the fly, or we can use a high-end disk for the experiment. Each of them has its advantages and disadvantages, analyzing data on the fly don’t require extra investment for the experiment, while we can only do the analyze once during the execution of the tracing program. If later on, we want to get more information from the results, we have to modify the tracing program and repeat the experiment again. We used a combination of these two strategies for our project. We didn’t save all information about the system calls during execution, but we do save for typical I/O system calls. And we keep the summary information for each tracing result. Second, since xterm is a setuid program, we cannot trace it to get an user’s pattern in using xterm. So, we trace shell to get the pattern. Because actually, xterm is actually executing shell to process the input commands.

5. Experiment

We did several experiment by running the modified tracing program to trace different applications.

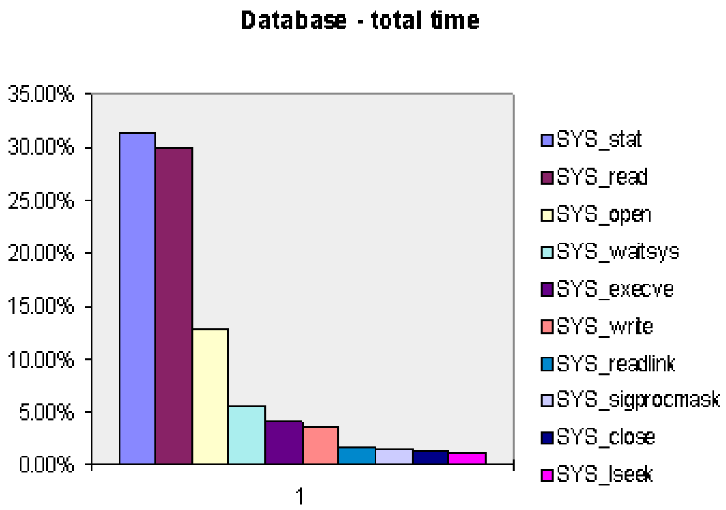

Figure 1

The first experiment is running the mini-database with a batch file input. The application finished in 2 minutes and the result shows that around 30% of system call times are spent on sys_read, 14% on sys_open and 4% on sys_write, 1% on sys_close and %1 on sys_lseek. Totally, around 50% of system call time are spent on doing I/Os.

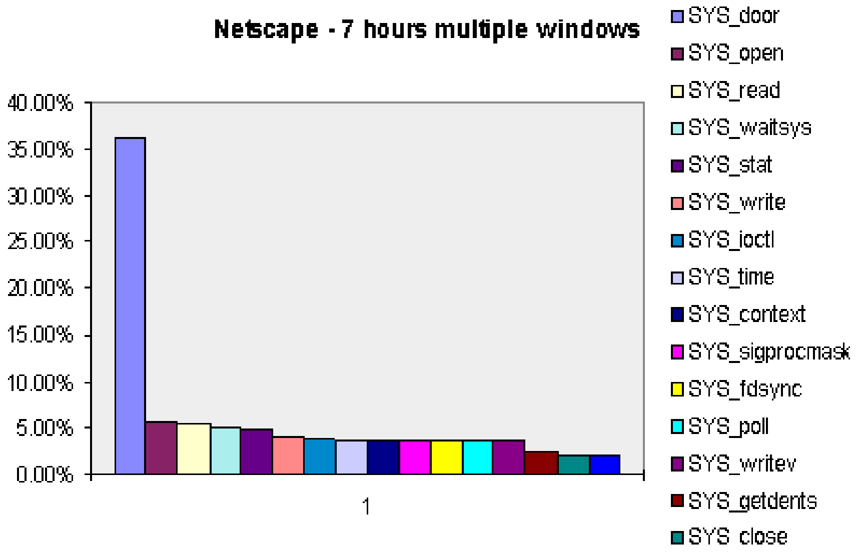

Figure 2

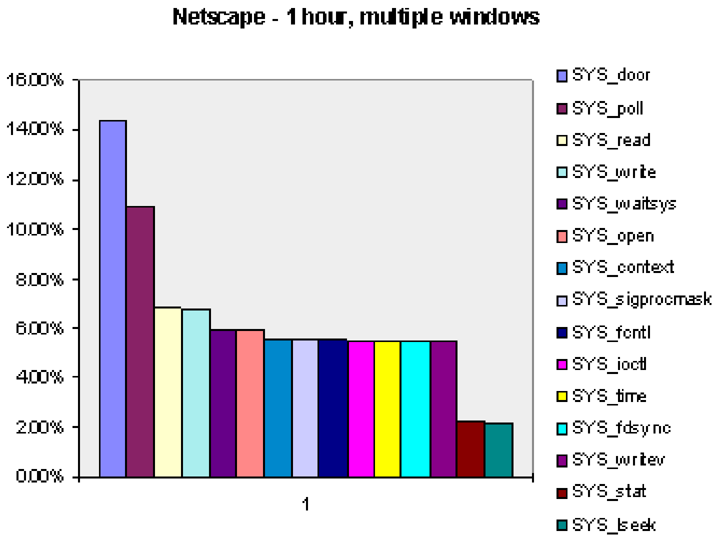

Figure 3

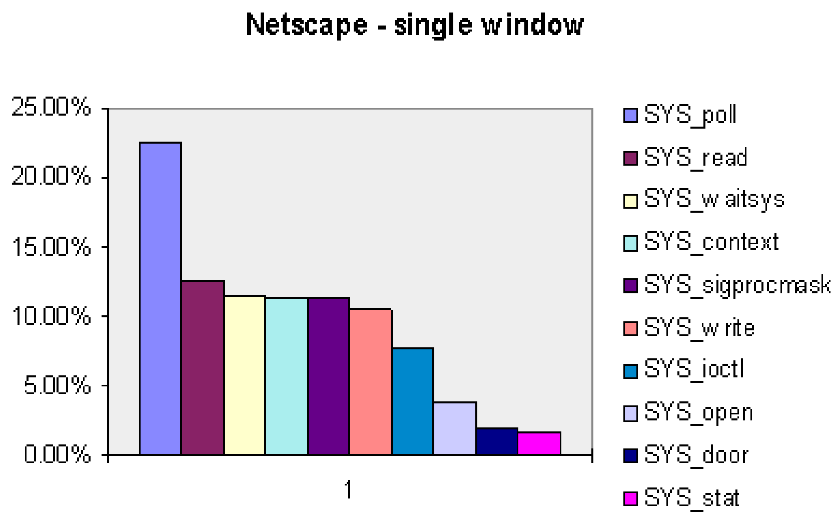

Figure 4

The second experiment is running the netscape for three different periods. We first run it for seven hours, hope the random factor in using browser is reduced by a long period. What we got is totally I/O system calls took 31.43% of total system times, while sys_door took 36.22%. Sys_door is a system call dealing with inter-process communications. Obviously, when we run netscape for a long time, we must generate a lot of related processes which will communicated with each other. Buy looking at the parameters of sys_open, sys_close, sys_read, sys_write, we know that when running netscape for a long time, both I/O and network won't count the biggest cost, while interprocess communications will become more dominant. Our second test on netscape is running it for an hour, while with multiple windows, in this way, 50% of time is spent in I/Os, and sys_door took 14.39%. The third experiment is running netscape in one hour with single window, this way, 60.60% of system time is doing I/O, while sys_door took only 2%.

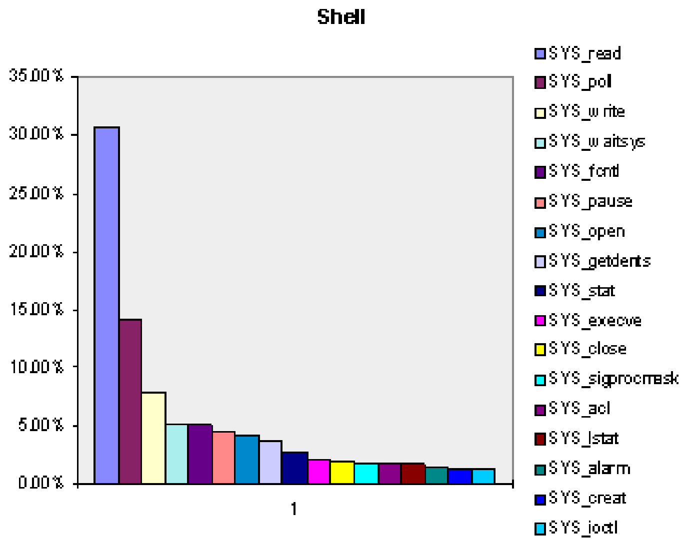

Figure 5

The third experiment is to trace the shell of a user for a day. In this experiment sys_read took 31% and totally I/O took 65.27% of time.

From the experiment results, we notice that, I/O time for these three types of applications take more than half of the system time. To improve the performance, we should improve the I/O performance of a system.

6. Related Work

Computer architecture designs have been studied extensively using trace-driven performance analysis. Traces of application execution are captured and fed into software models of a system. Various design tradeoffs can be evaluated using this methodology. The quality of the traces will directly affect the success or failure of any design tradeoff being investigated, so these traces should be carefully inspected.

There are several methods to capture operating-system-rich traces, in which traces contain operating system execution. Hardware based tracing [1, 2] involves the construction of a monitor which can capture trace records at machine speeds, while not perturbing the system under study. These tracing systems typically exploit some form of multiplexing across a set of memory arrays in order to match the speed of trace production with the speed of trace capture (which is bound by the speed of the trace storage device). The cost of constructing such a system is typically large, and many times can prohibit using this approach.

Another popular method for capturing traces which modifies either the program source or the object code is called software-based tracing[3, 4]. Here, hooks are placed into the executable and trace information is generated as the program is run. Code can be modified at assembly time, link time or execution time. The strength of software-based tracing is its modifiability. If the desired trace information changes, the trace mechanism is easily updated.

Another tracing technique is to use emulation. Here, a layer is constructed between the host machine and the operating system under evaluation. A good example of such a system is the SimOS project at Stanford [5]. The layer only emulates enough of the system to allow the operating system to run correctly. While this system provides a flexible interface for collecting operating-system rich traces, the accuracy of the captured trace is suspect. Since emulation is performed, execution will be perturbed. This can significantly reduce the accuracy of the trace-driven simulation.

7. Conclusion

From the experiment we did, we know that for database, netscape and shell, more half of the system time a process will be stalled to wait for I/O. With the increasing gap between CPU performance and I/O performance, to increase the performance of the whole system, we need to improve the architecture of the I/O subsystem.

Reference

1. K. Flanagan, J. Archibald and K. Grimsrud, "BACH: BYU Address Collection Hardware, the Collection of Complete Traces", 6th International Conference on Modeling Techniques and Tools for Computer Evaluation, 1992, pp. 128-137

2. D. Kaeli, O. LaMaire, W. White, P. Hennet and W. Starke, "Real-Time Trace Generation", Internatinoal Journal on Computer Simulation, Vol. 6, No. 1, 1996, pp. 53-68

3. B. Chen and B. Bershad, "The impact of Operating System Structure on Memory System Performance", Operating Systems Review, Vol. 27, No. 5, Dec. 1993, p. 120-133

4. J. Larus, "Abstract Execution: A Technique for Efficiently Tracing Programs" Univ. of Wisconsin-Madison Technical Report, 1990.

5. M. Rosenblum, E. Bugnion, S. Devine, and S. Herrod, "Using the SimOS Machine Simulator to Study Complex Computer Systems", ACM Trans. On Modeling and Simulation, Vol. 7, No. 1, January 1997, pp. 78-103

6. D. Wagner and I. Goldberg and R. Thomas "A Secure Environment for Untrusted Helper Applications: Confining the Wily Hacker" 1996 USENIX Security Symposium. To Appear.