Performance measurement

Overview

In building an operating system, it is important to be able to determine the performance characteristics of underlying hardware components (CPU, RAM, disk, network, etc.), and to understand how their performance influences or constrains operating system services. Likewise, in building an application, one should understand the performance of the underlying hardware and operating system, and how they relate to the user's subjective sense of that application's "responsiveness". While some of the relevant quantities can be found in specs and documentation, many must be determined experimentally. While some values may be used to predict others, the relations between lower-level and higher-level performance are often subtle and non-obvious.

In this project, you will create, justify, and apply a set of experiments to a system to characterize and understand its performance. In addition, you may explore the relations between some of these quantities. In doing so, you will study how to use benchmarks to usefully characterize a complex system. You should also gain an intuitive feel for the relative speeds of different basic operations, which is invaluable in identifying performance bottlenecks.

You have complete choice over the operation system and hardware platform for your measurements. You can use your laptop that you are comfortable with, an operating system running in a virtual machine monitor, a smartphone, a game system, or even a supercomputer.

You may work either alone or in a group of 2. Groups do the same project as individuals. All members receive the same grade. Note that working in groups may or may not make the project easier, depending on how the group interactions work out. If collaboration issues arise, contact your instructor as soon as possible: flexibility in dealing with such issues decreases as the deadline approaches.

This project has two parts. First, you will implement and perform a series of experiments. Second, you will write a report documenting the methodology and results of your experiments. When you finish, you will submit your report as well as the code used to perform your experiments.

Report

Your report will have a number of sections including an introduction, a machine description, and descriptions and discussions of your experiments.

1) Introduction

Describe the goals of the project and, if you are in a group, who performed which experiments. State the language you used to implement your measurements, and the compiler version and optimization settings you used to compile your code. If you are measuring in an unusual environment (e.g., virtual machine, Web browser, compute cloud, etc.), discuss the implications of the environment on the measurement task (e.g., additional variance that is difficult for you to control for). Estimate the amount of time you spent on this project.

2) Machine Description

Your report should contain a reasonably detailed description of the test machine(s). The relevant information should be available either from the system (e.g., `sysctl` on BSD, `/proc` on Linux, System Profiler on Mac OS X, the `cpuid` x86 instruction), or online. Gathering this information should not require much work, but in explaining and analyzing your results you will find these numbers useful. You should report at least the following quantities:

1. Processor: model, cycle time, cache sizes (L1, L2, instruction, data, etc.)
2. Memory bus
3. I/O bus
4. RAM size
5. Disk: capacity, RPM, controller cache size
6. Operating system (including version/release)

3) Experiments

Perform your experiments by following these steps:

1. Estimate the base hardware performance of the operation and cite the source you used to determine this quantity (system info, a particular document). For example, when measuring disk read performance for a particular size, you can refer to the disk specification (easily found online) to determine seek, rotation, and transfer performance. Based on these values, you can estimate the average time to read a given amount of data from the disk assuming no software overheads. For operations where the hardware performance does not apply or is difficult to measure (e.g., procedure call), state it as such.
2. Make a guess as to how much overhead software will add to the base hardware performance. For a disk read, this overhead will include the system call, arranging the read I/O operation, handling the completed read, and copying the data read into

the user buffer. We will not grade you on your guess, this is for you to test your intuition. (Obviously you can do this after performing the experiment to derive an accurate "guess", but where is the fun in that?) For a procedure call, this overhead will consist of the instructions used to manage arguments and make the jump. Finally, if you are measuring a system in an unusual environment (e.g., virtual machine, compute cloud, Web browser, etc.), estimate the degree of variability and error that might be introduced when performing your measurements.

3. Combine the base hardware performance and your estimate of software overhead into an overall prediction of performance.
4. Implement and perform the measurement. In all cases, you should run your experiment multiple times, for long enough to obtain repeatable measurements, and average the results. Also compute the standard deviation across the measurements. Note that, when measuring an operation using many iterations (e.g., system call overhead), consider each run of iterations as a single trial and compute the standard deviation across multiple trials (not each individual iteration).
5. Use a low-overhead mechanism for reading timestamps. All modern processors have a cycle counter that applications can read using a special instruction (e.g., rdtsc). Searching for "rdtsc" in Google, for instance, will provide you with a plethora of additional examples. Note, though, that in the modern age of power-efficient multicore processors, you will need to take additional steps to reliably use the cycle counter to measure the passage of time. You will want to disable dynamically adjusted CPU frequency (the mechanism will depend on your platform) so that the frequency at which the processor computes is determinstic and does not vary. Use "nice" to boost your process priority. Restrict your measurement programs to using a single core.

In your report:

1. Clearly explain the methodology of your experiment.
2. Present your results:
    1. For measurements of single quantities (e.g., system call overhead), use a table to summarize your results. In the table report the base hardware performance, your estimate of software overhead, your prediction of operation time, and your measured operation time.
    2. For measurements of operations as a function of some other quantity, report your results as a graph with operation time on the y-axis and the varied quantity on the x-axis. Include your estimates of base hardware

        performance and overall prediction of operation time as curves on the graph as well.

3. Discuss your results:
    1. Cite the source for the base hardware performance.
    2. Compare the measured performance with the predicted performance. If they are wildly different, speculate on reasons why. What may be contributing to the overhead?
    3. Evaluate the success of your methodology. How accurate do you think your results are?
    4. For graphs, explain any interesting features of the curves.
    5. Answer any questions specifically mentioned with the operation.
4. At the end of your report, summarize your results in a table for a complete overview. The columns in your table should include "Operation", "Base Hardware Performance", "Estimated Software Overhead", "Predicted Time", and "Measured Time". (Not required for the draft.)
5. State the units of all reported values.

Do not underestimate the time it takes to describe your methodology and results.

4) Operations

1. CPU, Scheduling, and OS Services
    1. Measurement overhead: Report the overhead of reading time, and report the overhead of using a loop to measure many iterations of an operation.
    2. Procedure call overhead: Report as a function of number of integer arguments from 0-7. What is the increment overhead of an argument?
    3. System call overhead: Report the cost of a minimal system call. How does it compare to the cost of a procedure call? Note that some operating systems will cache the results of some system calls (e.g., idempotent system calls like getpid), so only the first call by a process will actually trap into the OS.
    4. Task creation time: Report the time to create and run both a process and a kernel thread (kernel threads run at user-level, but they are created and managed by the OS; e.g., pthread_create on modern Linux will create a kernel-managed thread). How do they compare?
    5. Context switch time: Report the time to context switch from one process to another, and from one kernel thread to another. How do they compare? In the past students have found using blocking pipes to be useful for forcing context switches.
2. Memory

1. RAM access time: Report latency for individual integer accesses to main memory and the L1 and L2 caches. Present results as a graph with the x-axis as the log of the size of the memory region accessed, and the y-axis as the average latency. Note that the lmbench paper is a good reference for this experiment. In terms of the lmbench paper, measure the "back-to-back-load" latency and report your results in a graph similar to Fig. 1 in the paper. You should not need to use information about the machine or the size of the L1, L2, etc., caches when implementing the experiment; the experiment will reveal these sizes. In your graph, label the places that indicate the different hardware regimes (L1 to L2 transition, etc.).
2. RAM bandwidth: Report bandwidth for both reading and writing. Use loop unrolling to get more accurate results, and keep in mind the effects of cache line prefetching (e.g., see the lmbench paper).
3. Page fault service time: Report the time for faulting an entire page from disk (mmap is one useful mechanism). Dividing by the size of a page, how does it compare to the latency of accessing a byte from main memory?

References

The following papers may be useful for help with system measurement:

- John K. Ousterhout, Why Aren't Operating Systems Getting Faster as Fast as Hardware?, Proc. of USENIX Summer Conference, pp. 247-256, June 1990.
- J. Bradley Chen, Yasuhiro Endo, Kee Chan, David Mazieres, Antonio Dias, Margo Seltzer, and Michael D. Smith, The measured performance of personal computer operating systems, Proc. of ACM SOSP, pp. 299-313, December 1995.
- Larry McVoy and Carl Staelin, lmbench: Portable Tools for Performance Analysis, Proc. of USENIX Annual Technical Conference, January 1996.
- Aaron B. Brown and Margo I. Seltzer, Operating system benchmarking in the wake of lmbench: a case study of the performance of NetBSD on the Intel x86 architecture, Proc. of ACM SIGMETRICS, pp. 214-224, June 1997.

You may read these papers, or other references, for strategies on performing measurements, but you may not examine code to copy or replicate the implementation of a measurement. For example, reading the lmbench paper is fine, but downloading and looking at the lmbench code violates the intent of the project.

Finally, it goes almost without saying that you must implement all of your measurements. You may not download a tool to perform the measurements for you.