Execution Time

Descriptive Data Analysis and Statistical Inference

1 Introduction

Computer performance analysis consists of discovering and ascertaining the efficiency of a computer system; it may be, for example, concerned with the estimation of the performance behavior of systems under construction, or monitoring that of an existing one. The findings of a performance analysis may be used to guide decisions relating to system design, the allocation of machine resources, the acquisiton of additional facilities, or the tuning of an existing configuration. Carrying out proper performance analysis is recognized to be an integral part of the professional construction and management of computer systems. The basic terminology and main concepts involved in a computer system's performance analysis are summarized in

`http://www.frontrunnercpc.com/info/infomain-fs.htm`

There exist many software packages that were developed explicitly to measure a computer system's performance. All of these software packages are based on the same concepts, and therefore for a beginner, it is better to invest the time in learning these concepts before embarking into learning about those packages.

A fundamental performance metric of any computer system is the time required to execute a given application program. The system that produces the smallest total execution time for a given application program has the highest performance. It is therefore important to know how to measure the execution time of a program, or a portion of a program.

A computer system has an internal counter that simply counts the number of clock ticks that have occurred since the system was first turned on. A time interval then is measured by reading the value of the counter at the start of the event to be timed and again at the end of the event. The elapsed time is the difference between the two count values multiplied by the period of the clock ticks.

As an example, consider the program shown below. In this example, the

```
init_timer()
```

function initializes the data structures used to access the system's timer. This timer is a simple counter that is incremented continuously by a clock with a period defined in the variable

```
clock_cycle
```

. Reading the address pointed to by the variable

```
read_count
```

returns the current count value of the timer.

To begin timing a portion of a program, the current value in the timer is read and stored in

```
start_count.
```

At the end of the portion of the program being timed, the timer value is again read and stored in

```
end_count.
```

The difference between these two values is the total number of clock ticks that occurred during the execution of the event being measured. The total time required to execute this event is this number of clock ticks multiplied by the period of each tick, which is stored in the constant

```
clock_cycle.
```

This technique for measuring the elapsed execution time of any selected portion of a program is often referred to as the *wall clock* time since it measures the total time that a user would have to wait to obtain the results produced by the program. That is, the

measurement includes the time spent waiting for input/output operations to complete, memory paging, and other system operations performed on behalf of this application, all of which are integral components of the program execution. However, when the system being measured is time-shared so that it is not dedicated to the execution of this one application program, this elapsed execution time also includes the time the application spends waiting while other users' applications execute.

Many researchers have arguesd that including this time-sharing overhead in the program's total execution time is unfair. Instead, they advocate measuring performance using the total time the processor actually spends executing the program, called the *CPU Time*. This time does not include the time the program is context switched-out while another application runs. But using only this CPU time as the performance metric ignores the waiting time that is inherent to the application as well as the time spent waiting on other programs. A better solution is to report both the CPU time and the total execution time so the reader can determine the significance of the time-sharing interference.

2 Activity

In addition to the system overhead effects, the measured execution time of an application program can vary significantly from one run to another since the program must contend with random events, such as the execution of background operating system tasks, different virtual-to-physical page mappings and cache mappings from explicitly random replacement policies, variable system load in a time-shared system and so forth. As a result, a program's execution time is a random variable. It is important, then, to measure a program's total elapsed execution time several times and report at least the mean and variance of the times.

In this activity, you are going to do the following:

- Generate the data that you are going to analyze to compare the execution time of two different computers in running a program.
  - Choose a computer where you can run a C program like the one below. Describe the characteristics of this computer.
  - Run the following program 1000 times and store the 1000

    `elapsed_time`

    values in another variable called

`exectime1`

. Alternatively, you can run a program of your choice, in the language you want, but make sure to comment it and summarize what it is doing and to include the counter for the execution time.

Notice that you will have to write other routines in the program to instruct the computer to repeat the running of the program 1000 times, and to store the execution times in a variable. So the program below is incomplete.

```
main()
{
 int i;
  float a;

  init_timer();

/* Read the starting time */
start_count = read_count;

/*Stuff to be measured */
for (i=0; i < 10000000; i++){
   a = i* a /10;
   }

/* Read the ending time. */

end_count=read_count;

elapsed_time = (end_count - start_count)*clock_cycle;

}
```

– Go to another computer where you can also run a C program like the one above (or alternatively the program you chose), and run the same program another 1000 times, storing the new values of

`elapsed_time`

in another variabe called exectime2

– After you are done running the programs, you should have a text file with two columns, one column with the exectime1 and exectime2 one after another and another column with a dummy variable, indicating whether the measurement is

from computer1 or computer2. Alternatively, you can have two columns, one for each exectime, and then do some data management in the statistical software. That file will be read by a Statistical Software Package to do the statistical analysis that follows

Now, your data is created and you are ready to start using a statistical software package to do the statistical analysis.

- Compute all summary statistics for the new data (mean, media, percentiles, standard deviation, maximum, minimum), and plot the histograms and box plots for the new data. Describe what the summary statistics and the graphs are saying.
- Do a two-sample test of hypothesis to determine whether there is significant difference in the performance of the two computer systems.
  - Write the null and the alternative hypothesis
  - Write the test statistic and its value
  - Write the P-value
  - Write your decision (use $\alpha = 0.05$)
  - Write the answer to the question: is there statistically significant difference in the mean execution time of the two computer systems?
- Estimate the difference between the average execution time in the two computer systems with 95% confidence. Interpret what the interval is saying.
- Comment on the assumptions needed for the test and CI results to mean something about the population of running times of these computers at large (i.e., at all times). Are your sample random? Are the measurements iid?