Script

PROJECT MAC April 2, 1975

Computer Systems Research Division Request for Comments No. 71

Accurate Performance Measurement on the Development Machine: Initial Experiments

by Andrew H. Mason

Users of a computer utility are sensitive to prices and price changes. Because prices are generally a function of the system costs, the administrator in charge of the utility should have the capability to determine exactly how a change in his system will affect the system costs. He can then decide, on the basis of this information, whether to install the change.

The ultimate goal of the research described here is to develop a "useful" performance measurement tool for the Multics operating system. This tool, which will be in the form of a benchmark, can be used to predict system performance. Three criteria are being used to determine the "usefulness" of such a test: it should be inexpensive to use; the test should be precise and repeatable; and the results of the test should reflect the performance of the system under different operating conditions (i.e., the test should be accurate).

Currently, the major cost factor is machine time, so if the test ran in, say, ten minutes it would be cheap enough that it

---

could be repeated several times if necessary. If the test is not repeatable, the validity of any quantitative results is suspect. As for accuracy, if the results of tests on systems A and B indicate that system A is two percent faster than system B, one would like to have confidence that system A will actually run two percent faster than system B. This third criterion is hard to measure, so research to date has concentrated on the first two. The strategy for developing the performance test has been to use an existing benchmark as a basis, modifying it in ways to increase its speed and accuracy.

Currently, Honeywell uses a modified version of the performance benchmark developed by Roger Roach (see MTB-126 and MAB-016) to evaluate system changes. This version of the test will be called the standard test. It runs on the CISL development machine and is available to all development machine users. The primary value measured by the benchmark is the real time required for a number of absentee processes to run to completion. To run this benchmark, a control process or driver is logged in. The driver then logs in several slave processes, makes each one wait until all have logged in, and then starts them all together. The run time is measured from the time when all slaves are started by the driver to when the last slave signals that it has finished. The coordination and inter-process communication between the driver and one slave is depicted in Figure I.

```
driver:                                  slave:
     call initialize_meters;
     login(slave);
     block(synch);
                                              wakeup(driver,synch);
                                              block(start);
     call start_meters;
     wakeup(slave,start);
     block(finish);
                                              .
                                              .
                                              .
                                              wakeup(driver,finish);
                                              logout;
                                              end;
     call stop_meters;
     call print_meters;
     end;
```

Figure I. Driver - Slave Coordination Before Modification.

Each slave process follows one of five standard scripts. The commands in the scripts are mostly editing and compilation commands. One of the scripts, however, follows each command with a "flush" command. The "flush" command pages heavily, insuring that the paging device is in constant use. In a typical run on the development machine, twenty slaves are measured, four following each script.

For the purposes of system measurement, the standard test presents two problems: it is expensive to use and it is imprecise. The standard test runs in a little under three hours, but the total run time has been known to vary more than ten percent in tests made on identical systems and identical configurations! This implies that if a ten percent change in total run time is measured between two systems, it is impossible to tell how much, if any, of the difference is due to the system change. The only way to produce meaningful results is to repeat the test many times on each system, at great expense. When enough tests have been completed, statistical analysis of the run times will yield a mean and a standard deviation. Comparison of the mean run times for each system will indicate which is faster and by how much, and the standard deviation will tell how precise the test is.

As of this writing, four modifications to the standard test have been finished. These modifications are not adequate to meet the goals of the project. Therefore, two other modifications

will be proposed in this paper which should bring the performance test closer to the goals. Figure II lists the results of tests made at each stage of modification. Some of the measured speed up is due to the installation of cache memory on the development machine. This occurred between modifications 2 and 3.

The first completed modification involved reducing the number of slave processes from twenty to five. One slave was started following each script. The average run time for the test dropped to about twenty minutes. This was a vast improvement over a three hour run time, but the discrepancy between identical tests was still high. For one system, one test ran in 20 minutes and 2 seconds, and another ran in 22 minutes and 21 seconds. The variation in run time was so large (about 10 percent) that no attempt was made to obtain a statistically significant sample.

Instead, an effort was made to isolate the causes of the large difference. As a result of this effort, a bug was found in the logout sequence for absentee processes which indicated one possible cause: As each slave logged out, it might encounter an error condition which caused it to create stack frames until it overran its stack. At this point, the process was terminated. This extra activity demanded a large amount of system resources, causing interference with other slaves.

Since this bug did not occur with a known frequency, an unpredictable amount of interference was introduced into the test. However, the logout sequence is not a logical part of the

SYSTEM	# OF RUNS	HIGH	LOW	MEAN
before modification	1			8930
after modification 1	3	1341	1202	1254
after modification 2	2	1132	894	1013
after modification 3	4	915	813	842
after modification 4	3	753	702	730

Figure II. Increases in System Performance Test Speed.

(times are measured in seconds)

test (see Figure I) and should be completely removed! At best, if logouts are included, the results of the test should be too slow by some constant factor. At worst, however, the error introduced is some complicated function of many variables. Therefore, a change to the test was made by Steve Webber which eliminated the logout sequence. Instead of logging out, each slave process signals the driver that it has finished and waits to be told to logout. When all slaves have finished, the control process stops the test and then signals all slaves to logout. This has the effect of eliminating the interference caused by a slave's logout while other slaves are still running. The driver - slave coordination after this modification is shown in Figure III.

On a standard system, there are three (ordered) scheduling queues used by the traffic controller. The queue in which a process is waiting to run defines the process' priority and its time quantum. Each time a process interacts with a terminal the process is promoted to the highest priority queue. Absentee processes, however, do not interact with terminals, so all slave processes drift down to the lowest queue and stay there. Since Multics is primarily a time-sharing facility, service installations normally have many interactive processes and few absentee processes. To better reflect this situation, the slave processes should simulate terminal interaction. The third modification will do this! In the system data base, there is a

```
driver:                                  slave:

     call initialize_meters;

     login(slave);

     block(synch);

                                              wakeup(driver,synch);

                                              block(start);

     call start_meters;

     wakeup(slave,start);

     block(finish);

                                              .
                                              .
                                              .

                                              wakeup(driver,finish);

                                              block(term);

     call stop_meters;

     wakeup(slave,term);

     call print_meters;

     end;                                     logout;

                                              end;
```

Figure III. Driver - Slave Coordination After Modification 2.

parameter called "priority_sched_inc". It is usually set at 80 seconds. It means that if a process has been blocked for more than 80 seconds, it should be placed in the highest priority queue when it next tries to run. Setting this value to 1 second has the effect of making slaves act more like interactive processes.

The last modification also deals with the scheduling algorithm. In an attempt to reduce thrashing, the scheduler tries to estimate a process' working set at the time the process goes blocked. When the process next wants to run, the working set estimate is compared to the number of free pages in core. If the working set is larger, the process is not allowed to run. Instead, the processor idles. Unfortunately, when only five slaves are run during a test, the processor is idle for this reason about 16 percent of the time. In an effort to reduce this amount of idle, parameters of the working set estimator were adjusted so that all processes appeared to have working sets of fifty pages. This cut down on idle time and reduced the average run time of the test and the number of disk reads.

This last effect indicates that some of the test variation may be due to variations in disk reference patterns. This hypothesis is further reinforced by the following evidence: First, the test run time correlates to a high degree with the number of disk reads made during the test. Second, the test run time also correlates well with the mean page-wait time for the

disk. Third, in a typical run of the modified version, the test takes about fifteen minutes, reads from the disk about 25,000 times, and has a mean page-wait time for the disk of about 60 milliseconds. This implies that each slave spends about five minutes of real time waiting for the disk! Unfortunately, little is known about the statistical distribution of page-wait times, so there is no way to judge its stability. Therefore, the first proposed modification is to install meters on the disk which will precisely describe this distribution. One possible method would be to divide page-wait time into bins one or two milliseconds wide. In each bin, keep track of the number of corresponding disk reads and the total amount of real time between page-faults and restarts. This information could be printed out in a histogram, giving a very good graphical picture of the page-wait time distribution. (1) Knowing the distribution of page-wait times will not explain the noted correlations, but it might indicate the nature of the variations in disk reference patterns.

measure page-wait time distribution

Another disturbing element is that from test to test, the order of slave logouts changes. In addition, as more slaves logout, processor idle increases. Intuition would indicate that if this is the case, more resources are available to each slave, and the "computation rate" of each slave increases. This is not

---

(1) In 1973, Lee Scheffler from Honeywell installed meters similar to the ones described here. At the present time, it is unknown whether they work.

necessarily the case. The commands executed by the slaves are taken from the system libraries, so it is quite likely that the slaves share pages. Therefore, the fact that one slave is using segment "pl1" may reduce the number of page-faults taken by another slave which is also using segment "pl1". This implies that if one slave logs out, the effective computation rate for the other slaves may, in fact, be reduced. To correct for effects of this nature, the driver - slave coordination should be changed as in Figure IV. Here, the important point is that after each slave finishes its assigned task, it should keep on running in order to keep the load on the processor and the inter-slave interference as constant as possible. One important effect of this modification is that the meaning of the value measured by the test is changed. Now, the test measures the rate a which work is being completed.

These modifications, although necessary, are only a first step in this project. When they are completed, there will hopefully be a better understanding of the sources of variation in the test. Knowing this, the project can begin to narrow in on how much of the variation can be feasibly eliminated.

Although the task of improving metering tools is important, a larger issue should be mentioned in this paper. Suppose that a "perfect" metering test has been invented for the development machine. Do the results obtained indicate anything at all about performance on another Multics? The development machine has a

```
driver:                                slave:
     call initialize_meters;
     login(slave);
     block(synch);
                                            wakeup(driver,synch);
                                            block(test);
     call start_meters;
     wakeup(slave,start);
     block(finish);
                                            .
                                            .
                                            .
                                            wakeup(driver,finish);
     call stop_meters;
     call logout_force(slave);              .
     call print_meters;                     .
     end;                                   .
```

Figure IV. Proposed Driver - Slave Coordination.

very small configuration: one processor, 256K words of secondary memory, and two disk drives. This makes it a very artificial environment in which to meter. Therefore, before any results from the development machine are used to make assumptions about relative system performance, the validity of such generalizations must be investigated.

5 processes represent a rather artificial environment too.

Variations in the benchmark:
random element: user think time
(causes the immediate 'bottleneck' to shift from run to run)
system state
(e.g. disks are in different position every time a benchmark is executed)