[illegible]

UCRL--89202

DE84 012603

Evaluating and Tuning System Response in the MFTF-B Control and Diagnostics Computers

Robert L. Palasek
David N. Butner
Earl G. Minor

Proceedings of the 10th Symposium
on Fusion Engineering
Fusion Energy Conference
Philadelphia, PA
December 5-9, 1983

November 30, 1983

Lawrence Livermore National Laboratory

EVALUATING AND TUNING SYSTEM RESPONSE IN THE MFTF-B CONTROL AND DIAGNOSTICS COMPUTERS

Robert L. Palasek, David N. Butner, Carl G. Minor
Lawrence Livermore National Laboratory
P. O. Box 5511, L-535
Livermore, CA 94550

Abstract

The software system running on the Supervisory Control and Diagnostics System (SCDS) of MFTF-B is, for the major part, an event driven one. Regular, periodic polling of sensors' outputs takes place only at the local level, in the sensors' corresponding local control microcomputers (LCC's). An LCC reports a sensor's value to the supervisory computer only if there was a significant change. This report is passed as a message, routed among and acted upon by a network of applications and systems tasks within the supervisory computer (SCDS). Commands from the operator's console are similarly routed through a network of tasks, but in the opposite direction to the experiment's hardware. In a network such as this, response time is partially determined by system traffic.

Because the hardware of MFTF-B will not be connected to the computer system for another two years, we are using the local control computers to simulate the event driven traffic that we expect to see during MFTF-B operation. In this paper we show how we are using the simulator to measure and evaluate response, loading, throughput, and utilization of components within the computer system. Measurement of the system under simulation allows us to identify bottlenecks and verify their unloosening. We also use the traffic simulators to evaluate prototypes of different algorithms for selected tasks, comparing their responses under the spectrum of traffic intensities.

[Work performed under the auspices of the U. S. Department of Energy by Lawrence Livermore National Laboratory under contract number W-7405-ENG-48.]

Introduction

It is axiomatic that new systems are badly out of tune. Beizer [1] enumerates several causes for this which we paraphrase.

1. Wild guesses and even careful analyses of load characteristics made by the designer turn out to be wrong.

2. The system's specification has changed throughout the development period; the basic load assumptions were not changed to suit.

3. There were factors left out of all models and analyses that were dominant; other things felt to be important became inconsequential after astute design.

4. The system itself changes the user's behavior which then changes the load characteristics.

A fifth common reason, not admitted by Beizer, is that designers consciously admit inefficient components, budgeting the time for the efficient replacement to occur after the delivery of the initial system.

So in some sense tuning is an iterative step, correcting mismatches between design assumptions and actual usages, cycling between the design and evaluation of stages of a software project. It necessarily must wait until the software and hardware have been debugged and integrated, after the system is operating and all its parts are playing together.

The experience gained during the MFTF Technology Demonstration [17, 6] reinforced the idea that tuning is part of systems integration. Algorithms for the updates of display fields on CRT monitors, the routing of commands to and reports from the experiment's hardware, software filtering, all were correct but were found to be intolerably inefficient when subject to the actual traffic load of operating the MFTF hardware. Diagnosis of the problems typically took between a half day and three days. Correction times ranged from two days to two months. Some problems needed more time to correct than there was remaining in the Technology Demonstration. These problems were either tolerated, or their corresponding features were excised from the system as unessential. The neophytes among us who hadn't anticipated the need for initial tuning experienced unwarranted chagrin when first confronted with slow response times.

This, and the recognition that the bottlenecks hadn't been completely shaken out by the conclusion of the hardware tests, led us to develop a traffic simulator that is presently being used to tune the supervisory control system as it evolves from the MFTF configuration to the MFTF-B configuration.

The Simulator

The local control computers cyclically poll instrumentation sensors through CAMAC once per second and compare the latest value read with the last value reported to the supervisory computer. When the LCC recognizes a significant change (significance is a delta parameter associated with the sensor), it sends a new report to the supervisory computer. With low pass filtering enforced, reports from any individual sensor are spread apart in time. It is not precise but still reasonable to model the series of reports from an individual sensor as a Poisson process. One can prove that the <u>aggregate</u> of reports over many sensors do form a Poisson process whose rate λ is equal to the sum of the rates of the individual reports. We accept this Poisson process as the characterization of the major source of input to the SCDS control system.

Once we've done this it becomes conceptually straightforward to simulate the input. When the LCC

[illegible]

time through its monitor loop. Instead of polling the device and checking for significant change, it samples a pseudo-random number generator, like tossing a weighted coin, and decides whether this device needs to "send a report." The probability of sending a report is set to λ_i. The estimated mean frequency of a device reporting during some operating regime. For example for MFTF-B, calculations and designs show that the superconducting magnets will cool in 120 hours. The temperature monitoring will have a "significance delta" of 1% of the expected range; so barring any transients, each magnet case temperature sensor should report about once every 3800 seconds during cooldown. The λ_i for these sensors is set to 1/3800.

For the magnet power tests of MFTF Tech Demo, each monitor report was logged by SCDS for later analysis. From the 600,000 records taken over 5 charging cycles during the course of two weeks, report frequencies were determined by class for the temperature sensors, strain guages, pressure guages, selected voltages, and currents. Each pseudo-sensor was assigned a report rate so that the sum of the rates in any class would match the recorded net rate for that class. The total aggregate rate seen during the last power test, the one for which the most tuning had been done, was .72 reports per second. This is the benchmark traffic mix being used to study and measure the existing system and presently being used to evaluate software modifications in the control part of SCDS.

Measurement and Evaluation

Tuning is usually done in light of a goal, typically minimum delay, maximum throughput, or maximum utilization. Optimization of any one of these usually implies a degradation of another. For example, fastest response will occur on an otherwise idle or unloaded system. Full utilization of some resource such as CPU, disk or some other I/O device implies that transactions must be queued up and waiting at that device, ready to use it the moment it becomes free. This queueing contributes to the delay in completion of the average transaction.

Analytic queueing models [16] are used to evaluate the tradeoffs among these conflicting goals. Relatively simple models have recently been developed which are easy to apply and have proved to be amazingly robust despite many violations of stochastic assumptions of the models [15, 3]. The particular model we chose for the control and diagnostic system is the open system queueing network of [10] also described in other works on Operational Analysis [4]. Figure 1 is a diagram of the queueing network, Figure 2 gives some definitions and terms, and Figure 3 shows the algorithm and its inputs and outputs.

The model's inputs are the service demands $D_i(\lambda)$ which is the total time needed at service center i by each transaction when the arrival rate is λ. The model's outputs are utilizations, average residence times at each service center (waiting in queue and receiving service), and the average number of transactions at each service center. The service demands measured for a representative instrumentation monitor report are given in Table 1. We discuss below how the measurements were made.

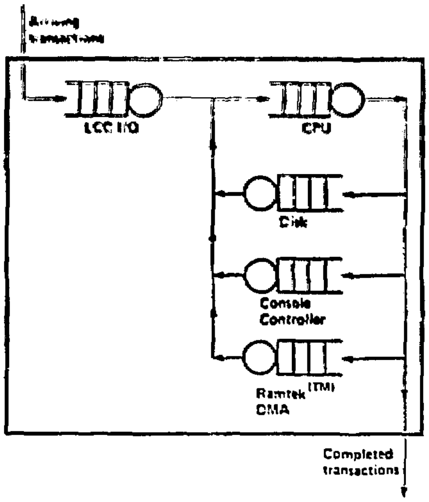

Figure 1. Queueing network representation of the MFTF-B Supervisory Control System. All service centers queue arriving transactions when busy.

T	-	Some sufficiently long time interval of observation.
X	-	Throughput. Number of completed transactions per unit time.
R	-	Residence time (or response time). Time a typical transaction spends in the system.
U_i	-	Utilization. Proportion of time a service center is not idle.
D_i	-	Service demand. Service time needed by a typical transaction at service center i.
R_i	-	Residence time. Time a typical transaction spends at center i, both receiving service and waiting in queue.
λ	-	Arrival rate of transactions. In a stable system $X = \lambda$.
N_i or Q_i	-	Population at center i. Average number of transactions both receiving service and in queue at service center i.
N	-	Population. Average number of transactions in the system.

Figure 2. Important terms for operational analysis of queueing networks.

```
Inputs:       λ, Arrival rate,
              D_i(λ), Service demands
                  for each center.

Outputs:      U_i, R_i, N_i
              λ saturation, X, R, N

Assumptions:  D_i is constant over
               λ, and N_i.
                λ < λsaturation

Algorithm:
               λsat    := 1/D_MAX
               X(λ)    := λ
               U_i(λ)  := λ · D_i
               R_i(λ)  := D_i/(1-U_i(λ))
               Q_i(λ)  := U_i(λ)/(1-U_i(λ))
               R(λ)    := Σ R_i(λ)
               N(λ)    := λ ·R(λ)
```

Figure 3. Open queueing network model.

The model predicts average response time and throughput. Under the assumption that service demands are constant with respect to arrival rate, it predicts capacity. The device having the largest *service demand is considered the bottleneck device.* It is the one which will saturate (its utilization becomes 1.0) first when the arrival rate is increased. Any additional load will just queue up at this center while throughput remains at capacity. In Table 1 we see that the CPU is the bottleneck device under this criterion. Hence, initial tuning efforts are best directed at reducing the CPU demand of each transaction.

The measurements tell us other things as well. If we assume that a transaction cannot be receiving service simultaneously from more than one service center, then the minimum possible response time, seen when a transaction enters an idle system, is the sum of the demands. Tuning to reduce the demand at any of the centers will reduce the minimum response time by that amount, although reducing the demand at a non-bottleneck service center will have no effect in increasing the system's capacity. We also note that the demands of the latter two devices (service centers) are insignificant with respect to the first three. In this system it is not worthwhile spending any tuning effort there. However trading CPU time for I/O time at these devices, if feasible, would be worthwhile.

The service time at the disk results from designs responding to limited memory in the original system. We estimate that with the upgraded computers having a larger address space, we should be able to eliminate 80% of those disk accesses. The associated CPU activity for each disk access will also disappear. Benchmarks have shown that upgrading the CPU to a faster one has resulted in a 25% decrease in service demand.

A larger and faster disk had a similar effect. The seeks were fewer and faster. Another benchmark has shown we can get another 20% reduction in CPU demand with an optimizing compiler. Computer scientists generally agree that important reductions in processing time are gained by identifying inefficient algorthims and restructuring the processing there. This is the basis of a major tuning effort reported elsewhere [9].

Study of Ramtek activities show that it is unlikely that any improvement will be found there. Decrease in demand there would only be obtained by replacing it with newer, faster hardware. From a capacity and response standpoint, replacement isn't warranted until the CPU service demand decreases and approaches that of the Ramtek--or usage patterns change, increasing Ramtek demand to levels greater than those of the other devices. When all the demands are balanced, upgrading would consist of replacing all devices with faster ones or else replicating the whole system.

Table 1. Statistics from Benchmark Measurements of the SCDS Magnet System Operating Software.

	Model Inputs Arrival Rate =.72	Model Outputs Averages		
Service Center	Service Demands per transaction *(seconds)*	Utilization	Residence Time	Number at Center
CPU	.64	.46	1.19	.86
Disk	.38	.28	.53	.38
Ramtek DMA	.21	.15	.24	.18
LCC	.03	.02	.03	.02
Console Controller	.006	.004	.006	.004
		Throughput $X = \lambda = .72$ Capacity = $\lambda_{SATURATION} = 1/D_{MAX} = 1.56$ Response Time = ΣR_i = 2.00 seconds.		

Measuring Service Demands

To get service demands, we installed the accounting package that comes with the computer's operating system. For each executing task, it collects CPU time used both by the task and by the operating system running on the task's behalf. It also counts the number of I/O's and accumulates the total number of bytes transferred for each device that the task uses. When the task completes execution, the operating system writes the accumulated statistics into an accounting file from which billing statements are periodically generated. We rewrote the report generator to break out and sum the statistics by task name rather than by account number.

We wrote a second program that would dump the partially accumulated statistics for all currently running tasks in a format identical to the accounting records so that they could also be processed by our report generator. With these we were able to make measurements as follows. The magnet applications tasks were started on the SCDS computer and the simulator-benchmark was started on the LCC. The measurement period was 1000 seconds. At t=0, we saved a snapshot of the partially accumulated statistics and also reassigned the accounting file so that any task completion records made during the measurement period would be diverted there. At t=1000, we took a second snapshot of the partially accumulated statistics of the then active tasks and closed the file which collected the task completion records. Net usages were found by subtracting the t=0 statistics from the sum of the t=1000 statistics and the task completion statistics.

I/O counts of LCC messages told us the number of monitor reports from which we derived the average resource demand per monitor report. CPU time came directly from the report. Disk transfer time was derivable from the number-of-oytes statistics. The operating system kept statistics elsewhere on average seeks and latencies. From this we calculated that the average disk access took 13.25 msec. with a variation of less than .5 msec. Message sizes from the less busy devices were of fixed size and transmitted serially at 9600 baud, and so their times were simply derived.

The service demand times for the Ramtek were more difficult. The actual transfer times depend on the values of the data transferred. The Ramtek (TM) is a graphics system that drives between 5 and 8 CRT's at an operator console. Each CPU in the SCDS system drives one Ramtek which in turn drives all the CRT's on one operator console. The CPU sends a buffer full of 16-bit Ramtek commands. The Ramtek accepts and executes these commands, one at a time; it does not accept the next command from the DMA until it has finished interpreting the previous one. The commands have a wide variation in execution times. We used a logic analyzer to plot the hardware DMA-not-available signal and from that estimated a mean value of 90 msec. for a buffer transfer. Typically there were two I/O's per update; the first selected the CRT screen, the second actually redrew the graphics field on the screen. The Ramtek is considered busy from the start of the first I/O to the end of the second. (See Figure 4.)

We calibrated some of the statistics of the accounting package with hardware monitors and feel there is enough discrepancies in others to warrant their calibration also. We are presently building a hardware circuit that will measure hardware utilization directly so that we can validate the accounting statistics (Figure 5).

Reporting on itself, the accounting package used 1.5% of the CPU usage and 15% of the disk usage (checkpointing contributed to 2/3 of this). The data shown in Table 1 is net demand after the accounting package overhead was discounted.

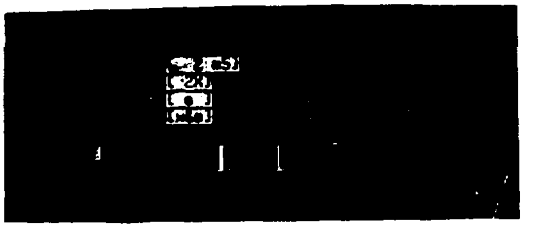

Figure 4. Busy time of Ramtek I/O device for two typical transactions. The waveform on the left represents an update to the date-time displays on all CRT's. These updates are periodic, every two seconds. The one on the right corresponds to a data field update; updates of this kind are aperiodic.

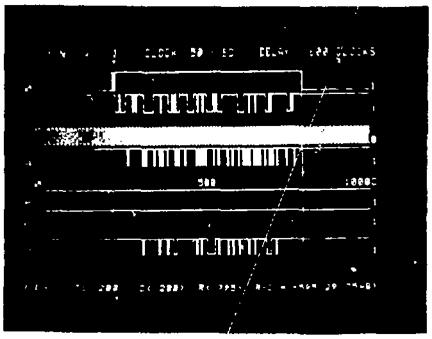

Figure 5. Activity burst (busy time) in the CPU. Row 0--CPU busy. Row--1 Instruction fetches. Row--3 Processor clock. Rows 4 and 7--Other memory timing signals. Integrating the busy signal (and that of Figure 4) over a long period of time serves as calibration for the software accounting package.

Besides their use in capacity planning, the accounting statistics uncover tuning problems. One data base system task was found to be doing three times as many disk accesses as its author anticipated. Examination showed a flag was being incorrectly set, causing memory resident pages to be unnecessarily written through to the disk. This is not a computational error that can be detected by typical testing methods.

We believe that the information given by the accounting package is well worth the price of system overhead.

Tuning the Code

The lore of code improvement has two fundamental principles whose first demonstration is commonly attributed to Knuth [7]. I. More than fifty per cent of the time is spent in four per cent of the code. II. The location of that four per cent is not intuitively obvious. Principle II demands that the code must be instrumented to determine what is actually being executed. Several good references thoroughly enumerate the different ways this can be done [13, 5]. Our first choice, and what we expect to be our most important tool, is to hang a hardware monitor (in our case a logic analyzer) on the address bus of the computer and tally the frequencies of instruction fetches over ranges of code. The hardware monitor is non-invasive; it never introduces bugs into fragile code as a software probe sometimes does. It has no software overhead in collecting and storing statistics and so we needn't worry about having to compensate for artifacts. Thirdly, we can observe parts of the operating system that are normally unavailable to software probes.

Figure 6 shows an X-Y plot of activity on the high order bits of the address bus. The coordinates of the dotted horizontal line near the middle of the picture indicate heavy usage of a group of routines in one re-entrant run time library. Some of these routines are being tuned. The need for others has been re-examined with the result that references to them are being dropped from the applications codes.

We have found a logic analyzer that can tally address hits in any of eight arbitrary ranges for runs of 1024 instruction fetches. With this feature we can profile instruction fetches to any level of detail by divide-and-conquer tactics.

The feature operates on a sample-tally-display cycle, and unless the CPU is busy 100% of the time, the sampling is biased toward the initial instructions of the CPU-busy epochs. Even then, there is the problem of beats due to the analyser and the software having some common divisor of their periods. We intend to replace our ad hoc avoidance of these problems with an external trigger that will schedule the sampling to be fair.

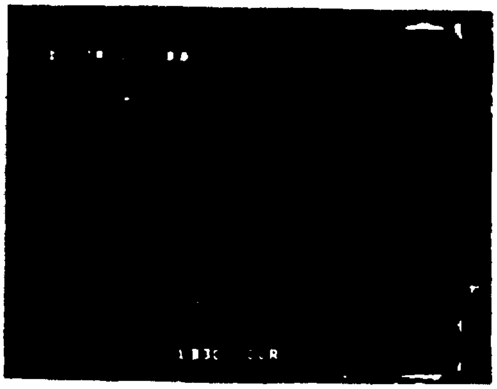

Figure 6. Map of addresses of instruction fetches. Y-axis is high order bits of address bus. X-axis is middle order bits of address bus. Dots represent multiple instruction fetches from those memory neighborhoods.

Particular addresses are recognized as states by the logic analyzer. We make timings between the recognition of the first address and the recognition of the second. We use this to verify that subroutines have been speeded up by selected tuning. This timing capability also allows us to verify or disprove hunches about what might be behaving badly due to marginal design. In one case, the measured execution time of one task, both from the accounting statistics and from the logic analyzer, suprised its author. Profiling showed it to be spending 20% of its time in a single routine (another author's first programming effort in the implementation language). The code was being executed in anticipation of an on-line trace, once used for testing but presently unused. Excising that and the related trace code cut the execution time by 5/6. We are just gaining experience with this technique, and it remains to be seen whether many small improvements on "the other 90% of the code" will have a significant effect on execution time.

Acknowledgement

The authors gratefully acknowledge the contributions of Ronald Bennett, Timothy Letendre, Bron Nelson, Donna Nowell, Robert Pancotti, Michael Toback, and Robert Wyman to this work.

References

1. Beizer, Boris, Micro-Analysis of Computer System Performance, VanNostrand Reinhold, 1978.

2. Butner, D. N., "MFTF Supervisory Control and Diagnostic System Hardware", in Proceedings of the 8th Symposium on the Engineering Problems of Fusion Research, 1980, pp. 2011-2016.

3. Buzen, J. P., Denning, P. J., "Measuring and Calculating Queue Length Distributions", IEEE Computer, v.13, #4, April 1980.

4. Denning, P. J., Buzen, J. P., "The Operational Analysis of Queueing Network Models", ACM Computing Surveys, v. 10, #3, September 1978.

5. Ferrari, D., Computer Systems Performance Evaluation, Prentice-Hall, 1978.

6. Forsberg, H., "Technology Demonstration Report for the Mirror Fusion Test Facility", UCRL-53386, Lawrence Livermore National Laboratory, Dec., 1983.

7. Knuth, D., "An Empirical Study of FORTRAN Programs", Software--Practice and Experience, 1, pp. 105-133, 1971, John Wiley, Great Britain.

8. Labiak, W. G., Minor, E. G., "Software Design of a General Purpose Data Acquisition and Control Executive", IEEE Trans. on Nuclear Science, v. NF-28, #5, October, 1981.

9. Lang, N. C., Nelson, B. C., "An Improvement in the Data Base Notification Response Times", 10th Symposium on Fusion Research, December, 1983.

10. Lazowska, E. D., Zahorjan, J., Grahm, G. S., Sevcik, K., Quantitative System Performance, Computer System Analysis Using Queueing Network Models, Prentice-Hall, In Preparation (1983).

11. McGoldrick, P. R., "Supervisory Control and Diagnostics System Distributed Operating System," in Proceedings of the 8th Symposium on the Engineering Problems of Fusion Research, 1980, pp. 2000-2002.

12. Ng, Walter, C., "Overview of the MFTF Supervisory Control and Diagnostics System Software," in Proceedings of the 8th Symposium on Problems in Fusion Engineering, 1979.

13. Nutt, G. J., "Tutorial: Computer System Monitors", IEEE Computer, November, 1975.

14. Ross, Sheldon, Introduction to Probability Models, Academic Press, 1972.

15. Spraggins, John, "Approximate Techniques for Modeling the Performance of Complex Systems", Computer Languages, v.4, pp. 99-129, 1979, Pergammon Press, Great Britain.

16. Spraggins, John, "Analytic Queueing Models," in IEEE Computer, v.13, #4, April, 1980.

17. Wyman, Robert H., "A Report on the Experience with the Supervisory Control Diagnostics System of MFTF-B", 10th Symposium on Fusion Research, December, 1983.