A major purpose of the Technical Information Center is to provide the broadest dissemination possible of information contained in DOE's Research and Development Reports to business, industry, the academic community, and federal, state and local governments.

Although a small portion of this report is not reproducible, it is being made available to expedite the availability of information on the research discussed herein.

CONF-840869--1

LA-UR -84-1561

CONF

NOTICE

PORTIONS OF THIS REPORT ARE ILLEGIBLE. It has been reproduced from the best available copy to permit the broadest possible availability.

Los Alamos National Laboratory is operated by the University of California for the United States Department of Energy under contract W-7405-ENG-36

TITLE: THE EFFECT OF QUEUEING DISCIPLINES ON RESPONSE TIMES IN DISTRIBUTED SYSTEMS

LA-UR--84-1561

DE84 012639

AUTHOR(S): Elizabeth Williams, C-8

SUBMITTED TO: 1984 International Conference on Parallel Processing
August 21-24, 1984, Bellaire, Michigan

DISCLAIMER

MASTER

Los Alamos National Laboratory
Los Alamos, New Mexico 87545

FORM NO 836 R4
ST NO 2629 5/81

THE EFFECT OF QUEUEING DISCIPLINES ON RESPONSE TIMES IN DISTRIBUTED SYSTEMS

Elizabeth Williams[†]
Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712

Abstract – A distributed program consists of processes, many of which can execute concurrently on different processors in a distributed system of processors. When several processes from the same or different distributed programs have been assigned to a processor in a distributed system, the processor must select the next process to run. The question investigated is: What is an appropriate method for selecting the next process to run? Standard processor queueing disciplines, such as first-come-first-serve and round-robin-fixed-quantum, are studied. The results for four classes of queueing disciplines tested on three problems are presented. These problems were run on a testbed, consisting of a compiler and simulator used to run distributed programs on user-specified architectures.

1. Introduction

When several processes from the same or different distributed programs have been assigned to a processor in a distributed system, an important design question is how a processor selects the next process to run. This problem has not been considered in a distributed environment. An interesting question arises: How do the processes at other processors and communication delays in the system impact the selection of the next process to run? As a beginning study we have investigated the standard queueing disciplines - first-come-first-serve, round-robin-fixed-quantum, preemptive priority, and nonpreemptive priority - in a distributed environment. The study shows that the response time metric can differ by 50% with different choices of queueing disciplines for three problems.

The queueing disciplines were studied with several problems that represent three important classes of problems. The partial differential equation solver is based on an iterative grid technique that is similar to those used in multidimensional applications such as weather prediction, structural mechanics, hydrodynamics, heat transport, and radiation transport. The centralized monitor has the typical tree structure of hierarchically designed applications. The producer-consumer pairs represent a multiprogramming environment in the distributed system and are representative of a large class of problems.

In Section 2 a model of the distributed architecture and the distributed language are described. The metric for comparing the performance of the different queueing disciplines and a description of the testbed are given in Section 3. In Section 4 we give a heuristic for assigning priorities for the priority dependent queueing disciplines. Section 5 describes the distributed programs and architectures on which each problem executes. The results are given in Section 6.

2. Model of Distributed Computing

2.1. Distributed Architecture

The distributed architecture is characterized by the number of processors, the speed of each processor, the queueing discipline at each processor, and the lines that connect the processors. The lines may have different capacities, lengths, and error rates. The processors have no shared memory and they communicate only by messages. We assume that any processor can communicate with any other processor by routing messages through intermediate processors over fixed paths.

[†]Present address: Computer Systems Group, C-8, Los Alamos National Laboratory, Los Alamos, New Mexico 87545

2.2. Distributed Language

A program in the distributed language consists of processes that communicate and share data by using messages. The language is similar to CSP, which is described in [2]. The language uses synchronous (blocking) communication primitives; the sending process cannot proceed until the receiving process is ready to receive the message. For each message sent at the program level, there are two messages sent at the protocol level that implements the language. In this language there is a static number of processes. Dynamic creation of processes is simulated by a process beginning execution only after some other process sends it a message.

2.3. Terminology

We define **virtual line time** for a message between two processors connected directly by a line as the product of the actual time to move the message over the line and a constant derived from line reliability and the overhead of lower level protocols. The actual time to move the message over the line is the usual function of message length in message units (packets), number of bits per message unit, line capacity, and line length. Virtual line time does not include the time a message waits to use the communication subnet. Virtual line time for a message between two processors is the sum of the virtual line times for the lines on the route. Currently in local area networks, lower level protocols executing in the processors usually reduce the physical line capacity by at least a factor of 10 for any message [1]. Virtual line time reflects this effective line capacity.

The **message delay** of a process for a synchronous communication as in CSP is a function of virtual line time, queueing at the port queues on the route in a store and forward network and the processing, waiting, and queueing time of the corresponding process at its processor. Message delays can be very large compared to a process's processing time between communications.

In the testbed **1 unit of time** can be thought of as 1 μs. For local area networks where processors are 1 km apart, transmission rates of 10 Mbit/s are common. For a packet of 255 bits it takes approximately 29 μs to send a packet over the line. With the factor of 10 or more for lower level protocols, 300 time units is a reasonable number for virtual line time in this model of a local area network.

For each problem in this paper, we assume all the processors have the same speed, all lines are identical, and a message unit is 256 bits. We also assume that on any simulation run all processors have the same queueing discipline. These assumptions are made to isolate the effects of the choice of queueing discipline from other system variables.

3. Testbed and Metric

The metric for comparing various queueing disciplines is defined as follows. All the processes of a distributed program are assumed to start at time zero. Each process i terminates at some time, t(i). The metric is the sum over N processes of the termination times t(i) divided by N, and is termed the average of the process termination times (APTT). APTT reflects both the instruction processing requirements of processes and the message delays. Total time, defined as the maximum t(i), is not always a good metric for comparing queueing disciplines, because when message

delays are very small, total time is comparable for all queueing disciplines.

The testbed runs distributed programs coded in the distributed language mentioned above, which is similar to CSP. In addition to the distributed program, the testbed also requires a specification of the distributed architecture. The testbed consists of a compiler, interpreter, and simulator. The compiler produces pseudo-instructions for the hypothetical processors in the distributed system. The interpreter executes the pseudo-instructions. The simulator manages the interpreter, processor queues, and port queues and executes protocol routines. The simulator is based on the work presented in [4] and was validated extensively using commercial analytical and simulation packages [3,5].

4. Queueing Disciplines

The queueing disciplines tested were first-come-first-serve (FCFS), round-robin-fixed-quantum (RRFQ), nonpreemptive-priority (NPP), and preemptive-priority (PP) [4]. The two priority disciplines NPP and PP must assign priorities to the processes. In a PP discipline if an expected message arrives for a blocked process of higher priority, the blocked process preempts the currently running process. In the following discussion we give a heuristic for assigning priorities.

Generally we have observed that scheduling a single processor in a distributed architecture must be analyzed considering both the single processor (local component) and the distributed environment (global component). Our heuristic for assigning priorities is given as follows:

- Processes that communicate across a line are assigned high priority (highest priority when message delays are large since the global component is more important).
- A process on which several other processes may wait (a bottleneck process) is assigned high priority (highest priority when message delays are small since the local component is more important).
- Any other processes are assigned lower priorities to approximate shortest-remaining-time-first (SRTF) [4].

Thus a good priority discipline should generally give highest priority to those processes communicating across a line in order to minimize the processor idle periods and thus to finish executing all processes at the processor sooner. The discipline should be preemptive so that messages over the line can be received by the corresponding process as quickly as possible. Choosing priorities using this heuristic is demonstrated in the problems in the next section.

A priority discipline with priorities assigned as described above is denoted by PPg for preemptive priority and NPPg for nonpreemptive priority. A preemptive priority discipline with priorities assigned in such a way as not to follow the heuristic given above is denoted by PPp; processes that communicate across lines and bottleneck processes are assigned lowest priority, and all the other processes are assigned highest priority. We have found that PPg usually does better than FCFS, RRFQ, PPp, and NPPg; PPp does the poorest.

5. Problems

The problems tested are a partial differential equation solver (PDE), a centralized monitor (MONITOR), and a system of five producer-consumer pairs (PC's). For each problem we present a brief description of the program and a figure that represents the distributed program, architecture, assignment of processes to processors, and priorities for both PPg and NPPg. Each process is represented by a circle with the process number in the circle; the total instruction processing time requirement per process is given below each circle. The priority for a process is given above each circle. The number and average size in message units of messages sent at the program level between two communicating processes is given above each line as the ordered pair (number,size). Values for communication and processing time are obtained by running the program on the testbed with any assignment and architecture; for these programs these quantities are independent of the architecture and assignment. Circles enclosed in a box mean that the enclosed processes are assigned to one processor. For each problem the processors are identical and the virtual line time for a message unit is the same between pairs of processes that must communicate over a line.

5.1. Partial Differential Equation

We solve Laplace's partial differential equation (PDE) on a grid with the outer edges of the grid given as boundary conditions. The iterative method used is Gauss-Seidel. The grid is partitioned into subgrids where each subgrid is some number of contiguous rows. Each subgrid is solved by a process in the same way a sequential program would solve the entire grid. A grid value is computed as the average of its four adjacent neighbors; thus, to compute a row of values, the two adjacent rows are required. Hence, a process must request the two rows contiguous to its subgrid from its two neighboring processes.

Figure 1 shows the structure of the problem that runs on two processors. The two processors are connected by a line with virtual line time for a message unit set at 592 time units. In previous work we found that the assignment indicated in Figure 1 is best for this architecture [5].

All processes are comparable; there is no bottleneck process because each process is logically equivalent and computes an equal number of rows. Since each process must execute one time per Gauss-Seidel step over the same size subgrid, there is no need to assign priorities to approximate SRTF. The two processes that communicate over the line are given highest priority. For PPg and NPPg, processes 3 and 4 were assigned highest priority at 1.0; the others were assigned lower priority at 2.0. For PPp, processes 3 and 4 were assigned lowest priority at 2.0 and the others were assigned highest priority at 1.0.

5.2. Centralized Monitor

The centralized monitor consists of a resource process and three groups; each group consists of a requester process and its three user processes. Each user process executes some given amount of time and then makes a request to use the resource through its requester process. The requester process passes the user request on to the resource process. This is repeated 20 times before a user terminates. The processing times per iteration were chosen so that (1) there is a small, medium, and large processing user process at each processor and (2) the sum of the processing time of the users at each processor is approximately the same at each processor.

Figure 2 shows the structure of the centralized monitor that runs on four processors. Processor 4 is connected directly to processors 1, 2, and 3. Each line has a virtual line time of 58 time units for a message unit. In previous work we found that the assignment indicated in Figure 2 is best for this architecture [5].

The requester processes are 10, 11, and 12. A requester process has high priority because it is a bottleneck and also because it communicates over a line. The user processes - 1 through 9 - at each processor are not identical because of differing processing requirements. The user processes are assigned priority using the average processing time between I/O statements to estimate CPU bursts and thus to approximate SRTF. For PPg and NPPg, requester processes 10, 11, and 12 get priority 1.0; user processes 1, 4, and 7 get priority 2.0; user processes 2, 5, and 8 get priority 3.0; user processes 3, 6, and 9 get priority 4.0. For PPp, processes 10,

11, and 12 get priority 2.0, while all user processes 1 - 9 get priority 1.0. SRTF is an important component of the priority discipline because a user process with a small burst time can finish earlier than the others and thus decrease APTT.

5.3. Producer-Consumer Pairs

There are five producer-consumer pairs. Figure 3 shows the structure of the problem that runs on two processors. The two processors are connected by a line with virtual line time for a message unit set at 346 time units. Processes 1 to 5 are producers; processes 6 to 10 are consumers. Each pair - (1,6) (2,7) and (3,8) - has one-third the processing requirement of each pair - (4,9) and (5,10). Each producer sends 40 messages to its corresponding consumer.

One pair of processes communicates over the line and both are given highest priority. There are no bottleneck processes in this example. The two pairs with the large processing requirements should get lower priority to approximate SRTF. Priorities for PPg are assigned as follows: processes 3 and 8 get priority 1.0; processes 1, 6, 2, and 7 get priority 2.0; processes 4, 9, 5, and 10 get priority 3.0. For PPp, processes 3 and 8 get priority 2.0; the other processes get priority 1.0. The producer-consumer pairs that are not split across two processors are independent of each other. These pairs can terminate independently of the other pairs; one process waiting on a line cannot cause all the processes on that processor to block as can happen in the other two problems.

6. Results

The results for each program and its architecture are given in Table 1. Of the disciplines tested, PPg is the best while PPp is the poorest. RRFQ always does better than FCFS; this is probably due to its preemptive characteristic. The nonpreemptive priority discipline, NPPg, is poorer than RRFQ for both the PDE and MONITOR problems. The percentage increase in APTT from PPg to PPp as computed by (max APTT - min APTT) / (min APTT) is 32% for PDE, 49% for MONITOR, and 57% for PC's.

7. Conclusion

We have presented the results for five queueing disciplines tested on three problems. The disciplines tested are first-come-first-serve, round-robin-fixed-quantum, nonpreemptive-priority, and preemptive-priority with two sets of priorities. A heuristic is given to assign priorities. We found that the preemptive priority discipline with priorities assigned according to our heuristic was the best discipline tested.

8. Acknowledgments

The author wishes to thank Professor K. Mani Chandy for suggesting this problem and providing valuable guidance during this research. This research was supported in part by Air Force Office of Scientific Research under grant AFOSR 81-0205. This paper was prepared under the auspices of the U. S. Department of Energy.

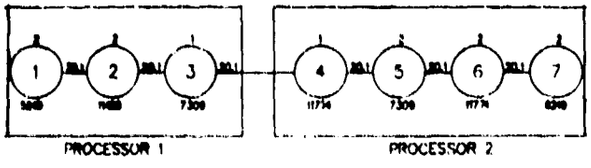

Figure 1. Structure of PDE Problem

References

[1] E. E. Balkovich, Digital Equipment Corporation; David Wood, Mitre Corporation; Dieter Baum, Hahn-Meitner-Institute, Germany; Private Communications, 1983.

[2] C. A. R. Hoare, "Communicating Sequential Processes," *Comm. ACM*, August 1978, pp. 666-677.

[3] *PAWS User's Manual, CADS User's Manual*, Information Research Associates, Austin, TX, 1981.

[4] C. H. Sauer and K. M. Chandy, *Computer Systems Performance Modeling*, Prentice-Hall, 1981, Chapter 7.

[5] E. A. Williams, *Design, Analysis, and Implementation of Distributed Systems from a Performance Perspective*, Ph.D. Thesis, The University of Texas at Austin, 1983.

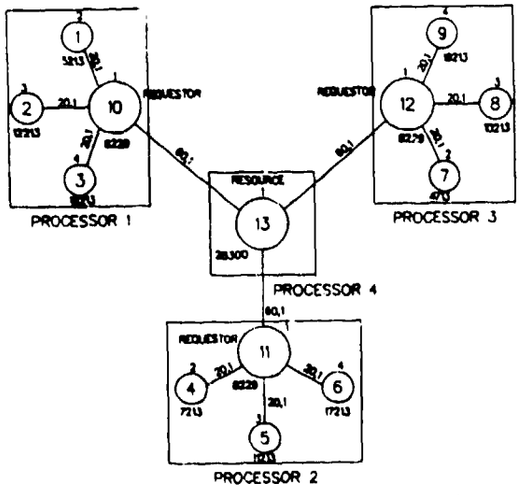

Figure 2. Stucture of Centralized Monitor

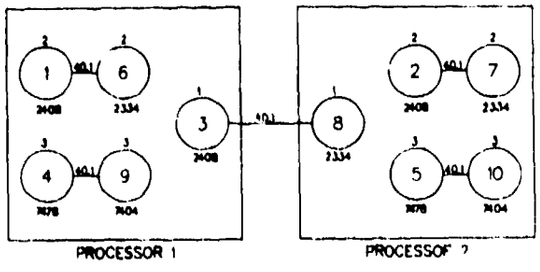

Figure 3. Structure for Producer – Consumer Pairs

APTT for each Problem and Discipline			
Queueing Discipline	PDE	Centralized Monitor	Producer-Consumers
FCFS	57071	57409	23182
RRFQ*	50031	49107	19320
NPPg	54625	57081	16208
PPg	45132	39910	14976
PPp	59038	59439	23461

* Quantum size: PDE-100; MONITOR-75; PC's-80

Table 1