STATE UNIVERSITY OF NEW YORK AT

STONY BROOK

CEAS TECHNICAL REPORT 641

Scheduling Algorithms for Divisible Jobs in a Multiprogrammed Multiprocessor System

S. Bataineh & T.G. Robertazzi

September 30, 1992

Scheduling Algorithms for Divisible Jobs in a Multiprogramed Multiprocessor System

Sameer Bataineh and Thomas Robertazzi, Senior Member, IEEE

Dept. of Electrical Engineering,

SUNY at Stony Brook,

Stony Brook, N.Y 11794

(516)-632-8412/8400

Abstract

Static and dynamic scheduling algorithms for load sharing a divisible job over processors interconnected through a bus are presented and simulated. A comparative study between their performance results is presented. The processing time, waiting time and processor's utilization for the dynamic algorithm is found to be less than for the static one.

Index Terms:

Load Sharing, Divisible Job, Multiprocessors, Multiprogramming.

1 Introduction

Computers enter all aspects of life in today's world. The heavy use of computers made it is necessary to build fast and reliable ones. Computing speed has thus become of great interest among researchers. Parallel processing is an effective way to improve the speed of computation. The highest level of parallelism is achieved in multiprogramed multiprocessor systems. In such systems, more than one processor is used to process one or more different jobs (programs) simultaneously.

Most of the work to date on load sharing has considered algorithms to schedule a number of jobs among n processors in order to achieve the minimum finish (processing) time [4, 5, 6, 7, 8, 9, 10]. Ni and Hwang discussed the case of different classes of jobs where each class is to be processed in a preassigned group of processors [11]. This previous work involved the paradigm of indivisible jobs. Under this paradigm a job can be scheduled and processed by at most one processor. Ravi and Stankovic discussed the effect of communication delays on performance. However, none of the previous work has discussed the communication time as well as the computation time together.

In this paper two algorithms, one static and one dynamic, are presented that schedule different sized divisible jobs among a number of processors in order to minimize the finish time. A divisible job is a job that can be arbitrarily split in a linear fashion among a number of processors. Applications include the processing of very large data files such as occurs in signal and image processing, Kalman filtering and experimental data processing. In addition to the effect of computation time on finish time, we will study the effect of communication time as well. The communication time is the time that is needed to distribute the load to the processors in a multiprocessor system.

The organization of the paper is as follows: in the second section we present the

static and the dynamic algorithms. Comparative studies and performance evaluations are presented in the third section. Finally, the fourth section contains the conclusion.

2 Scheduling Algorithms

2.1 Introduction

In this section we will introduce two algorithms to distribute the load among n processors interconnected through a bus in a multiprogramed multiprocessor environment. The system to be studied is shown in Fig. 5.1. Jobs of different sizes arrive into an infinite sized FIFO queue. The arrival process is Poisson and a job's size is uniformly distributed. An arriving job is not queued if the bus is free and the processors are not operating at their full capacity. Processor i operates in its full capacity if it has n different jobs running simultaneously. If the bus is free and the processors can accept an additional job, the job at the head of the queue will be processed. The server for the queue communicates to each processor in the bus its fraction of that job. The communication time for processor i, $i = 1, 2, \ldots, N$, is proportional to the amount of data that has to be assigned to that processor. Three major factors determine the amount of measurement data assigned to each processor. The first is the computational speed of processor i in the network. Faster processors will receive more data than the slower ones. The second is the order of the load distribution among the processors in the network. The third is how much of its' computational power processor i can allocate to a job. Each processor begins to compute its share of the load once the share has been completely received. Bus propagation delay is ignored. The timing diagram of the system is depicted in Fig. 5.2.

Let us first introduce the following notation.

$\propto_i$: The fraction of measurement data (load) that is assigned to processor i

T_{cp} : The time that it takes the ith processor to process the entire load when $w_i = 1$.

T_{cm} : The time that it takes to transmit the whole load (job) over the bus when $Z = 1$.

w_i : A constant that is inversely proportional to the computation speed of the ith processor. The ith processor can process the entire load in time $w_i T_{cp}$.

Z : A constant that is inversely proportional to the speed of the single bus. The entire load can be transmitted over the bus in time ZT_{cm}.

T_f : The finish time of the process is the time when the last processor finishes processing.

$$T_f = max(T_1, T_2, \cdots, T_n) \tag{2.1}$$

The equations that govern the relations among various variables and parameters in the system are

$$T_1 = \alpha_1 ZT_{cm} + \alpha_1 w_1 T_{cp} \tag{2.2}$$

$$T_2 = (\alpha_1 + \alpha_2) ZT_{cm} + \alpha_2 w_2 T_{cp} \tag{2.3}$$

$$T_3 = (\alpha_1 + \alpha_2 + \alpha_3) ZT_{cm} + \alpha_3 w_3 T_{cp} \tag{2.4}$$

$$T_4 = (\alpha_1 + \alpha_2 + \alpha_3 + \alpha_4) ZT_{cm} + \alpha_4 w_4 T_{cp} \tag{2.5}$$

.

$$T_n = (\alpha_1 + \alpha_2 + \cdots + \alpha_n) Z T_{cm} + \alpha_n w_n T_{cp} \tag{2.6}$$

The fractions of the total measurement load should sum to one:

$$\alpha_1 + \alpha_2 + \ldots + \alpha_n = 1 \tag{2.7}$$

The minimum finish time T_f is achieved when all processors stop at the same time [1]. Intuitively this can be proved by contradiction; if the processors do not all stop at the same time, some will be idle while others are busy and the minimum finish time can be improved by transfering load to the idle processors. Based on this, the optimal values of $\alpha's$ for n processors can be computed by solving recursively the following set of recursions:

$$\alpha_1 + \alpha_2 + \alpha_3 + \cdots + \alpha_n = 1 \tag{2.8}$$

$$\alpha_{n-1} = \frac{\alpha_n (w_n T_{cp} + Z T_{cm})}{w_{n-1} T_{cp}} \tag{2.9}$$

$$\alpha_{n-2} = \frac{\alpha_{n-1} (w_{n-1} T_{cp} + Z T_{cm})}{w_{n-2} T_{cp}} \tag{2.10}$$

.

.

.

$$\alpha_2 = \frac{\alpha_3 (w_3 T_{cp} + Z T_{cm})}{w_2 T_{cp}} \tag{2.11}$$

$$\alpha_1 = \frac{\alpha_2 (w_2 T_{cp} + Z T_{cm})}{w_1 T_{cp}} \tag{2.12}$$

Numerical values can be obtained by assuming $\alpha_n = 1$, determining the other α_i's and then normalizing the α_i's (they must sum to one). Here α_i is solved for by equating T_i to T_{i+1}

2.2 Algorithms

Static

Each processor simultaneously processes n jobs at most, and each job receives $\frac{1}{n}$ of the computational power of the processor. In other words, regardless of the number of jobs in a given processor, each one will get a fixed amount of the computational power. This will lead to underutilization of the processors under light loads.

Dynamic

Each processor simultaneously processes n jobs at most. However, if there is less than n jobs at a processor those jobs share the whole computational power of that processor. For instance, if there is only one job running on a processor, that processor will devote all its power to that particular job. If a new job arrives, the computational power will be equally divided between the two jobs and so on until that processor gets its n^{th} job. Note that a job may not finish at the same time on every processor because the distribution of new jobs causes a non-uniformity of the computational power available to each job. However, a new job is not distributed to the processors until there is spare capacity on every processor.

3 Performance Evaluation

Let us begin by defining the following:

Waiting time is the time that a job arriving to the queue will wait before the server starts distributing it to the processors over the bus.

Processing time : is the time that elapses between the time that a job begins distribution over the bus and the time it exits the system.

Processors utilization: is the average fraction of the multiprocessor's computing power actively working on a job(s).

- In Fig. 3, 4 and 5, the average processing time, the average waiting time, and the processors utilization are plotted respectively against the average arrival time (load). In those plots $w = 1, Z = 1, n = 5$ and the number of processors used is $N = 5$. Also T_{cm} is distributed as U[0,1] and T_{cp} is four times T_{cm}.
  - Fig. 3 shows that the average processing time of the dynamic algorithm is approximately three times better than the static one. It also shows that the processing time is independent of the load. It depends only on the speed of the bus and the speed of the processors.
  - As expected, Fig. 4 shows that as mean interarrival time increases the average waiting time decreases. This due to the low utilization of the bus and processors when the load is low. The waiting time depends on the state of the bus, free or busy, and the state of the processors in terms of whether they can accommodate that job or not. Under low load the two algorithms will have similar bus and processors states. The bus is almost always free and the processors are also almost always able to accommodate a new job. This why the two curves merge into one under very light load.
  - Fig. 5 shows that the processor utilization of the dynamic algorithm is less than that of the static one. This because the average processing time under the dynamic algorithm is less than under the static one. In other words, the time that the processors in the dynamic algorithm spend in processing compared to the total time is less than in the static algorithm.

- In Fig. 6, 7, and 8 the average processing time, the average waiting, and the processors utilization are plotted respectively against the number of processors in the system. In these plots $w = Z = 1$, $n = 5$ and the average arrival rate $= 1$.

  - Fig.6 shows that the dynamic algorithm yields a better processing time than the static one especially if few processors are used. The two curves level off after a certain number of processors. This is because the majority of the load will be delivered to the first few processors. The rest of the processors' share of the load tends to be small and so they will not contribute to a significant improvement in performance. Note that both curves level off at an asymptote of 0.5 as the bus becomes a bottleneck and the asymptotic time to distribute a job over the bus is $Z\overline{T_{cm}} = 1.0 * 0.5 = 0.5$.

  - Fig. 7 shows that when less than 10 processors are used the average waiting time of the static algorithm is larger than that of the dynamic. After a certain number of processors, the average waiting time will be independent of the number of processors for both algorithms. An arriving job then will have no problem in finding available computation power in the processors to use. In other words, it does not have to wait for available computation power. The only source of delay left is whether the bus is free or not. The bus in both algorithms has identical characteristics and so the two curves are identical for large numbers of processors. Using Little's law and the asymptotic result of Fig. 7 it can be seen that the asymptotic mean queue length is $L = \lambda W = 1.0 * 0.26 = 0.26$.

  - The utilization in Fig. 8 decreases as the number of processors increases. It implies that each processor will have less data to work on and so the utiliza-

tion dwindles. Meanwhile, idle time increases. This is a general problem in multiprocessor systems. One would like to use more processors to decrease the finish time at the same time that one would like not to see the utilization decrease.

- The throughput is plotted in Fig. 9 and Fig. 10 against the number of processors and the speed of the bus respectively. For those two plots, $w = 1$, $n = 5$, $Z = 1$ in Fig. 9 and the number of the processors used in Fig. 10 is 5.

  As shown in Fig. 9, the throughput is constant after certain number of processors. Fig. 10 shows that as the bus gets slower the throughput decreases. This is because the nature of the system is such that jobs have first to be served by the bus before getting to the processors. So the throughput can be determined by the bottleneck speed of the bus and the average arrival rate. It does not matter how many processors are present as long as the number of processors are big enough to be able to keep up with the number of jobs delivered by the bus. This why the curve is a straight line in Fig. 9 after certain number of processors. To comprehend the above two observations, imagine a supermarket with one server and N exits. The number of people to exit the supermarket will determined by the efficiency of the server and the number of customers. The number of customers to exit the store could be affected by the number of exits available if the server is able to process quite a lot of customers who will queue in the exit doors. This is very unlikely to happen especially if the doors are wide enough.

  Note that the knee of the curve in Fig. 10 is at $Z = 2$. This occurs as this is the transition point at which the time to send a job through the bus ($Z\overline{T_{cm}} = 2*0.5 = 1.0$) matches the mean arrival rate of jobs (for Fig. 10). If Z is greater than 2, the

bus becomes a bottleneck and throughput decreases. Note also that throughput saturates near 1.0 in Fig. 9 and Fig. 10 as this is the mean arrival rate.

4 Conclusion

Two algorithms, one static and one dynamic have been presented for multiprogramming divisible jobs in a bus connected multiprocessor system. This first look at multiprogramming divisible jobs has demonstrated the feasibility of this concept. It has also demonstrated the need for, and superiority of, dynamic load balancing algorithms in this application.

References

[1] Bataineh, S. and Robertazzi, T.G., "Distributed Computation for a Bus Networks with Communication Delays", *Proceedings of the 1991 Conference on Information Sciences and Systems*, The Johns Hopkins University, Baltimore MD, March 1991, pp. 709-714.

[2] Bataineh, S. and Robertazzi, T.G., "Bus Oriented Load Sharing for a Network of Sensor Driven Processors", *IEEE Transactions on Systems, Man and Cybernetics*, Sept. 1991, Vol.21, No. 5, pp. 1202-1205

[3] Hsiung, T. and Robertazzi, T.G., "Performance Evaluation for Distributed Communication Systems for Load Balancing", SUNY at Stony Brook, College of Engineering and Applied Science Technical Report No. 612, Dec. 17, 1991, Available from T. Robertazzi.

[4] Baumgartner, K.M. and Wah, B.W., "GAMMON: A Load Balancing Strategy for Local Computer Systems with Multiaccess Networks", *IEEE Transactions on Computers*, Vol. 38, No. 8, August 1989, pp. 1098-1109.

[5] Bokhari, S.H, "Assignment Problems in Parallel and Distributed Computing", Kluwer Academic Publishers, Boston, 1987.

[6] Lo, V.M., "Heuristic Algorithms for Task Assignment in Distributed Systems", *IEEE Transactions on Computers*, Vol. 37, No.11, Nov. 1988, pp. 1384-1397.

[7] Ramamrithamm K., Stankovic, J.A. and Zhao, W., "Distributed Scheduling of Tasks with Deadlines and Resources Requirements", *IEEE Transactions on Computers*, Vol. 38, No. 8, August 1989, pp. 1110-1122.

[8] Shin, K.G. and Chang, Y-C., " Load Sharing in Distributed Real-Time Systems with State Change Broadcasts", *IEEE Transaction on Computers*, Vol. 38, No. 8, August 1989, pp. 1124-1142.

[9] Stone, H.S., "Multiprocessor Scheduling with the Aid of Network Flow Algorithms", *IEEE Transaction on Software Engineering*, Vol . SE-3, No. 1, Jan. 1977, pp. 85-93.

[10] Mirchandaney, R. Towsley, D. and Stankovic, J.A., "Analysis of the Effects of Delays on the Load Sharing", *IEEE Transactions on Computers*, Vol. 38, No. 11, Nov. 1989, pp. 1513-1525.

[11] Ni, L.M. and Hwang, K., "Optimal Load Balancing in a Multiple Processor System with Many Job Classes", *IEEE Transaction on Software Engineering*, Vol. SE-11, No. 5, May 1985, pp. 491-496.

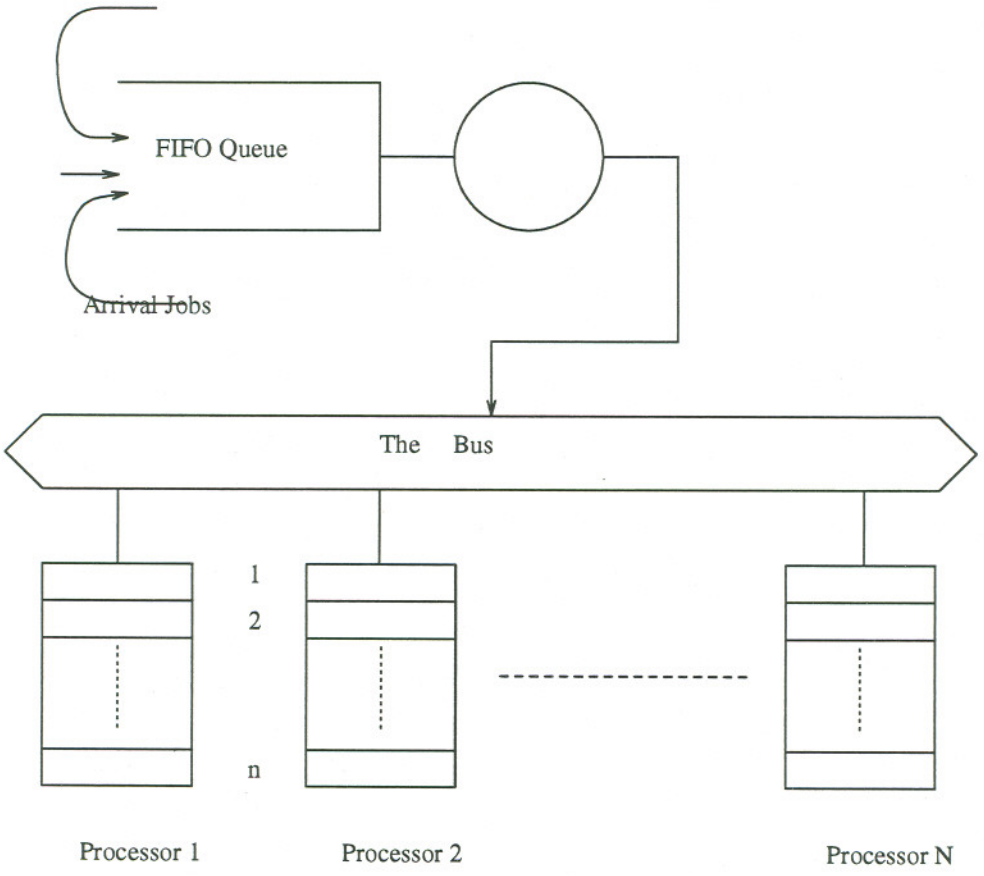

Fig. 5.1: Linear Network With FIFO Queue

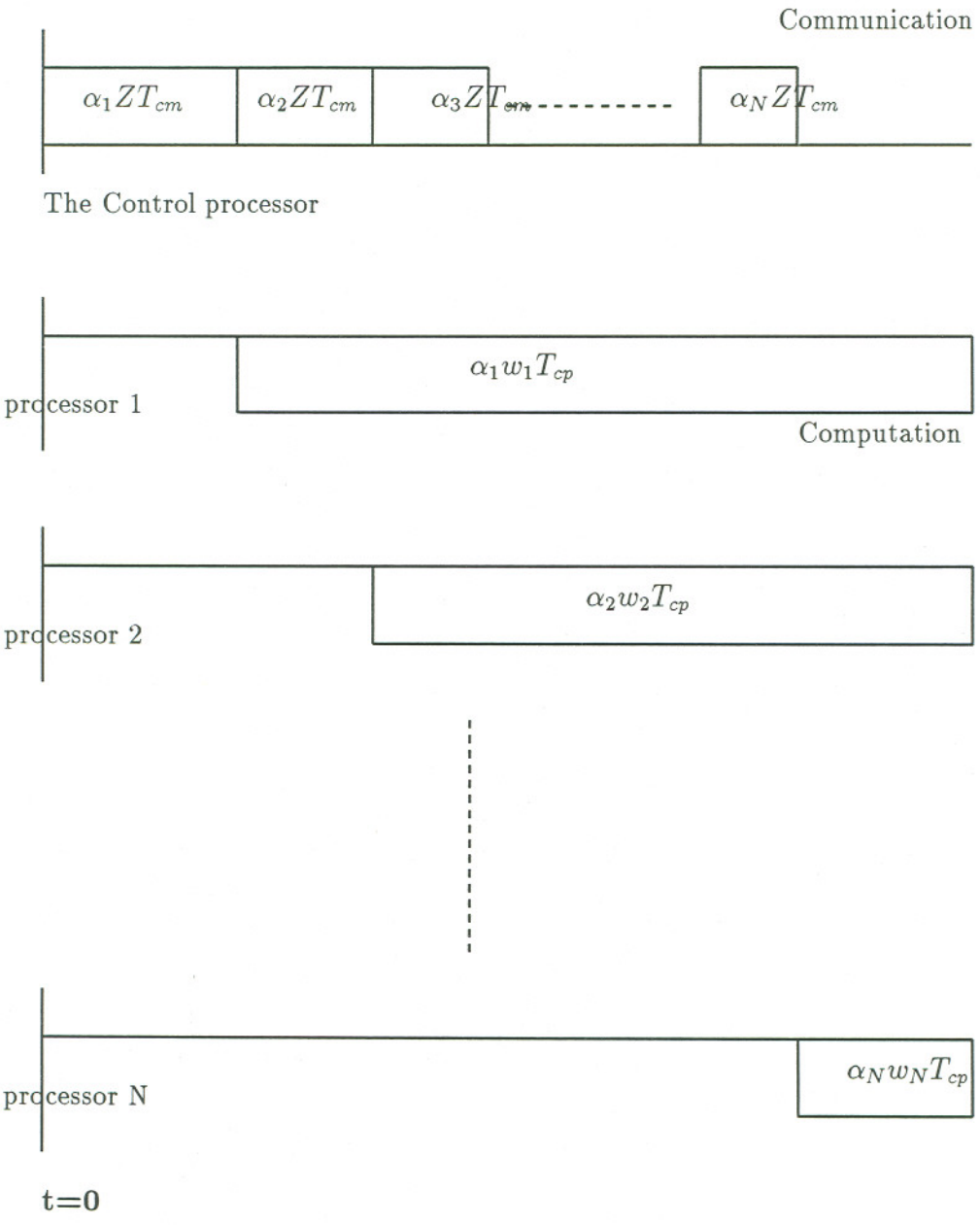

Fig. 5.2: The Timing Diagram

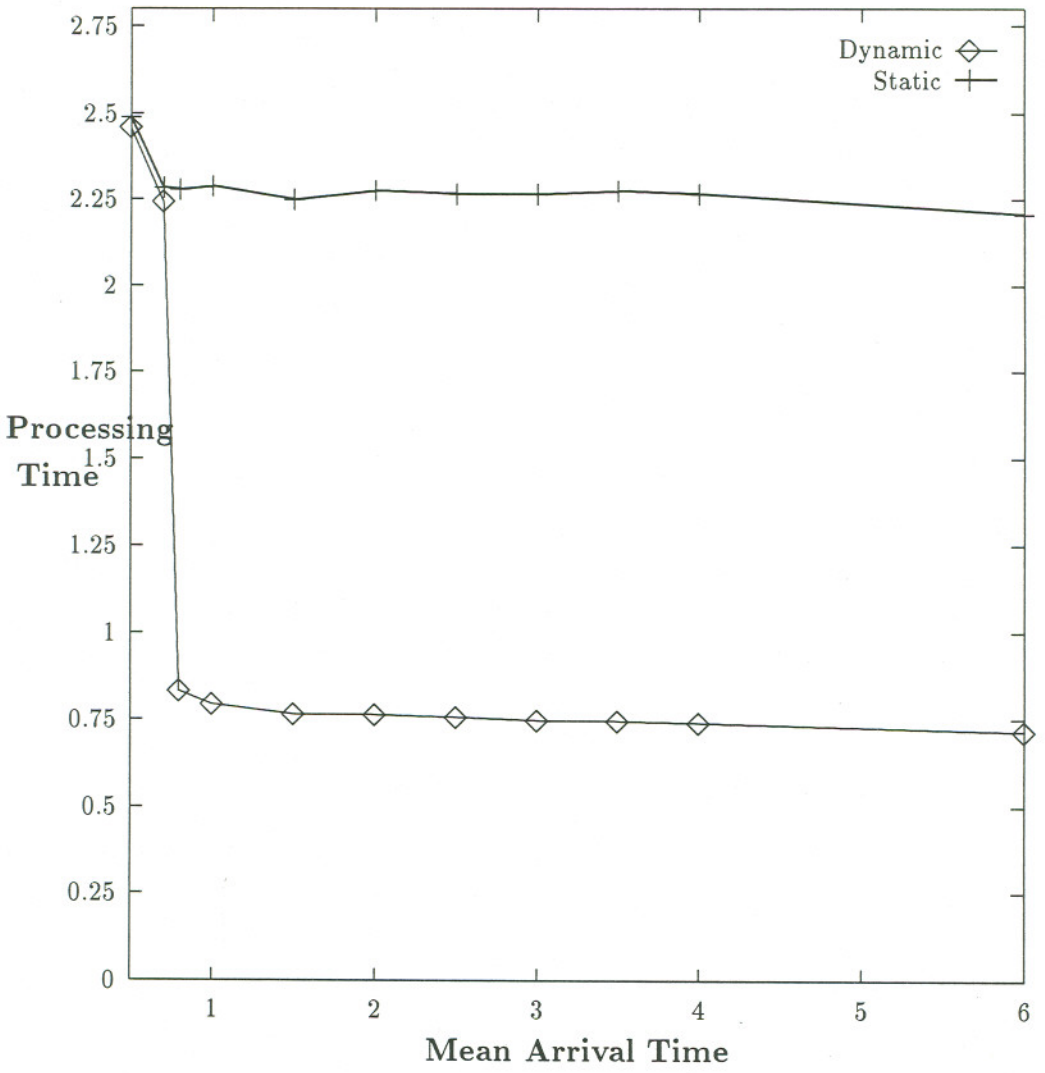

Fig. 5.3: Processing Time vs Mean Arrival Time
w = 1, n = 5, Z = 1, N = 5

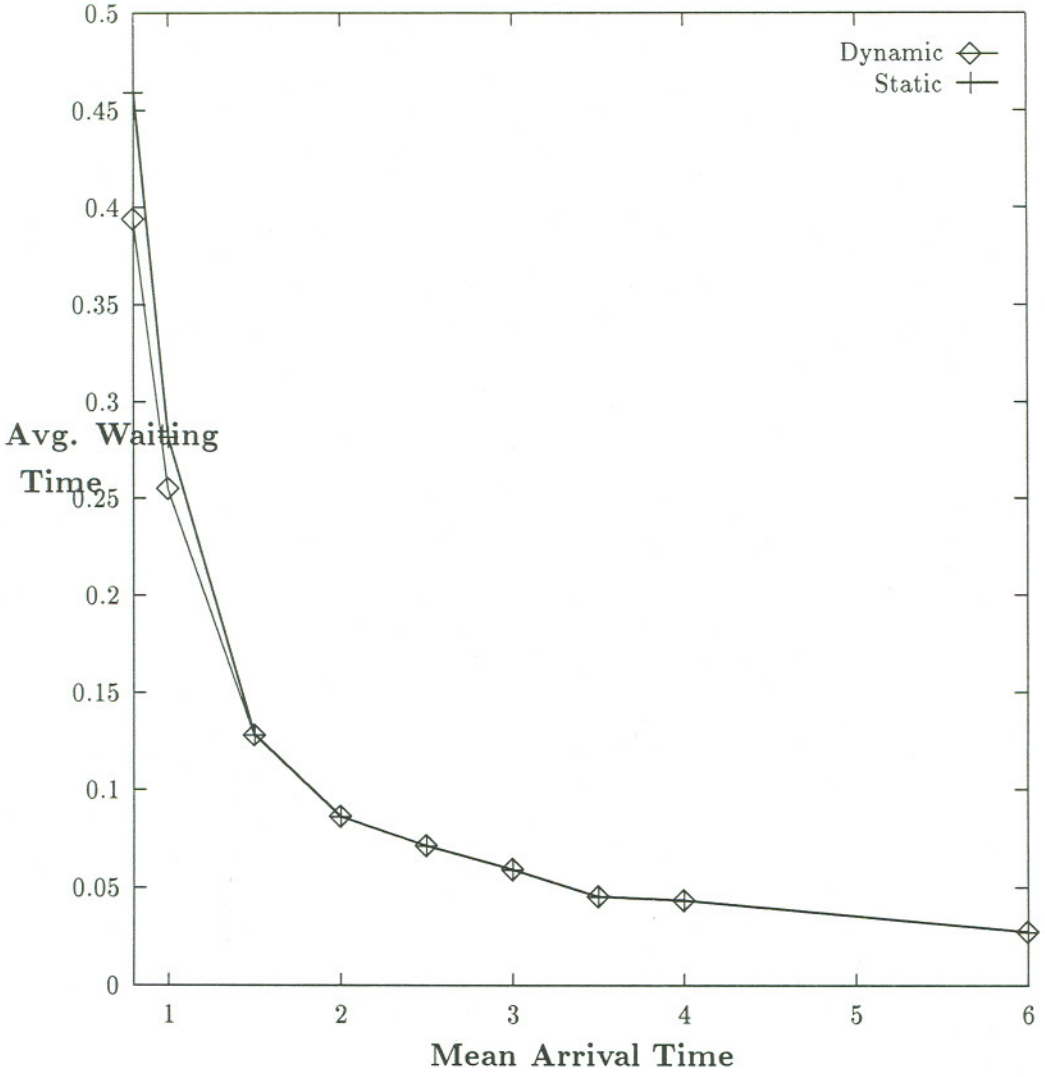

Fig. 5.4: Avg. Waiting Time vs Mean Arrival Time
w = 1, n = 5, Z = 1, N = 5

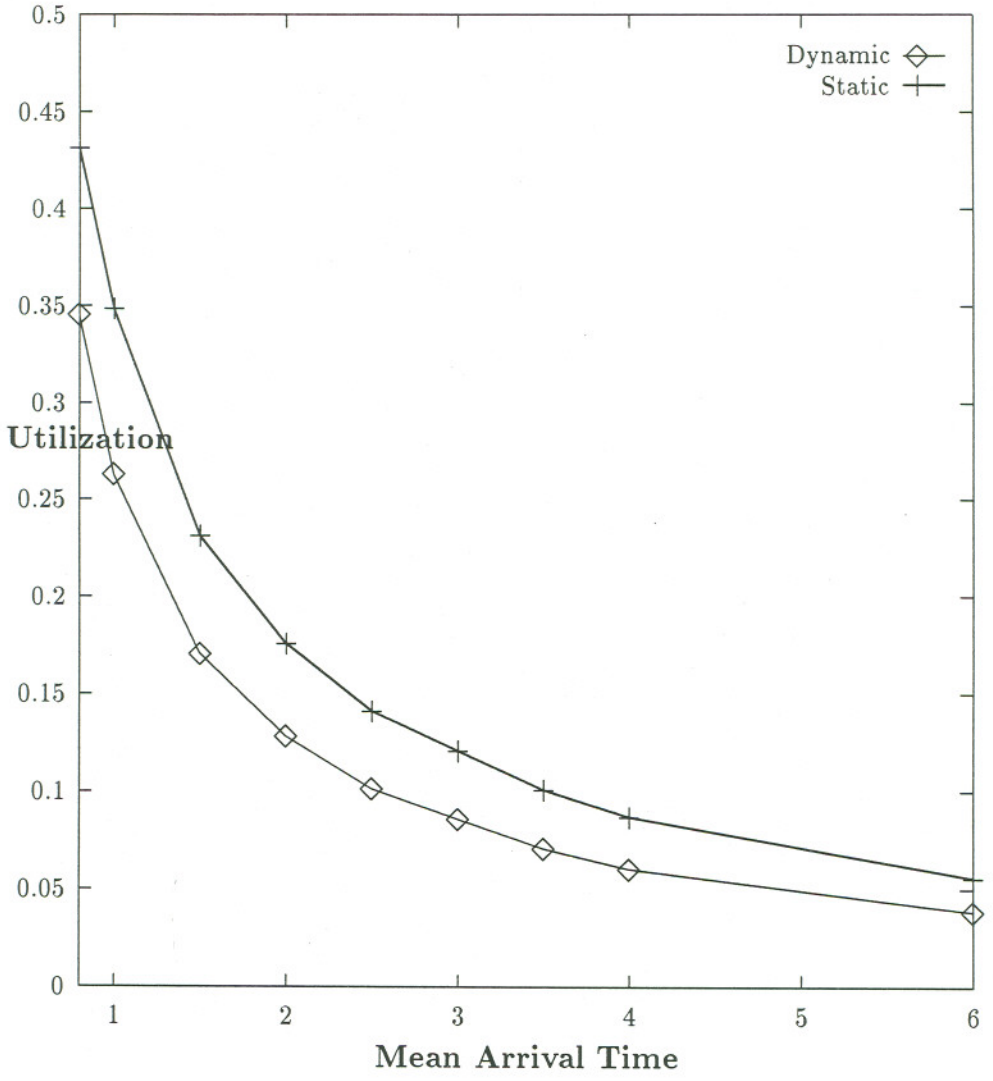

Fig. 5.5: Utilization vs Mean Arrival Time
$w = 1, n = 5, Z = 1, N = 5$

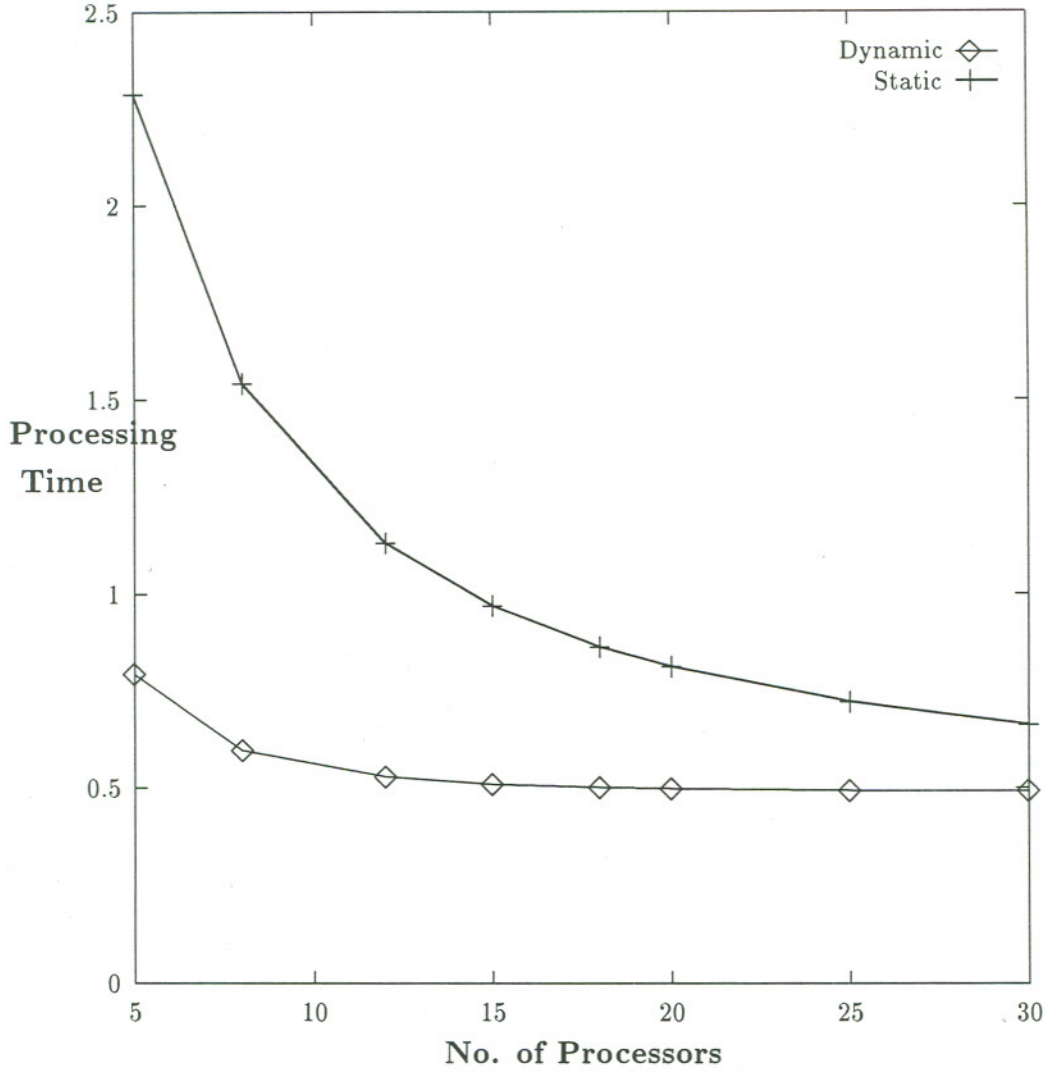

Fig. 5.6: Processing Time vs No. of Processors
$w = 1, n = 5, Z = 1$, mean arrival time = 1

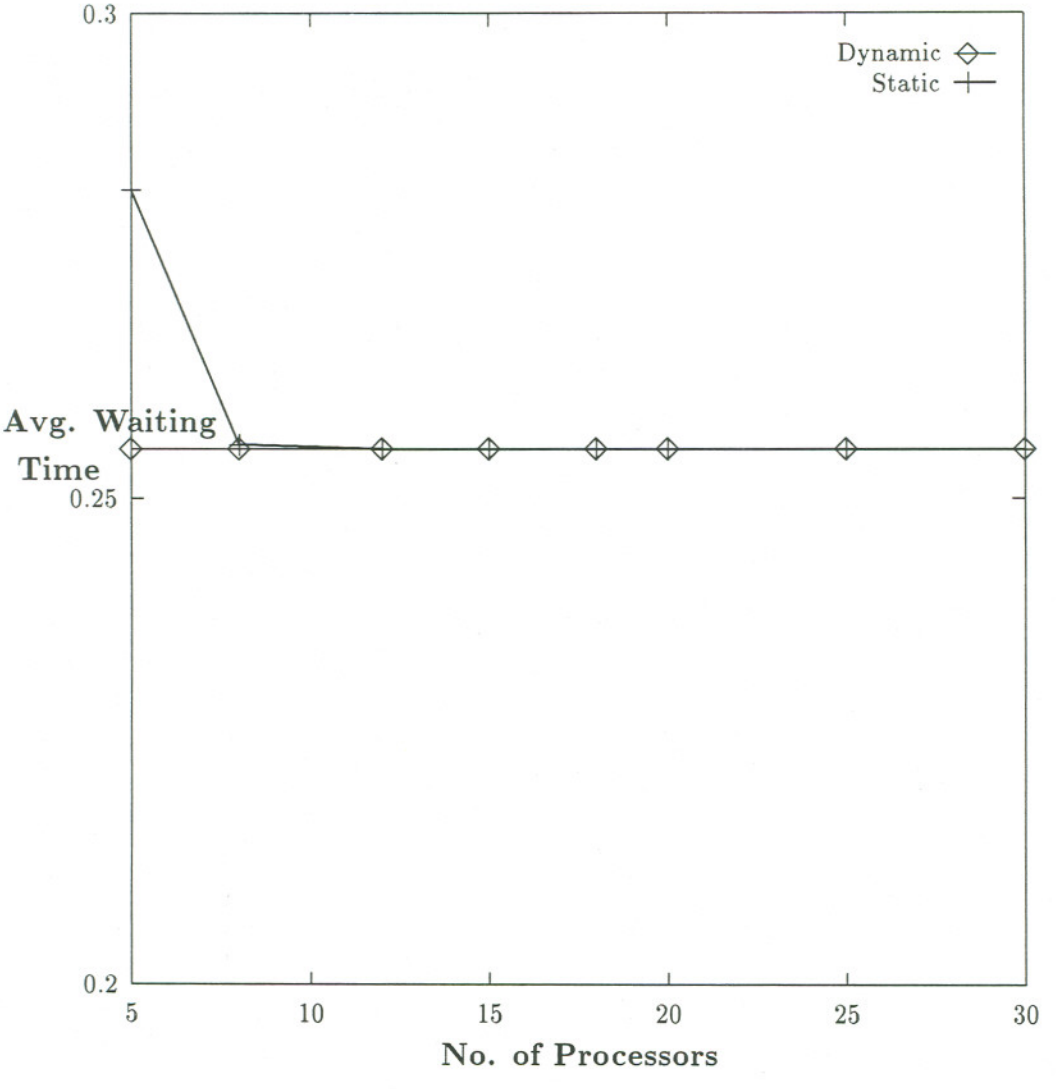

Fig. 5.7: Avg. Waiting Time vs No. of Processors
$w = 1, n = 5, Z = 1$, mean arrival time = 1

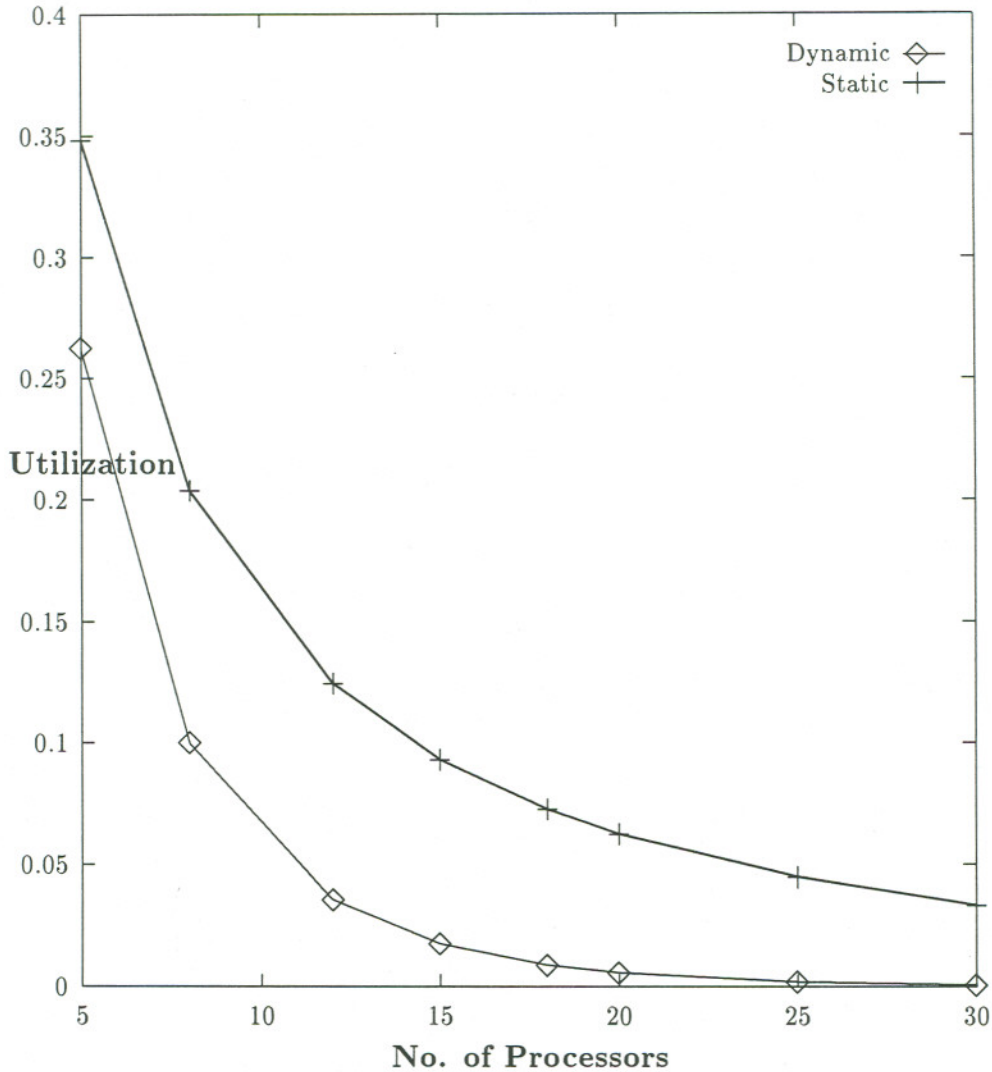

Fig. 5.8: Utilization vs No. of Processors
w = 1, n = 5, Z = 1, mean arrival mean = 1

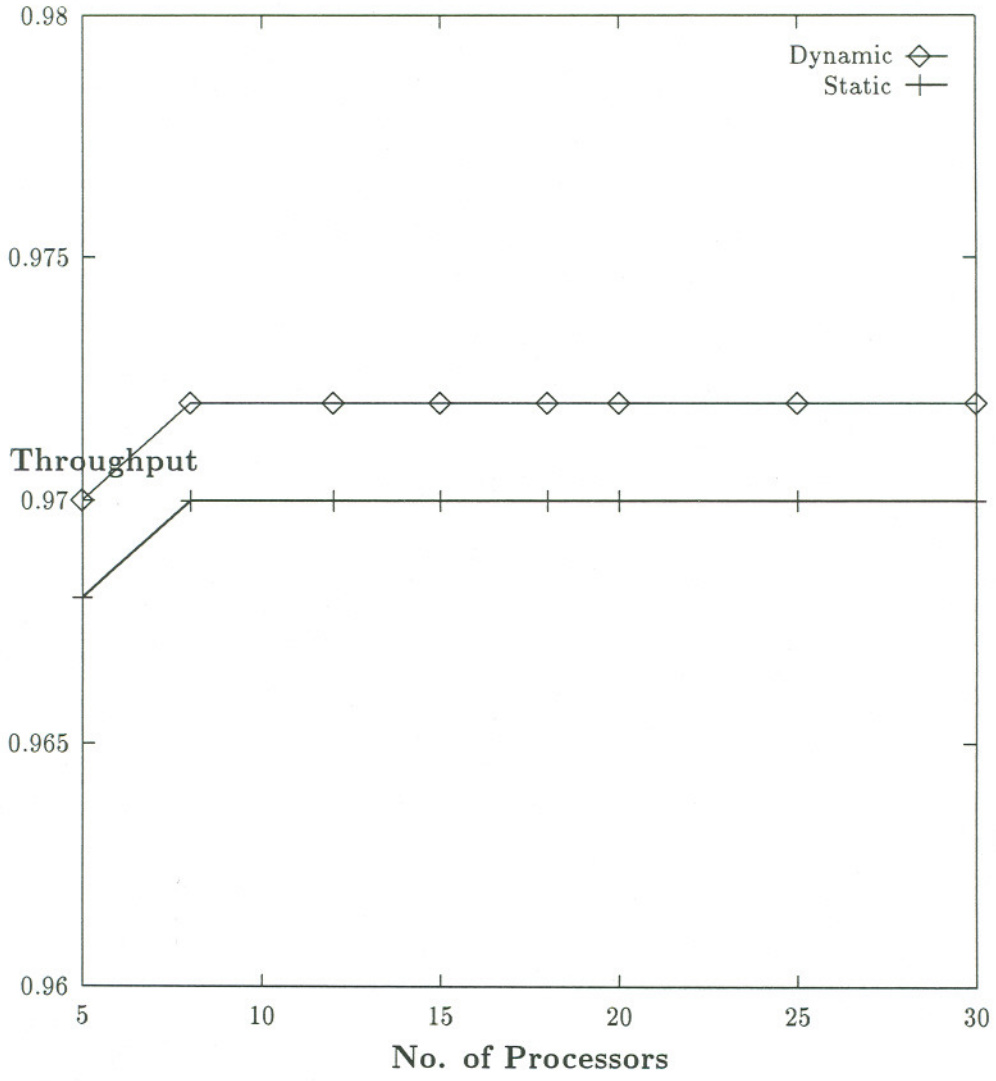

Fig. 5.9: Throughput vs No. of Processors
w = 1, n = 5, Z = 1, mean arrival time = 1

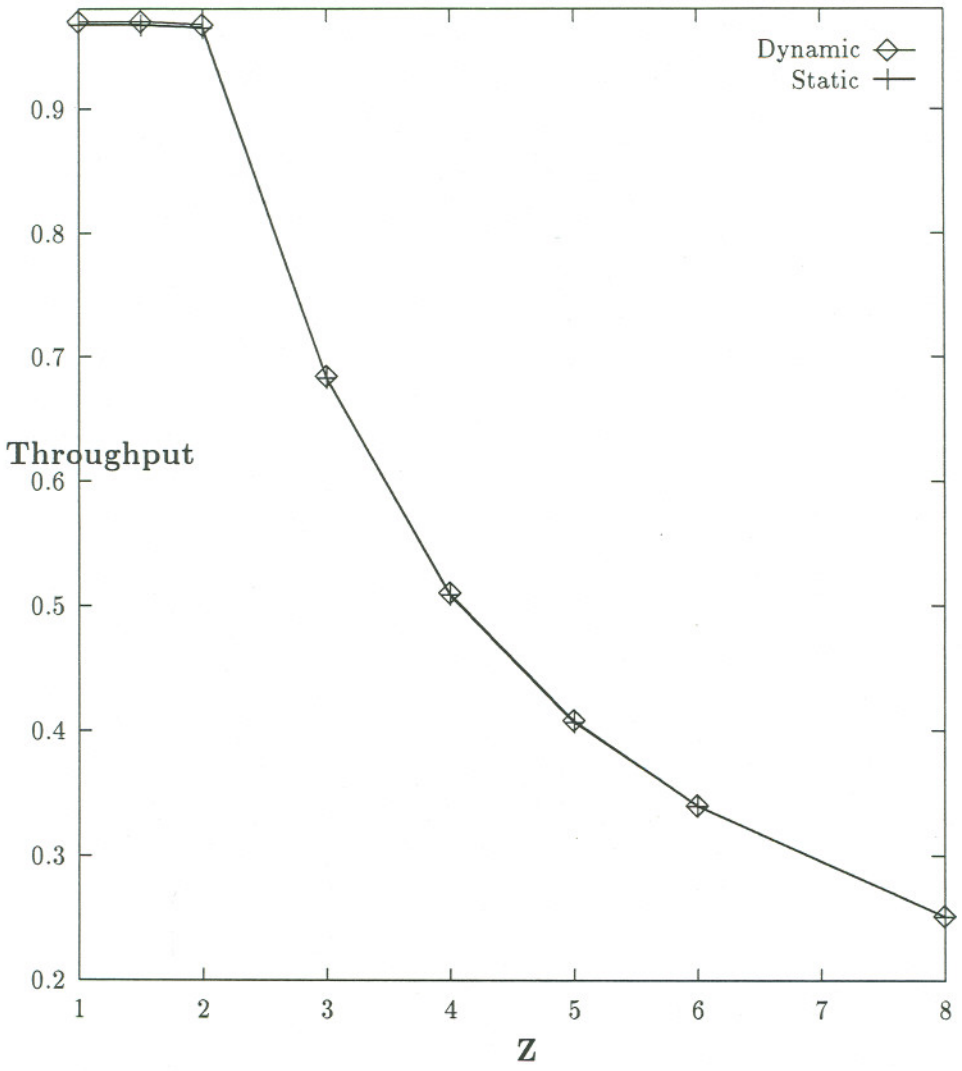

Fig. 5.10: Throughput vs The Bus Speed
w = 1, n = 5, N = 5, mean arrival time = 1