DESIGN OF EFFICIENT PARAMETER ESTIMATORS FOR DECENTRALIZED LOAD BALANCING POLICIES[1]

Spiridon Pulidas[2], Don Towsley[3], Jack Stankovic[4]
University of Massachusetts
Amherst, MA 01003

COINS Technical Report 87-79
August 20, 1987

Abstract

In this paper we study the problem of efficiently determining the optimum parameter values present in a decentralized threshold load balancing policy. The purpose is to improve the performance of a distributed system (e.g. minimize the average response time of a job) by using the processing power of the entire system. This is done by transfering jobs from heavily loaded nodes to lightly loaded nodes. A distributed optimization algorithm, where each host computes its own threshold value on-line is adapted to the selected load balancing policy. The algorithm requires that each host computer be able to estimate the change in throughput and expected queue length due to a change in either the threshold or the job arrival rate. Two efficient estimation techniques are proposed for estimating the required gradient information. These techniques are imbedded in the distributed threshold updating algorithm and simulation results are obtained for its behavior in a static as well as in a changing system environment.

[1]This work was supported by RADC under contract RI-448896X.
[2]Department of Electrical and Computer Engineering.
[3]Department of Computer and Information Science.
[4]Department of Computer and Information Science.

Table of Contents

1. INTRODUCTION

We consider distributed computer systems consisting of multiple host computers interconnected by a communication network. Jobs arrive at each host computer according to some arrival process and can be processed either locally or at other hosts after being transfered through the communication network. The results of jobs processed remotely are returned to the origin host computer. Communication delays are incurred due to the transfer of remote jobs and their results. In such an environment, *load balancing* attempts to improve the performance of the distributed system (e.g. to minimize the mean response time of a job) by using the processing power of the entire system to smooth-out periods of high congestion at individual nodes. This is done by transfering jobs from heavily loaded nodes to lightly loaded nodes.

Throughout this work we consider a class of *threshold* load balancing policies, that have been shown to be useful in several studies [1,2,3,4,5,6,7]. Specifically, we adopt a threshold policy studied by Eager, Lazowska and Zahorjan [5]. In this policy, jobs at each host are divided into two classes; local and remote jobs. *Local jobs* are those processed at the site of origination and *remote jobs* are those transfered for processing to another node. A job arriving to a node from the external world is processed locally only if the current queue length is less than a *threshold* parameter. Otherwise, the job is sent for remote processing to another host selected according to some probability distribution. Remote jobs are always accepted by the destination hosts and therefore jobs can move at most once. Both local and remote jobs are processed according to a first-come-first-served (FCFS) discipline at a given host.

Obviously, such a threshold policy has control parameters (e.g. threshold values and transfer probabilities for every host computer), that require fine-tuning in order to yield optimal or near-optimal performance. An efficient distributed algorithm for determining the optimum values for parameters present in the decentralized threshold scheduling policy has been proposed by Lee and Towsley [1]. The algorithm is iterative in nature and at each iteration the load balancing parameters at each host are updated. This algorithm requires that each host computer be able to estimate the change in throughput and expected queue length due to a change in either the threshold or the job arrival rate. The host computers

exchange this gradient information with each other and use these quantities to update the load balancing parameters for the next iteration of the algorithm.

Our approach towards estimating the gradients with respect to the threshold relies on the assumption that the arrival process is exponentially distributed. By adding a small perturbation to the observed parameter (i.e. a small decrease in the threshold), the estimator we have developed determines the effect of the change on the performance metric of interest (i.e. throughput, expected queue length), taking advantage of the *memoryless property* of the arrival process distribution. The estimator is originally presented for a system with Poisson arrivals and then is modified, so that it can be applied to a system with general distribution of arrival times but exponentially distributed service time. The arrival rate is estimated during the execution of the algorithm. The memory requirements necessary for the implementation of the estimation algorithm (i.e. number of counters required) are very low. The major advantage of this technique is its *on-line* potential, since it effectively attempts to provide performance sensitivity information while the actual system is running. The proposed estimation technique has been evaluated through simulations and the results were very accurate when compared to the analytically obtained results.

A different estimation method is proposed to obtain estimates with respect to the arrival rate. The method is based on a class of theorems derived from Likelihood Ratios and is extremely well suited to regenerative systems. A busy cycle of the processor has been used as the regeneration period. We have used the estimator in simulations with a very low increase in running time or memory requirements.

In Section 2, we describe the system model and the Load Balancing policy in detail. We consider the problem of determining optimum values for the parameters present in the load balancing policy and we adapt the distributed optimization algorithm developed by Lee and Towsley [1]. The necessary gradient information required by the above algorithm is also identified in this section. In Section 3, we present *Estimator-A*, which provides estimates of gradients with respect to the threshold. Simulation results from the application of the proposed estimator on various kinds of systems are also presented. The main example, considered in this section, is the application of Estimator-A on a processor modeled as a single-server queueing system with two classes of jobs (local and remote jobs), where

only one of these classes is governed by the threshold policy. In Section 4, *Estimator-B* is introduced in order to provide gradient information with respect to the arrival rate. The accuracy of the estimated gradients, as well as the convergence properties of the proposed estimator are evaluated through simulations. In Section 5, both Estimators A and B are imbedded in the decentralized threshold scheduling policy discussed in Section 2. Simulation results are presented for a system with five host computers modeled as single server queueing systems. It turns out that after a finite number of algorithm iterations, the behavior of the system, in a static environment is confined in a neighborhood of the optimal performance. Furthermore, a great improvement in the performance measure (average response time of a job in the system) has been observed in the system executing the distributed load balancing algorithm, compared to a system with no load balancing at all. Finally, we summarize our work in Section 6.

2. A DECENTRALIZED LOAD BALANCING SCHEME

2.1. System Model

The system considered in this work consists of N autonomous host computers interconnected by a communication network (Figure 2.1). Jobs arrive at each host from the external world according to some arrival process with rate λ_i, $i = 1, 2, 3, ..., N$. Jobs originating at host i can be processed either locally or at any other host $j \neq i$. For the sake of simplicity, it is assumed that jobs processed at each host have the same service time distribution, regardless of the origin host. The results of a job transfered for remote service are returned to the origin host computer. Communication delays are incurred during both transfers. Each host has a communication server that takes care of the job transfers between computers.

2.2. Load Balancing Policy

Jobs at each host computer are divided into two classes; namely *local* and *remote* jobs. Local jobs are those that are processed at the site of origination and remote jobs are those that have been transfered from other hosts for remote processing. Let L_i denote the total number of jobs at host computer i. The following threshold load balancing policy is considered:

- If a job arriving at host i from the external world finds that $L_i < T_i$, where T_i is a threshold parameter associated with host i, it is processed locally.
- If an external job arriving at host i finds $L_i \geq T_i$, then this job is transfered for remote service to a host $j \neq i$ with probability P_{ij}, where $\sum_{j \neq i} P_{ij} = 1$.
- Remote jobs arriving at host i from other hosts are always accepted.
- Jobs arriving at each host are processed according to a first-come-first-served (FCFS) basis.

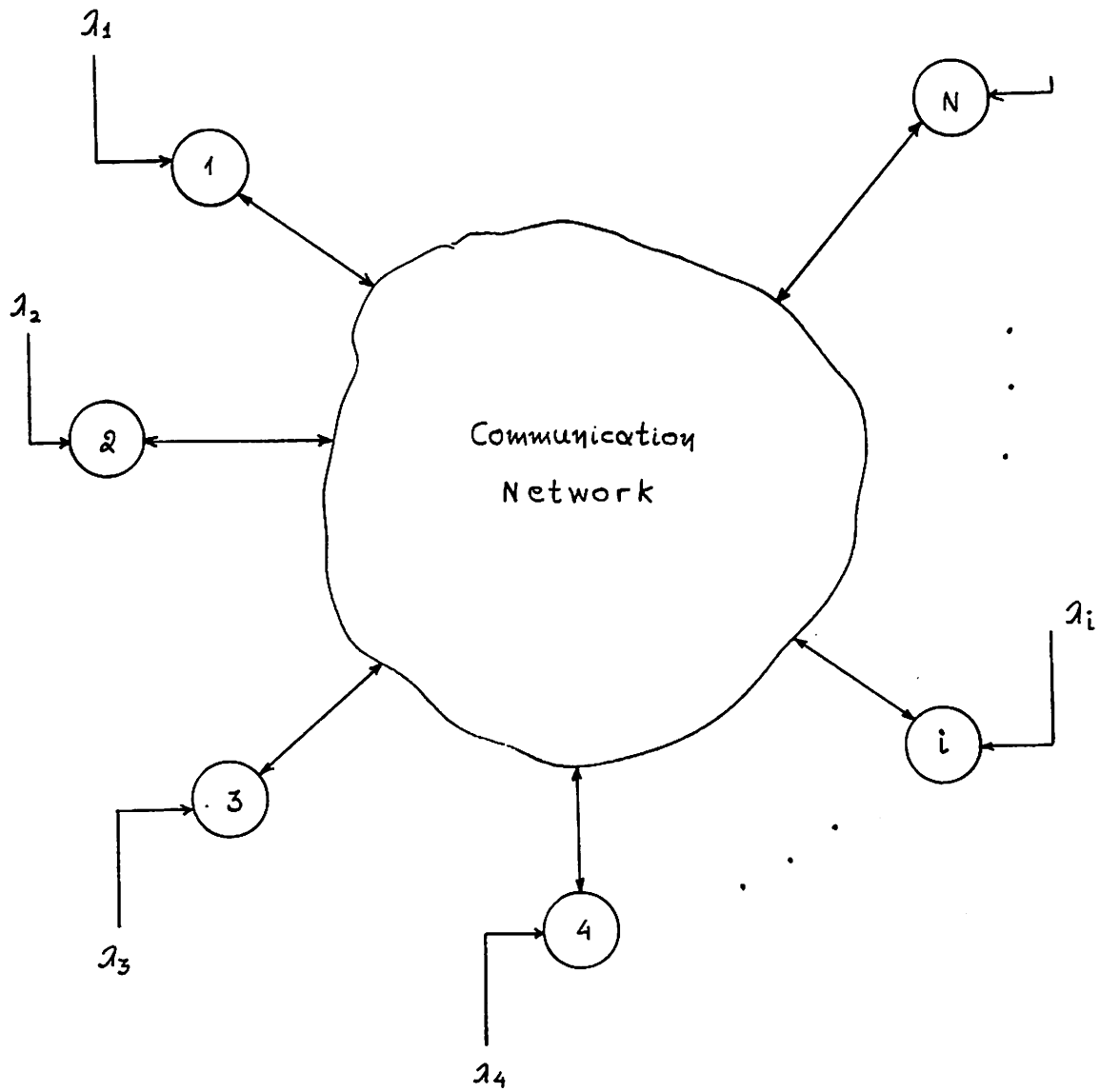


Fig. 2.1 - The System Model.

2.3. Optimal Load Balancing Problem

The threshold policy described in section 2.2 has control parameters (i.e. threshold value and transfer probabilities for each host computer) which require fine-tuning in a changing system environment. We select the mean response time of a job as the performance measure. Each host computer is modeled as a single-server queueing system with two classes of jobs (Figure 2.2). Let $\lambda_i^{(l)}$ and $\lambda_i^{(r)}$ be the throughput of local and remote jobs respectively at host i. Let also $f_i(\lambda_i^{(l)}, \lambda_i^{(r)})$ and $g(\Lambda^{(r)})$ denote the mean queue length at node i and the communication network respectively, where $\Lambda^{(r)} = \sum_{i=1}^{N} \lambda_i^{(r)}$. The mean response time $E[R]$ of a job in the system is given by the following formula [2]:

$$E[R] = \frac{\sum_{i=1}^{N} f(\lambda_i^{(l)}, \lambda_i^{(r)}) + g(\Lambda^{(r)})}{\Lambda} \tag{2.1}$$

where $\Lambda = \sum_{i=1}^{N} \lambda_i$. Under the described threshold policy the optimization problem can be stated as follows [1]:

MINIMIZE $\quad \Lambda E[R]$

with respect to

T_i's and P_{ij}'s,

under the constraints

T_i is non-negative integer, $\quad i = 1, 2, ..., N,$

$\sum_{j \neq i} P_{ij} = 1, \quad i = 1, 2, ..., N.$

This is an integer optimization problem with respect to integer and real variables. This kind of problem cannot be solved exactly and a heuristic algorithm is required for its solution. Such an approach is proposed in [1], where a distributed optimization algorithm is used.

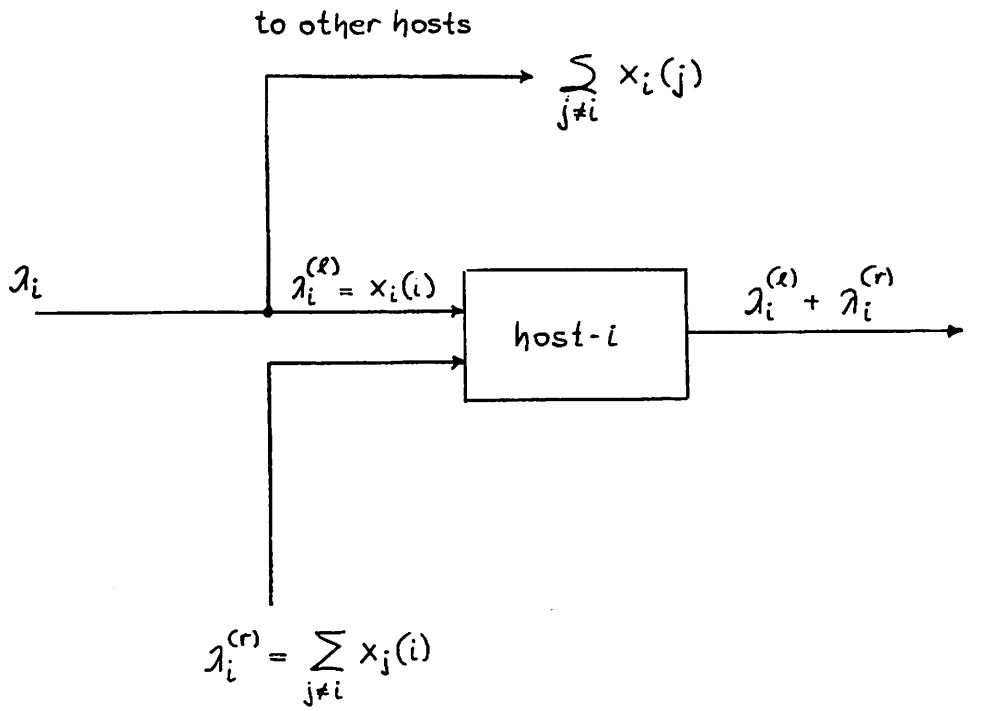


Fig. 2.2 - Host i as a single server queueing system with two classes of jobs.

By distributed algorithm we mean that each host computer performs a portion of the whole computation (it does not solve the entire optimization problem) and collaborates with each other to solve the problem by exchanging some information.

Let $x_i(j)$ denote the flow of jobs originating at host i and processed at host j. Obviously, $x_i(i) = \lambda_i^{(l)}$ and $\sum_{j \neq i} x_j(i) = \lambda_i^{(r)}$ (Figure 2.2). For given values of T_i's and P_{ij}'s, the steady state flows $x_i(j)$'s can be computed. However, it may not be possible to compute integer T_i's for arbitrary values of $x_i(j)$'s. We ignore this consideration and treat $x_i(j)$'s as non-negative real variables. The optimization problem can be reformulated as follows:

MINIMIZE $\quad \Lambda E[R]$

with respect to

$$x_i(j)\text{'s},$$

under the constraints

$$\sum_{j=1}^{N} x_i(j) = \lambda_i, \quad i = 1, 2, ..., N,$$

$$x_i(j) \geq 0, \quad i = 1, 2, ..., N.$$

Necessary conditions for the optimal solution of the above problem can be derived from the Kuhn-Tucker conditions [8] as follows. For all j's,

$$\frac{df_j}{dx_i(j)} + (1 - \delta_{ij}) \frac{dg}{dx_i(j)} \begin{cases} = C_i, & \text{for } x_i(j) > 0 \\ \geq C_i, & \text{for } x_i(j) = 0 \end{cases} \tag{2.2}$$

where C_i is some constant (Langrange multiplier), and δ_{ij} denotes the Kronecker delta function. This is true for each host computer i, $i = 1, 2, ..., N$. Note that $(1 - \delta_{ij})$ has been introduced since local jobs do not experience communication delay. The derivative $df_j/dx_i(j)$ represents the incremental delay incurred for jobs processed at host j due to the job flow from host i to host j. The derivative $dg/dx_i(j)$ represents the incremental delay incurred at the communication server for jobs originating at host i but transfered for remote service at host j. Relation (2.2) indicates that from host i's point of view the

incremental delay incurred for jobs processed at host j, due to the flow from host i to host j, should be equal for all j if there is a positive job flow from host i to host j. On the other hand, if there is no job flow from host i to host j, the incremental delay should be no less than the above value.

Using relation (2.2) Lee and Towsley developed a distributed algorithm for decentralized load balancing. In this algorithm each host compares its own incremental delay to the minimum incremental delay of the other hosts to determine whether to increase or decrease its threshold parameter. The algorithm is iterative in nature and the threshold and transfer probability parameters at each host are updated at each iteration. Initially, each host i sets T_i's and P_{ij}'s to some arbitrary values. At each iteration of the algorithm, host i executes the following.

Algorithm

- Host i computes the incremental delay information $df_i/dx_i(i)$ and $df_i/dx_j(i)$ for $j = 1, 2, ..., N$ and $j \neq i$. It reports the incremental delay information, due to jobs originating at a host $j \neq i$ but transfered for remote service at host i, to every host j, $j = 1, 2, ..., N$.

- Using the incremental delay information

$$df_j/dx_i(j) + dg/dx_i(j) \tag{2.3}$$

reported by every host $j \neq i$, host i computes the quantity

$$A(i) = \min_{j, j \neq i} \{df_j/dx_i(j) + dg/dx_i(j)\}. \tag{2.4}$$

- Host i compares its own incremental delay to the minimum incremental delay reported by the other hosts:

$$\begin{aligned} \text{If } df_i/dx_i(i) > A(i) + \theta \quad &\Longrightarrow \quad T_i := T_i - 1, \\ \text{If } df_i/dx_i(i) < A(i) - \theta \quad &\Longrightarrow \quad T_i := T_i + 1, \\ \text{else} \quad T_i &:= T_i, \end{aligned}$$

where θ is a non-negative constant, which must be tuned to prevent threshold change due to a slight imbalance to the incremental delays.

- Update P_{ij}'s using the following formula:

$$P_{ij} = \frac{\mu_j - x_j(j)}{\sum_{k,k\neq i}\{\mu_k - x_k(k)\}} \tag{2.5}$$

where μ_j denotes the maximum processing rate of host j.

In order to implement the above algorithm, the incremental delay information at each host must be computed. We shall assume that the delay incurred due to the transfer of jobs through the communication network is an exponential random variable with parameter μ_c. Therefore, the incremental delay due to the job transfer can be approximated by the mean communication delay μ_c. Recall that each host computer has a communication server that takes care of the job transfers between computers. Consequently, there are two categories of incremental delays; namely, those which are due to local job flow (i.e. $df_i/dx_i(i)$) and those which are due to remote job flow (i.e. $df_j/dx_i(j)$). Incremental delays of the former category are affected by the integer threshold constraints, since local job flow is directly controlled by the threshold parameter. On the other hand, we assume that incremental delays of the latter category are not affected by the integer threshold constraints.

The incremental delay due to local job flow corresponds to the derivative of the mean queue length with respect to the throughput of local jobs (since $x_i(i) = \lambda_i^{(l)}$, Figure 2.2). The following backward difference formula can be used to approximate the incremental delay due to local job flow:

$$\frac{df_i}{dx_i(i)} = \frac{dE[L_i]}{d\lambda_i^{(l)}} \approx \frac{E[L_i]_{T_i} - E[L_i]_{T_i-1}}{[\lambda_i^{(l)}]_{T_i} - [\lambda_i^{(l)}]_{T_i-1}} \tag{2.6}$$

where $E[L_i]_{T_i}$ and $[\lambda_i^{(l)}]_{T_i}$ denote the mean queue length and the local job throughput at host i when the threshold is T_i (Figure 2.2). The purpose of *Estimator-A*, introduced in

the next section, is to provide on-line estimates of $E[L_i]_{T_i-1}$ and $[\lambda_i^{(l)}]_{T_i-1}$, given that the system is operating at threshold T_i. The estimates are based on real data gathered from the actual system during an observation interval.

The incremental delay due to remote job flow corresponds to the derivative of the mean queue length at host j with respect to the arrival rate of remote jobs $\lambda_j^{(r)}$ as can be easily seen from relation (2.2) and Figure 2.2. Hence,

$$\frac{df_j}{dx_i(j)} = \frac{dE[L_j]}{d\lambda_j^{(r)}} \qquad (2.7).$$

Estimator-B, described in Section 4, provides on-line estimates of the above gradient information, based on a class of theorems derived from Likelihood Ratios.

3. GRADIENTS WITH RESPECT TO THRESHOLD

3.1. Introduction

In this section we introduce an estimator, whose the purpose is to provide estimates of gradients with respect to the threshold in systems involved in the decentralized threshold scheduling policy described in section 2. The estimator we developed determines the effect of a change in the threshold parameter on the performance metric of interest (i.e. throughput and expected queue length). The assumption is made that either the arrival process or the service time are exponentially distributed. The algorithm uses the *memoryless property* of the exponential distribution in order to efficiently estimate the desired gradients. The major advantage of the proposed estimator is the fact that effectively provides performance sensitivity information while the actual system is running (on-line estimation).

A relatively new approach towards estimating gradients with respect to continuous-valued parameters is referred to as *Perturbation Analysis.* The basis of Perturbation Analysis methodology is extensively described in [9,10]. An extension of this technique to systems where estimation of gradients with respect to integer-valued parameters is desirable, has been attempted in [11], where a Perturbation Analysis algorithm is presented, which provides sensitivity information with respect to a threshold parameter. However, the state memory required by the algorithm in [11] grows to infinity. On the other hand, the estimator introduced in this section requires only four counters in order to provide on-line estimation of gradients with respect to a threshold parameter.

3.2. Gradient Estimation in M/G Systems

We assume that each host computer of the distributed system described in 2.1. can be modeled as a two class $M/G/1/\mathbf{T}$ system, where $\mathbf{T} = (T, \infty)$ since jobs belonging to the second class (remote jobs) are always accepted. Only jobs of the first class (local jobs) are governed by the threshold parameter T. For the sake of simplicity in exposition, we first consider a single class $M/G/1/T$ system with Poisson arrival rate λ and finite queue

capacity T, and then we show how the estimator can be very easily extended to a two class system. Let L be the queue length and $TPUT$ the throughput of the system. The physical system will be referred to as the *nominal system.* What we actually need is to estimate the derivative $dE[L]/dTPUT$ while the nominal system is running, using the backward difference formula:

$$\frac{dE[L]}{dTPUT} \approx \frac{E[L]_T - E[L]_{T-1}}{[TPUT]_T - [TPUT]_{T-1}}. \tag{3.1}$$

We shall refer to the system with threshold value $T-1$ as the *perturbed system.* The nominal system observes itself and after an observation interval estimates the average queue length and the throughput as if it had a threshold value equal to $T-1$.

Fig. 3.1a illustrates the portion of a nominal sample path for a single class $M/G/1/T$ system, with $T=3$. Let a_i be the time of the $i-th$ arrival and d_j be the time of the $j-th$ departure. Figure 3.1b represents the corresponding portion of the perturbed system with threshold parameter $T=2$. We assume that the service time s_k of the $k-th$ customer ($k=1,2,...$) depends only on k. Arrivals a_1 and a_2 are accepted by both the nominal and the perturbed system. However, arrival a_3 is accepted by the nominal system while it is rejected by the perturbed one. In general, every time the nominal system reaches its threshold T the corresponding arrival is shipped out by the perturbed system with threshold value $T-1$. Obviously, every arrival rejected by the nominal system is also rejected by the perturbed one. As soon as an arrival accepted by the nominal system is rejected by the perturbed one, the latter system has one less customer than the former one. The two systems will continue to differ by one customer until an idle period appears in the nominal path. This is the case with the idle period just before the arrival a_5 in Figures 3.1a and 3.1b.

The first two departures in both the nominal and the perturbed paths of Figure 3.1a and 3.1b occur at times d_1 and d_2 ($d_1 = d'_1$, $d_2 = d'_2$). Departure d_2 leaves the nominal system with just one customer. Since the perturbed system has one less customer than the nominal one (due to the rejection of arrival a_3), a new *idle period* will appear in the perturbed path. This idle period is terminated by arrival a_4.

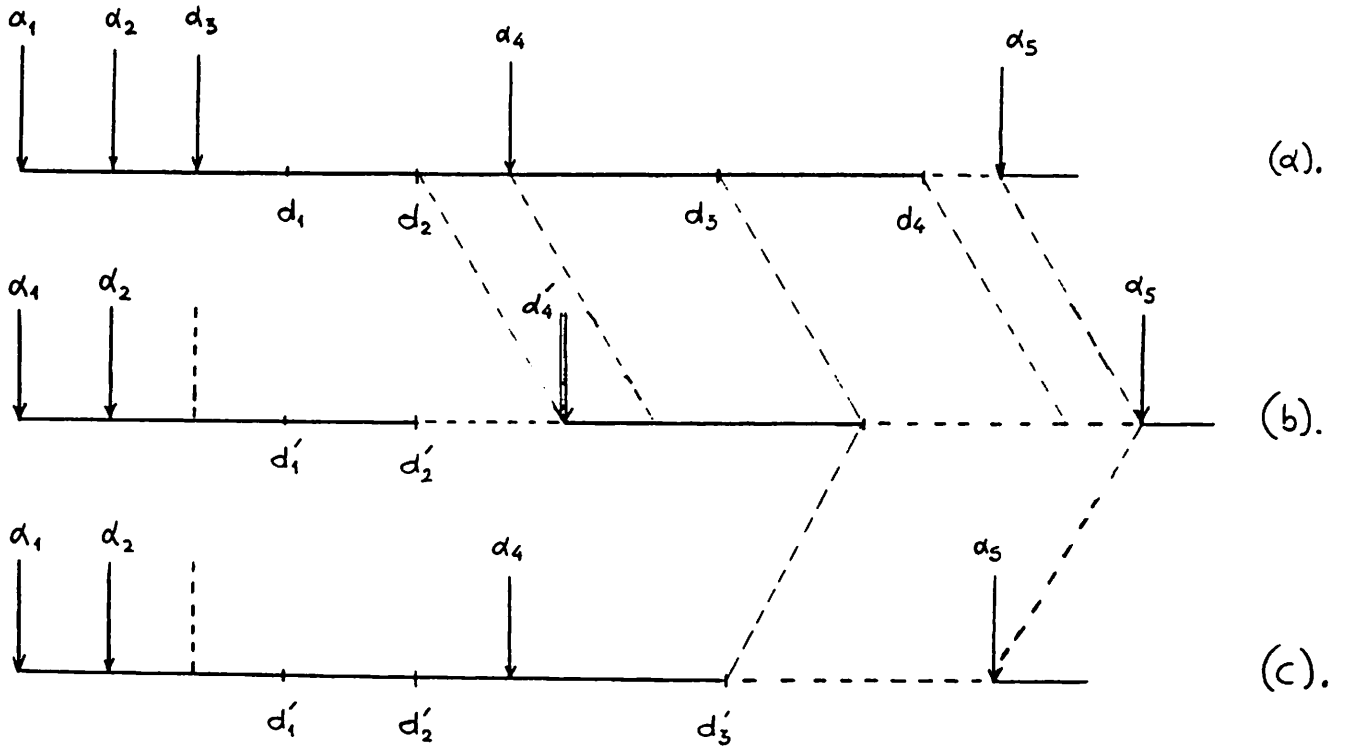


Fig. 3.1 - Estimation of gradients with respect to the threshold.

Therefore, the major effect of the perturbation $\delta T = -1$ on the perturbed path is the generation of new idle periods, due to the rejection of arrivals which are accepted by the nominal system. Since all the jobs rejected by the nominal system are also rejected by the perturbed one, it is not possible that an idle period that appeared in the nominal path be eliminated in the perturbed path.

The nominal system observes itself and can trace the case where the nominal and perturbed paths are out of phase (i.e. their queue lengths differ by one). This happens every time an arrival accepted by the nominal system is rejected by the perturbed one. When a departure leaves the nominal system with only one job and the two systems are out of phase, then the nominal system knows that the perturbed one starts an idle period waiting for the next customer. Instead of waiting for that next arrival, we generate an *idle period* terminated by a *ficticious customer* and we assign to that customer the currently available service time. This new idle period is derived from an exponential distribution with parameter λ; this is permissible because of the *memoryless property* of the Poisson arrival process. All the subsequent events in the perturbed path are shifted in time by the introduced idle period. The above situation is illustrated in Figure 3.1c, where a'_4 is the ficticious arrival terminating the introduced idle period.

Actually, although the sample path shows a specific idle period length, it is not required by the algorithm performing the estimation. We only need to keep a counter of how many idle periods have been introduced in the perturbed system during the observation interval. During that interval the nominal system can estimate the rate λ_{est} of its arrival process. At the end of the observation interval, it can use this value to estimate the *total idle time* introduced in the perturbed system, using the formula:

$$idle\ time = \frac{(\#\ idle\ periods)}{\lambda_{est}}. \tag{3.2}$$

In the following we describe the proposed *Estimator-A* more formally. We make the assumptions that the arrival process is a Poisson process and that the nominal system monitors its average queue length $E[L]$ and its throughput $TPUT$. The following four counters are required by the algorithm:

- *PHASE*: It indicates whether the nominal and the perturbed system are out of phase. It is 0 when both systems have the same number of customers; otherwise it is 1.
- *PLAST*: It indicates the last time the $PHASE$ changed from 0 to 1.
- *TLESS*: Total time during the observation interval that the perturbed system has one less customer than the nominal one.
- *IDLE*: Total number of idle periods introduced in the perturbed path during the observation interval.

Estimator-A

- Initially the counters $PHASE$, $PLAST$, $TLESS$ and $IDLE$ are set to 0.
- At the $i-th$ arrival the following instructions are executed:

```
If ( L(a_i) = T ) and ( PHASE = 0 ) then
  begin
    PHASE := 1;
    PLAST := a_i;
  end
```

where a_i denotes the time of the $i-th$ arrival and $L(a_i)$ the number of customers right after the $i-th$ arrival.

- At the $j-th$ departure the following instructions are executed:

```
If ( L(d_j) = 1 ) and ( PHASE = 1 ) then
  begin
    IDLE := IDLE + 1;
    PHASE := 0;
    TLESS := TLESS + d_j - PLAST;
  end
```

where d_j denotes the time of the $j-th$ departure and $L(d_j)$ the number of customers right after the $j-th$ departure.

The nominal system observes itself for an observation interval τ. At the end of that interval knowing the statistics $E[L]_T$ and $[TPUT]_T$, it is able to estimate the corresponding statistics for the perturbed system using the following formulas:

$$[TPUT]_{T-1} = \frac{\tau[TPUT]_T}{\tau + (IDLE/\lambda_{est})} \tag{3.3}$$

$$E[L]_{T-1} = E[L]_T - \frac{TLESS + (IDLE/\lambda_{est})}{\tau + (IDLE/\lambda_{est})} \tag{3.4}$$

where $IDLE/\lambda_{est}$ denotes the total idle time introduced in the perturbed path. The minimum length of the observation interval τ necessary to obtain a good estimate for $E[L]_{T-1}$ and $[TPUT]_{T-1}$ depends on the specific application of the system and can be determined empirically. As it turns out from the simulation results it depends on the utilization of the system. Section 3.3. contains simulation results to demonstrate the performance of the proposed estimator and discusses the convergence properties of the algorithm.

Estimator-A can be very easily used in the case of an $M/G/1/\mathbf{T}$ system with two classes of jobs (local and remote), where $\mathbf{T} = (T, \infty)$. Note that this is the model of the host computers, in the decentralized threshold scheduling policy described in Section 2. Remote jobs are always accepted by both the nominal and the perturbed system. Therefore, only local jobs can be accepted by the nominal system and rejected by the perturbed one, as a result of the change in the threshold value. Let λ_{est} be the arrival rate of jobs coming in from the external world and $\lambda_{est}^{(r)}$ be the arrival rate of remote jobs, estimated during the observation interval. Then the *idle time* introduced in the perturbed system is given by the formula:

$$idle\ time = \frac{IDLE}{\lambda_{est} + \lambda_{est}^{(r)}}. \tag{3.5}$$

The rest of the algorithm works exactly in the same way as in the single class $M/G/1/T$ system.

3.3. Simulation Results

In this section, we present simulation results to demonstrate the performance of *Estimator-A*. As an example, Tables 3.1 through 3.6 contain simulation results for a single class $M/M/1/3$ system with service rate $\mu = 1.0$ *jobs/sec* and different values of arrival rate: $\lambda = 0.2, 0.5$ and 0.8 *jobs/sec*. *Estimator-A* is used to provide on-line estimates of the average queue length $E[L]$ and throughput $TPUT$ of the perturbed system with threshold $T = 2$. The estimated values are compared to the actual ones which are computed analytically. In order to point out the convergence properties of the estimation technique, the simulation results are presented for different observation intervals.

Tables 3.7 and 3.8 contain simulation results for an $M/H_2/1/3$ system with one class of jobs, hyperexponential service time with parameters $\mu = 1.0$ and $C_v = 2$ and Poisson arrivals with parameter $\lambda = 0.5$ *jobs/sec*. The "actual" values of $E[L]_{T=2}$ and $[TPUT]_{T=2}$ are obtained by simulating the corresponding system with threshold $T = 2$.

#completions	$[TPUT]^{act}_{T=2}$	$[TPUT]^{est}_{T=2}$	*%error*	*#runs*	σ_x
50	0.426	0.450	5.33	10	0.064
100	0.426	0.440	3.28	10	0.042
200	0.426	0.438	2.81	10	0.035
500	0.426	0.430	0.94	10	0.015
1000	0.426	0.426	0.00	10	0.007
2000	0.426	0.426	0.00	10	0.007
5000	0.426	0.426	0.00	10	0.003
10000	0.426	0.426	0.00	10	0.003

Table 3.1 - Estimation of $[TPUT]_{T=2}$ for an $M/M/1/3$ system with $\lambda/\mu = 0.2$.

#completions	$E[L]^{act}_{T=2}$	$E[L]^{est}_{T=2}$	*%error*	*#runs*	$\sigma_{\bar{x}}$
50	0.571	0.599	4.90	10	0.114
100	0.571	0.590	3.33	10	0.093
200	0.571	0.553	3.15	10	0.069
500	0.571	0.554	2.97	10	0.039
1000	0.571	0.560	1.92	10	0.021
2000	0.571	0.564	1.22	10	0.015
5000	0.571	0.565	1.05	10	0.006
10000	0.571	0.567	0.70	10	0.005

Table 3.2 - Estimation of $E[L]_{T=2}$ for an $M/M/1/3$ system with $\lambda/\mu = 0.2$.

#completions	$[TPUT]^{act}_{T=2}$	$[TPUT]^{est}_{T=2}$	*%error*	*#runs*	$\sigma_{\bar{x}}$
50	0.194	0.212	9.27	10	0.042
100	0.194	0.201	3.61	10	0.027
200	0.194	0.199	2.57	10	0.019
500	0.194	0.195	0.52	10	0.006
1000	0.194	0.194	0.00	10	0.004
2000	0.194	0.194	0.00	10	0.003
5000	0.194	0.194	0.00	10	0.002
10000	0.194	0.194	0.00	10	0.002

Table 3.3 - Estimation of $[TPUT]_{T=2}$ for an $M/M/1/3$ system with $\lambda/\mu = 0.5$.

#completions	$E[L]^{act}_{T=2}$	$E[L]^{est}_{T=2}$	*%error*	*#runs*	σ_x
50	0.225	0.239	6.22	10	0.065
100	0.225	0.220	2.22	10	0.044
200	0.225	0.217	3.55	10	0.029
500	0.225	0.219	2.66	10	0.014
1000	0.225	0.220	2.22	10	0.010
2000	0.225	0.221	1.77	10	0.005
5000	0.225	0.222	1.33	10	0.003
10000	0.225	0.222	1.33	10	0.003

Table 3.4 - Estimation of $E[L]_{T=2}$ for an $M/M/1/3$ system with $\lambda/\mu = 0.5$.

#completions	$[TPUT]^{act}_{T=2}$	$[TPUT]^{est}_{T=2}$	*%error*	*#runs*	σ_x
50	0.590	0.644	9.28	10	0.091
100	0.590	0.602	2.12	10	0.055
200	0.590	0.600	1.70	10	0.046
500	0.590	0.595	0.84	10	0.021
1000	0.590	0.591	0.17	10	0.009
2000	0.590	0.590	0.00	10	0.006
5000	0.590	0.590	0.00	10	0.006
10000	0.590	0.590	0.00	10	0.003

Table 3.5 - Estimation of $[TPUT]_{T=2}$ for an $M/M/1/3$ system with $\lambda/\mu = 0.8$.

#completions	$E[L]^{act}_{T=2}$	$E[L]^{est}_{T=2}$	*%error*	*#runs*	σ_x
50	0.855	0.943	10.29	10	0.147
100	0.855	0.901	5.38	10	0.069
200	0.855	0.880	2.92	10	0.055
500	0.855	0.874	2.22	10	0.035
1000	0.855	0.868	1.52	10	0.020
2000	0.855	0.865	1.17	10	0.016
5000	0.855	0.862	0.82	10	0.011
10000	0.855	0.860	0.58	10	0.008

Table 3.6 - Estimation of $E[L]_{T=2}$ for an $M/M/1/3$ system with $\lambda/\mu = 0.8$.

#completions	$[TPUT]^{act}_{T=2}$	$[TPUT]^{est}_{T=2}$	*%error*	*#runs*	σ_x
50	0.406	0.460	13.30	10	0.114
100	0.406	0.413	1.72	10	0.058
200	0.406	0.417	2.70	10	0.032
500	0.406	0.412	1.47	10	0.020
1000	0.406	0.404	0.49	10	0.012
2000	0.406	0.406	0.00	10	0.007
5000	0.406	0.407	0.24	10	0.007
10000	0.406	0.406	0.00	10	0.004

Table 3.7 - Estimation of $[TPUT]_{T=2}$ for an $M/H_2/1/3$ system with $\lambda/\mu = 0.5$, $C_v = 2$.

#completions	$E[L]^{act}_{T=2}$	$E[L]^{est}_{T=2}$	*%error*	*#runs*	σ_x
50	0.593	0.605	2.02	10	0.155
100	0.593	0.592	0.17	10	0.067
200	0.593	0.602	1.52	10	0.056
500	0.593	0.584	1.54	10	0.054
1000	0.593	0.580	2.19	10	0.044
2000	0.593	0.584	1.54	10	0.025
5000	0.593	0.584	1.54	10	0.023
10000	0.593	0.588	0.84	10	0.015

Table 3.8 - Estimation of $E[L]_{T=2}$ for an $M/H_2/1/3$ system with $\lambda/\mu = 0.5$, $C_v = 2$.

Tables 3.9 and 3.10 contain simulation results for an $M/M/1/(3,\infty)$ system with two classes of jobs, where only jobs belonging to the first class are governed by the threshold parameter. Note that this is the model of each of the host computers involved in the decentralized scheduling policy described in Section 2. In our experiment the arrival rate of the first class (jobs coming in from the external world) is $\lambda = 0.5$ *jobs/sec*, the arrival rate of the second class (remote jobs) is $\lambda^{(r)} = 0.25$ *jobs/sec* and the service rate is $\mu = 1.0$ *jobs/sec*. The estimated parameters are the local throughput $[TPUT^{(l)}]_{T=2}$ and the average queue length $E[L]_{T=2}$ of the perturbed system.

#completions	$[TPUT^{(l)}]^{act}_{T=2}$	$[TPUT^{(l)}]^{est}_{T=2}$	*%error*	*#runs*	σ_x
100	0.350	0.365	4.28	10	0.167
500	0.350	0.365	4.28	10	0.017
1000	0.350	0.365	4.28	10	0.010
5000	0.350	0.360	2.86	10	0.004
10000	0.350	0.356	1.71	10	0.004

Table 3.9 - Estimation of $[TPUT^{(l)}]_{T=2}$ for a 2-class $M/M/1/(3,\infty)$ system.

$\#completions$	$E[L]^{act}_{T=2}$	$E[L]^{est}_{T=2}$	$\%error$	$\#runs$	σ_x
100	1.000	0.670	33.00	10	0.121
500	1.000	1.055	5.50	10	0.110
1000	1.000	1.041	4.10	10	0.041
5000	1.000	1.016	1.60	10	0.023
10000	1.000	1.015	1.50	10	0.019

Table 3.10 - Estimation of $E[L]_{T=2}$ for a 2-class $M/M/1/(3,\infty)$ system.

3.4. Gradient Estimation in G/M Systems

Estimator-A can be used to provide gradient estimates for G/M systems. In this case we take advantage of the *memoryless property* of the service time distribution. The nominal system observes itself for an observation interval τ and during that interval it counts the number of jobs rejected by the perturbed system, as a result of the perturbation $\delta T = -1$ in the threshold parameter. We assume also that the nominal system can monitor its throughput and average queue length. Four counters are required for the implementation of the algorithm; namely $PHASE$, $TLESS$, $PLAST$ and $NREJ$. The first three counters have exactly the same meaning as in the M/G case, while the fourth denotes the number of jobs rejected by the perturbed system but accepted by the nominal one. The algorithm can be described as follows:

- Initially the counters $PHASE$, $PLAST$, $TLESS$ and $NREJ$ are set to 0.
- At the $i-th$ arrival the following instructions are executed:

If ($L(a_i) = T$) and ($PHASE = 0$) then do
begin
$PHASE := 1$;
$PLAST := a_i$;

$NREJ := NREJ + 1;$

end

where a_i denotes the time of the $i-th$ arrival and $L(a_i)$ the number of customers right after the $i-th$ arrival.

- At the $j-th$ departure the following instructions are executed:

If ($L(d_j) = 1$) and ($PHASE = 1$) then do

begin

$TLESS := TLESS + d_j - PLAST;$

$PHASE := 0;$

end

where d_j denotes the time of the $j-th$ departure and $L(d_j)$ the number of customers right after the $j-th$ departure.

We can compute the estimated quantities by using formulas analogous to those of the M/G case. For example the $[TPUT]_{T-1}$ of the perturbed system can be calculated as follows:

$$[TPUT]_{T-1} = [TPUT]_T - \frac{NREJ}{\tau} \tag{3.6}$$

where τ is the observation interval.

4. GRADIENTS WITH RESPECT TO THE ARRIVAL RATE

4.1. Introduction

In this section we introduce an estimator, whose the purpose is to estimate performance sensitivities with respect to the arrival rate in systems involved in the decentralized threshold scheduling policy described in section 2. The estimator we have developed determines the derivative of mean values (e.g. average queue length) with respect to an arrival rate. The method is based on the work done by Reiman and Weiss [12], who used *likelihood ratio* techniques to prove their theorems. This is a typical result: if $E_\lambda(\psi)$ is the mean value of quantity ψ as a function of a Poisson rate λ then

$$\frac{d}{d\lambda}E_\lambda(\psi) = E_\lambda[(\frac{N}{\lambda} - T)\psi] \tag{4.1}$$

where T is the duration of the observation interval and N is the number of Poisson events in time T [11]. Therefore, using the idea that derivatives of expectations are themselves expectations, we can have a consistent estimate of $dE_\lambda(\psi)/d\lambda$ by simply estimating $E[(N/\lambda - T)\psi]$ during the observation period T. Reiman and Weiss assume in their work that the Poisson rate λ is known; we have slightly modify their method so that the Poisson rate λ can be estimated during the observation interval T. The method is extremely well suited to regenerative systems and can be implemented with very little increase in running time and memory requirements. Note also that the method is suitable for any parameter (not only Poisson rates) which does not change the possible sample paths, but merely changes their probability [12]. We will use this method to estimate the incremental delay due to remote job flow at host i of the distributed system considered throughout this work.

4.2. Gradient Estimation in Regenerative Systems

In this section we will show how the theorems derived in [11] can be used in *regenerative systems* for automatically estimating derivatives. Our discussion is based on

regenerative process theory which shows that steady state expected values can be obtained as ratios of expected values over a regenerative cycle. We assume that our queueing system can be modeled as an $M/G/1$ system with Poisson arrival rate λ.

We consider the initiation of a new busy cycle as the regeneration point. Let 0 be the number of customer initiating the $i-th$ busy cycle, N_i be the number of the first customer to encounter an empty system (indicating the initiation of the $(i+1)-th$ busy cycle) and T_i be the duration of the $i-th$ busy cycle. Assume that we observe the system for a number of busy cycles and W_i is the total waiting time in the $i-th$ busy period. We then have:

$$E[L] = \frac{E[W]}{E[T]} \tag{4.2}$$

where E[L] is the average queue length over the total observation period. What we really want to compute, is the quantity $dE[L]/d\lambda$. The quotient rule combined with (4.2) yields

$$\frac{dE[L]}{d\lambda} = \frac{\frac{dE[W]}{d\lambda}E[T] - \frac{dE[T]}{d\lambda}E[W]}{E[T]^2}. \tag{4.3}$$

Now we can use the results derived by Reiman and Weiss [11] to represent the derivatives appearing on the right hand side of (4.3) as

$$\frac{d}{d\lambda}E[W] = E[(\frac{N}{\lambda} - T)W] \tag{4.4}$$

and

$$\frac{d}{d\lambda}E[T] = E[(\frac{N}{\lambda} - T)T] \tag{4.5}.$$

The expectations at the right hand side of equations (4.4) and (4.5) can be very easily computed as averages over n busy cycles. Thus,

$$E[(\frac{N}{\lambda} - T)W] = \frac{1}{n}\sum_{i=1}^{n}(\frac{N_i}{\lambda} - T_i)W_i \tag{4.6}$$

$$E[(\frac{N}{\lambda} - T)T] = \frac{1}{n}\sum_{i=1}^{n}(\frac{N_i}{\lambda} - T_i)T_i \qquad (4.7).$$

Therefore, by substituting (4.6) and (4.7) in (4.3) we have

$$\frac{d}{d\lambda}E[L] = \frac{[\sum_{i=1}^{n} T_i][\sum_{i=1}^{n}(\frac{N_i}{\lambda} - T_i)W_i] - [\sum_{i=1}^{n} W_i][\sum_{i=1}^{n}(\frac{N_i}{\lambda} - T_i)T_i]}{[\sum_{i=1}^{n} T_i]^2} \qquad (4.8).$$

As we can easily see from equation (4.8) we only need to keep track of N_i, T_i and W_i and update the corresponding counter at the end of each busy cycle.

Although the above analysis implies that the Poisson arrival rate is known, we can slightly modify it so that it can be applied to a system which estimates its arrival rate simultaneously with its evolution. Actually, we only need to update the following counters at the end of each busy cycle:

$$C_1 = \sum_{i=1}^{n} T_i, \qquad C_2 = \sum_{i=1}^{n} W_i, \qquad C_3 = \sum_{i=1}^{n} N_i W_i,$$

$$C_4 = \sum_{i=1}^{n} T_i W_i, \qquad C_5 = \sum_{i=1}^{n} (T_i)^2, \qquad C_6 = \sum_{i=1}^{n} N_i T_i.$$

At the end of the observation period (i.e. after n busy cycles) we can use the above counters and the estimated arrival rate λ_{est} to compute the derivative of the average queue length with respect to the arrival rate as follows:

$$\frac{d}{d\lambda}E[L] = \frac{C_1(\frac{C_3}{\lambda_{est}} - C_4) - C_2(\frac{C_6}{\lambda_{est}} - C_5)}{(C_1)^2}. \qquad (4.9)$$

A similar analysis can be applied for estimating derivatives of other mean values with respect to a Poisson rate. As an example, we refer to the estimation of the derivative of the average response time $E[R]$ of a job with respect to the arrival rate λ using the formula

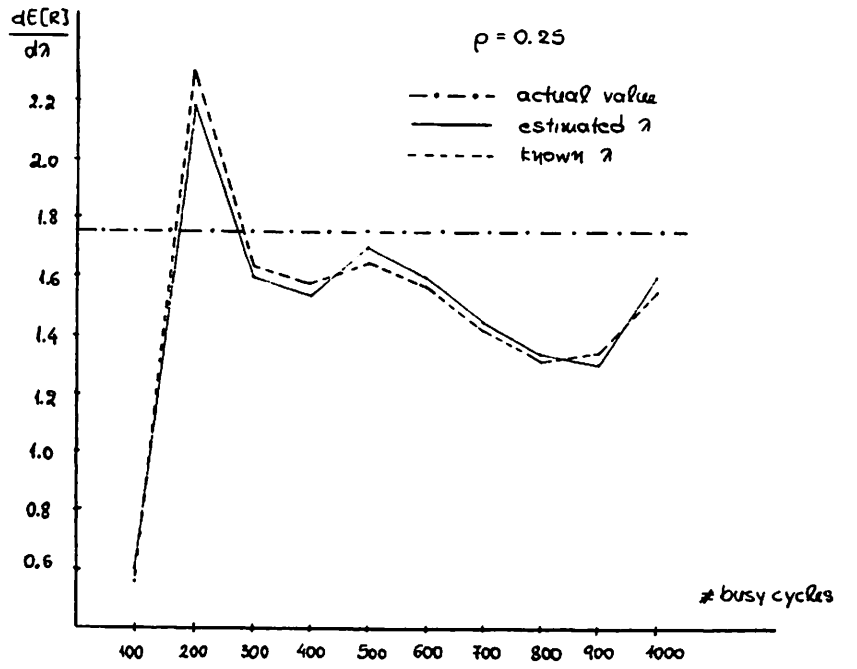
$$E[R] = \frac{E[W]}{E[N]} \tag{4.10}$$

where $E[W]$ is the average total waiting time and $E[N]$ is the average number of arrivals over n busy cycles.

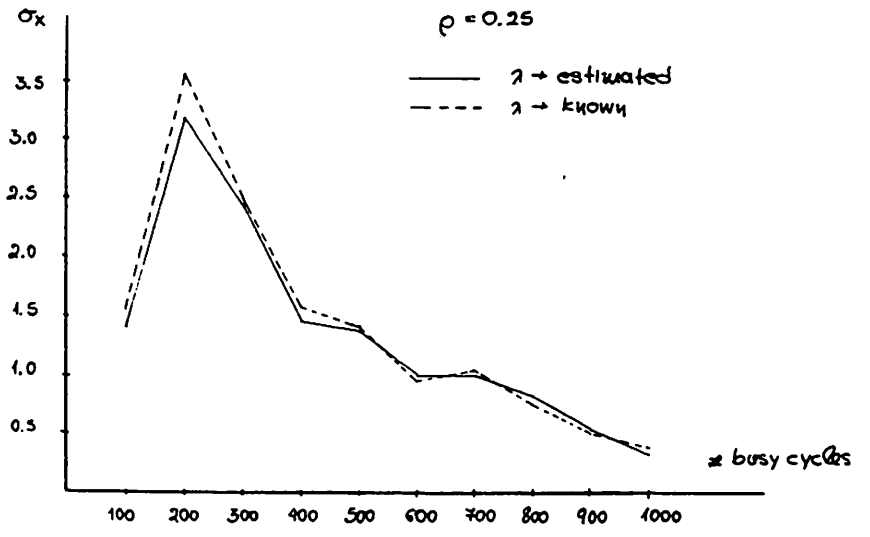
4.3. Simulation Results

To investigate the performance and convergence of the sensitivity analysis method described in the previous sections, we applied it to a regenerative simulation of an $M/M/1$ system with a single class of jobs. The choice of an $M/M/1$ queue was made because the simulation itself was very easy to write, and theoretical results are available, making it possible to check the numerical results. We investigate the performance of the estimation algorithm for three different values of the utilization, namely $\rho = 0.25$, 0.5 and 0.9, and for different lengths of the observation period. We also compare the convergence of the original algorithm which considers a known arrival rate to the convergence of the modified algorithm which uses the estimated arrival rate over the observation period. As it turns out from the simulation results the modified algorithm behaves as good as the original one.

Figures 4.1 to 4.6 show the behavior of the algorithm for the three different values of utilization mentioned above with respect to the length of the observation interval (in number of busy cycles). The first three figures (4.1 to 4.3) present simulation results for observation intervals in the range 100 to 1000 busy cycles. Figures 4.4 to 4.6 contain the corresponding results for observation periods in the range 1000 to 50000 busy cycles. Each figure consists of two parts. Part-a shows the estimated derivative of the average response time of a job with respect to the arrival rate λ for both the algorithms with known and estimated arrival rate. The resulting data points are the average over 20 runs. Part-b shows the standard deviation over the 20 runs.

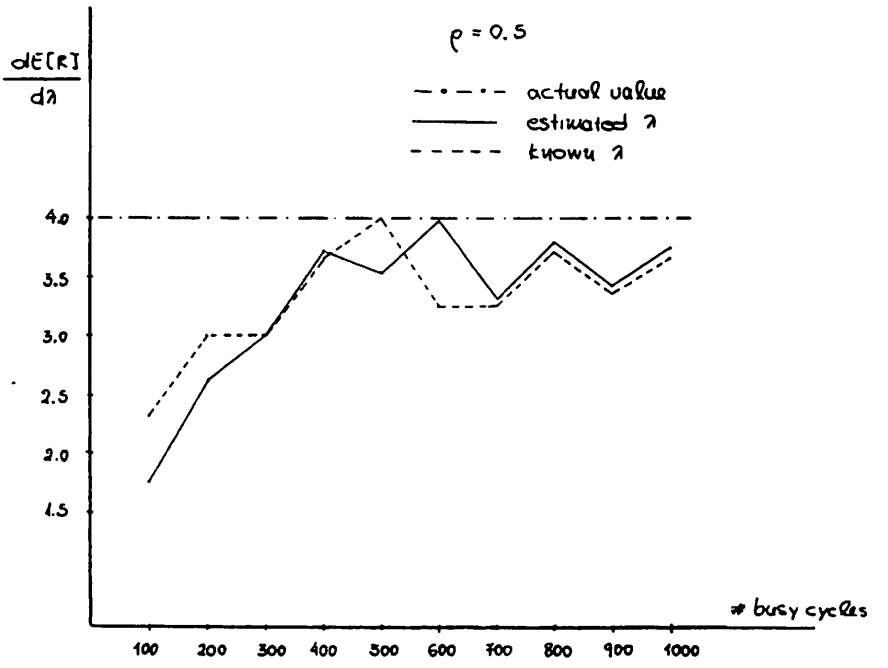


(a).

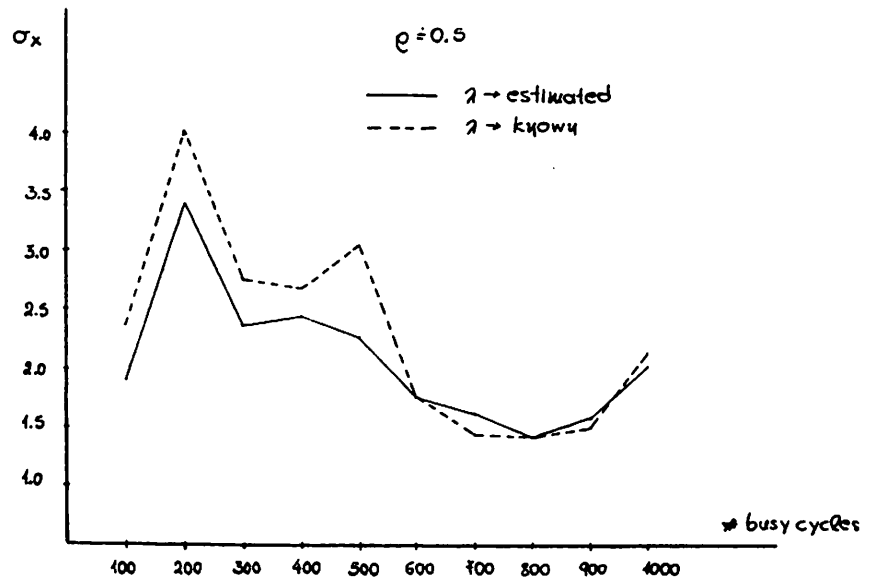


(b).

Fig. 4.1 - Estimation of $dE[R]/d\lambda$ for an $M/M/1$ system with $\rho = 0.25$ (observation period: 100 to 1000 busy cycles).

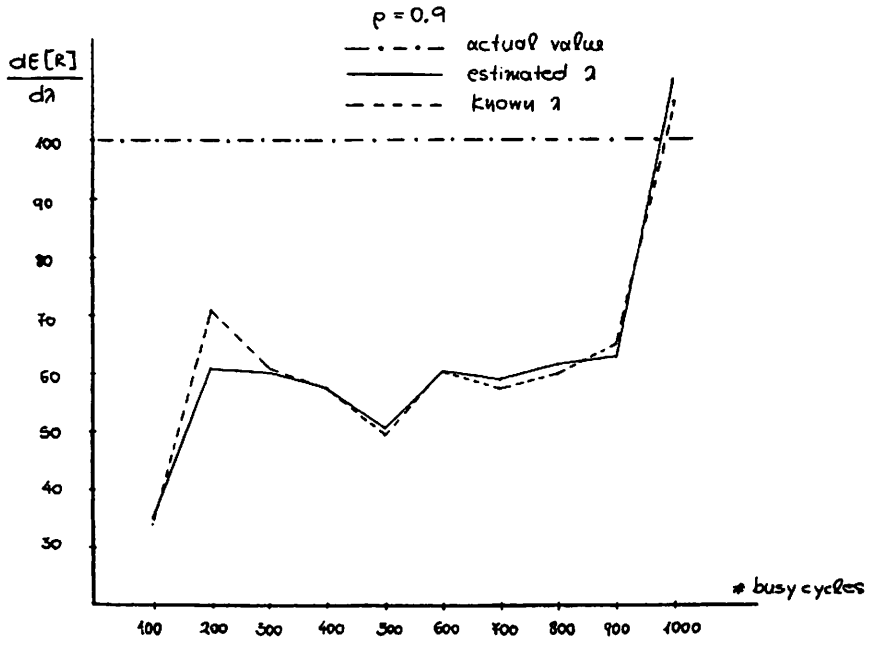


(a).

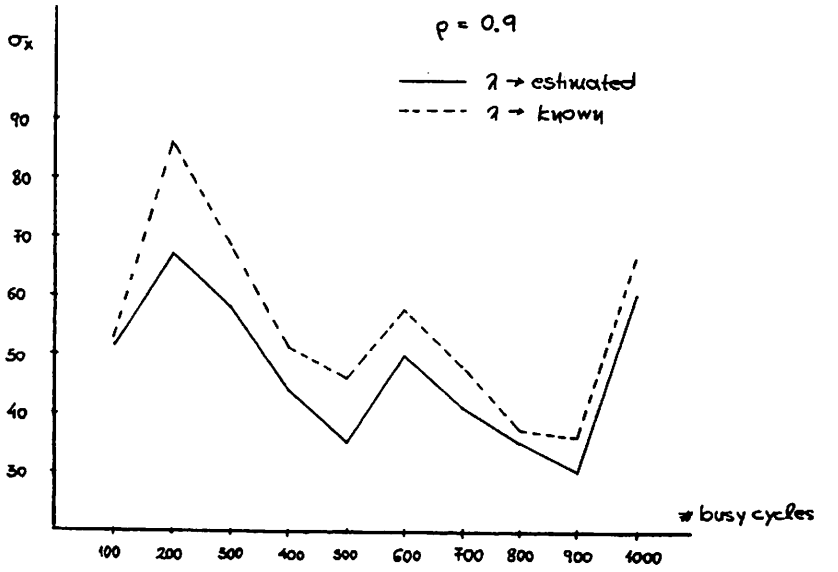


(b).

Fig. 4.2 - Estimation of $dE[R]/d\lambda$ for an $M/M/1$ system with $\rho = 0.5$ (observation period: 100 to 1000 busy cycles).

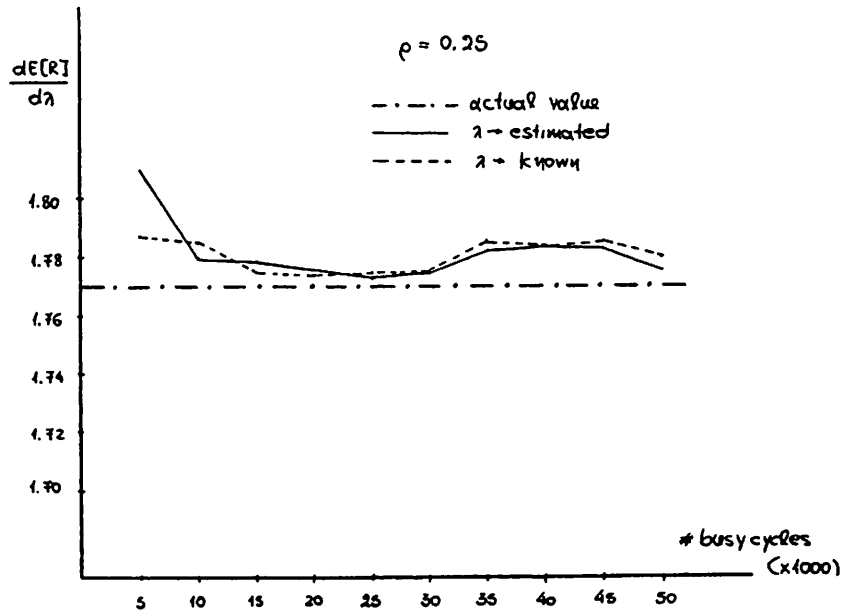


(a).

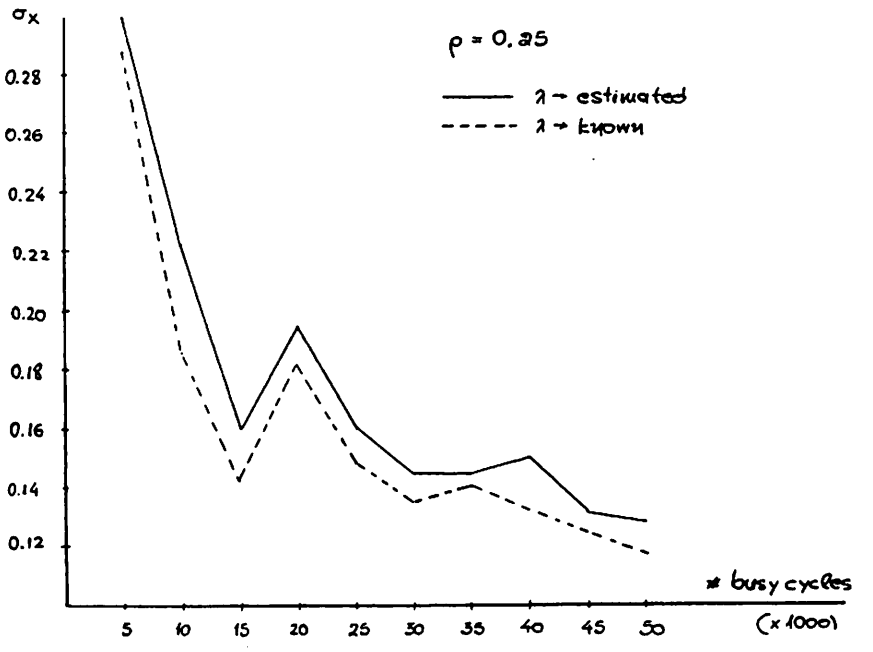


(b).

Fig. 4.3 - Estimation of $dE[R]/d\lambda$ for an $M/M/1$ system with $\rho = 0.9$ (observation period: 100 to 1000 busy cycles).

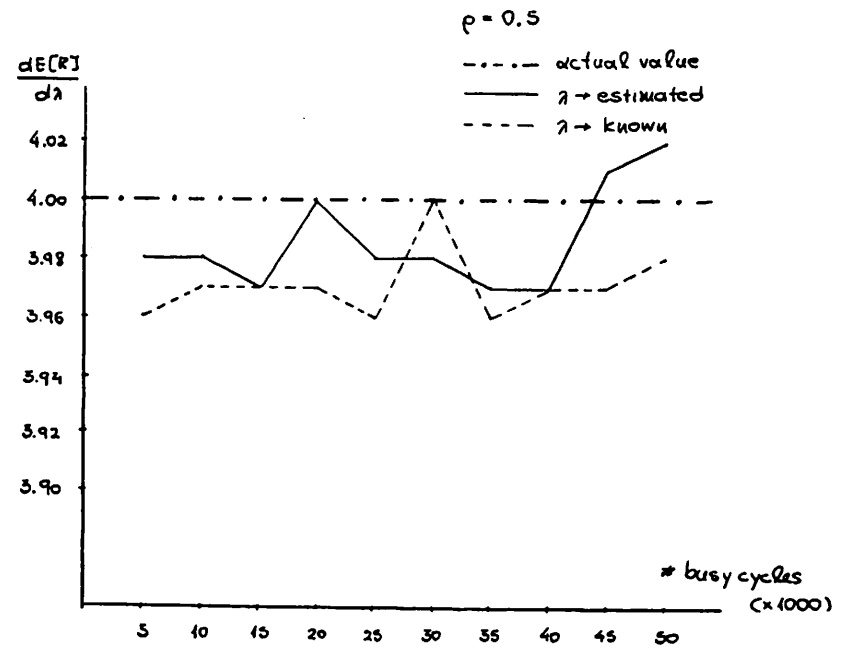


(a).

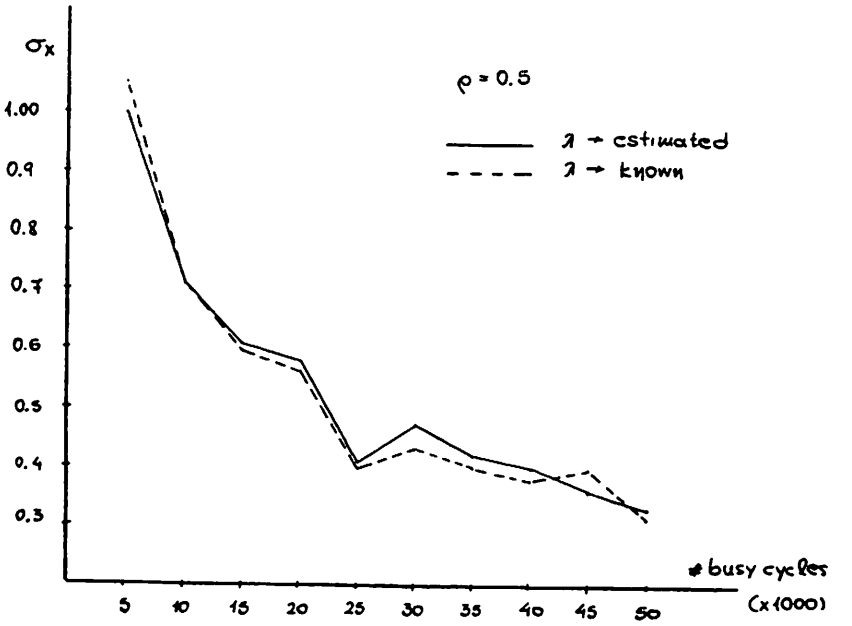


(b).

Fig. 4.4 - Estimation of $dE[R]/d\lambda$ for an $M/M/1$ system with $\rho = 0.25$ (observation period: 1000 to 50000 busy cycles).

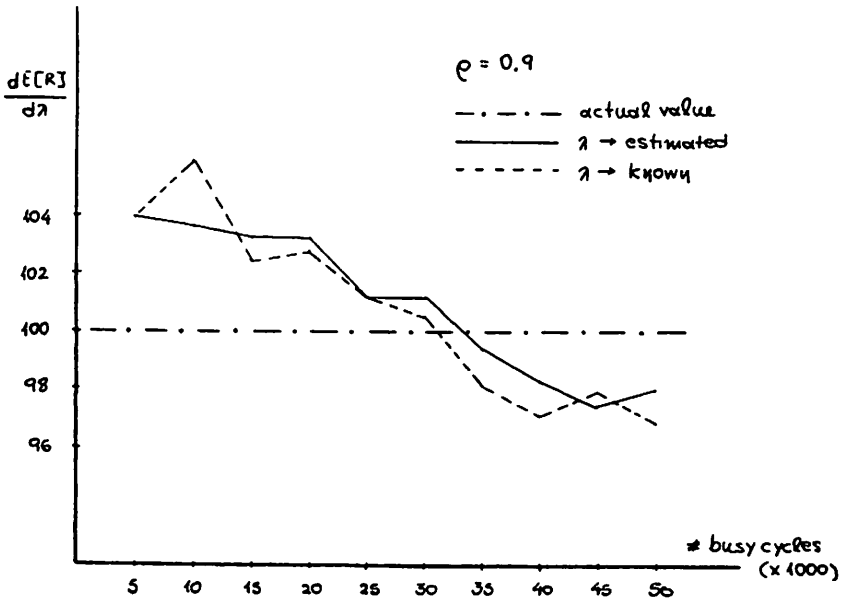


(a).

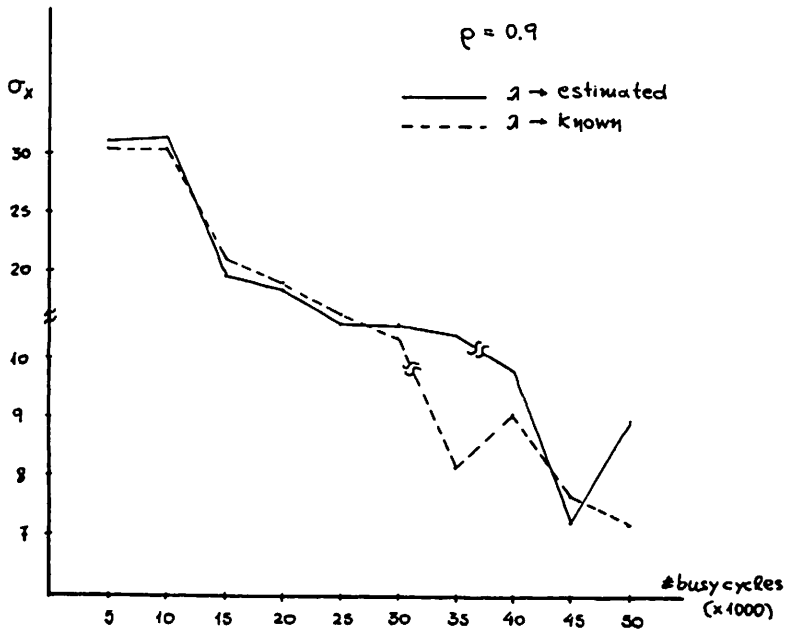


(b).

Fig. 4.5. - Estimation of $dE[R]/d\lambda$ for an $M/M/1$ system with $\rho = 0.5$ (observation period: 1000 to 50000 busy cycles).



(a).



(b).

Fig. 4.6 - Estimation of $dE[R]/d\lambda$ for an $M/M/1$ system with $\rho = 0.9$ (observation period: 1000 to 50000 busy cycles).

The actual values of $dE[R]/d\lambda$ for the different utilizations, indicated in all the Figures, can be computed very easily. We can notice that the observation period must be long enough in order for the algorithm to give consistent estimates of the desired gradient. If the observations take place over a short period of time, the system does not have the time to respond to the arrivals and furthermore, the variability of arrivals is relatively high over short periods of time. However, we believe that the method can be effectively applied in the distributed threshold scheduling policy described in Section 2. The reason is that we only need to compare derivatives without worrying about their absolute value. Therefore, the estimate derived even over a small number of busy cycles (e.g. 200 cycles or less) is probably good enough for that purpose.

4.4. Systems with two classes of jobs

So far, we have discussed a method (Estimator-B) for estimating derivatives of mean values with respect to a Poisson arrival rate in systems with one class of jobs. However, the systems involved in the decentralized threshold scheduling policy adopted in this work have two different classes of arrivals; namely local and remote jobs. In this section we show that we can very easily extend the method described in section 4.2 so that we are able to estimate the gradient of the average queue length with respect to the arrival rate of remote jobs. We recall that this gradient represents the incremental delay due to remote job flow and is required by the threshold updating algorithm described in section 2.3.

We assume that each host computer of the distributed system described in 2.1. can be modeled as a two class $M/G/1/\mathbf{T}$ system, where $\mathbf{T} = (T, \infty)$ since jobs belonging to the second class (remote jobs) are always accepted. Only local jobs are governed by the threshold parameter T. The purpose of *Estimator-B* is to estimate the gradient $dE[L]/d\lambda^{(r)}$, where $\lambda^{(r)}$ is the arrival rate of remote jobs. Using equation (4.2) and the quotient rule we have

$$\frac{dE[L]}{d\lambda^{(r)}} = \frac{\frac{dE[W]}{d\lambda^{(r)}}E[T] - \frac{dE[T]}{d\lambda^{(r)}}E[W]}{E[T]^2}. \tag{4.11}$$

Using equation (4.1) we can represent the derivatives in the right hand side of (4.11) as

$$\frac{d}{d\lambda^{(r)}}E[W] = E[(\frac{N^{(r)}}{\lambda^{(r)}} - T)W] \tag{4.12}$$

and

$$\frac{d}{d\lambda^{(r)}}E[T] = E[(\frac{N^{(r)}}{\lambda^{(r)}} - T)T] \tag{4.13}$$

where $N^{(r)}$ corresponds to the number of remote arrivals and $\lambda^{(r)}$ to the arrival rate of remote jobs. All the other variables have the same meaning as in equations (4.4) and (4.5). The same method as in section 4.2 can be used to compute the expectations in the right hand side of equations (4.12) and (4.13). We can also use the same modification as in section 4.2 to apply the method to a system which estimates the arrival rate of remote jobs while it runs.

Table 4.1 contains simulation results for an $M/M/1/(3,\infty)$ system with two classes of jobs, where only jobs belonging to the first class are governed by the threshold parameter. In our experiment the arrival rate of the first class (jobs coming in from the external world) is $\lambda = 0.5$ *jobs/sec*, the arrival rate of the second class (remote jobs) is $\lambda^{(r)} = 0.25$ *jobs/sec* and the service rate is $\mu = 1.0$ *jobs/sec*. Estimator-B is used to estimate the derivative of the average queue length with respect to the arrival rate of remote jobs. It can be derived analytically that the actual value of the above derivative for the particular system is $dE[L]/d\lambda^{(r)} = 2.70$.

#busy cycles	$[dE[L]/d\lambda^{(r)}]^{act}$	$[dE[L]/d\lambda^{(r)}]^{est}$	*%error*	*#runs*	σ_x
1000	2.700	2.446	9.40	5	0.277
5000	2.700	2.748	1.77	5	0.200
10000	2.700	2.690	0.37	5	0.157
20000	2.700	2.693	0.26	5	0.134

Table 4.1 - Estimation of $dE[L]/d\lambda^{(r)}$ for a 2-class $M/M/1/(3,\infty)$ system.

5. AN EXAMPLE

In this section we consider a simple example where host computers are interconnected through a communication network and execute the *distributed load balancing* algorithm described in section 2. The estimation techniques described in sections 3 and 4 are imbedded in the decentralized threshold scheduling policy to provide the necessary gradient information. Each host computer is modeled as a single-server queueing system with two classes of arrivals; namely local and remote jobs. In such a system we study the behavior of the algorithm in a static as well as in a changing environment.

The service time at host i is exponentially distributed with mean $1/\mu_i$ for $i = 1, 2, ..., N$. The interarrival times of jobs coming in a host from the external world are exponentially distributed with mean $1/\lambda_i$ for $i = 1, 2, ..., N$. The communication delay of a job due to its transfer for remote service is assumed to be an exponentially distributed random variable with mean $1/\mu_c$. Recall that jobs can move only once through the communication network, since remote jobs are always accepted. Whenever a host rejects a job because it reaches its threshold, it sends this job randomly to any one of the other hosts for remote service. This last assumption is reasonable in the case where all the hosts are homogeneous with the same utilization.

We have simulated a distributed system with five host computers which can be easily expanded to a system with N hosts. Each host executes the threshold updating algorithm independently to each other. All of them start the execution of the algorithm at time 0. Initial thresholds values for each host are drawn randomly from a uniform distribution. Each host observes itself for a number of busy cycles and at the end of the observation period computes the derivatives of its average queue length with respect to the local and remote job flow. Then, it sends the necessary gradient information to the other nodes and uses the currently available gradients reported by the other nodes, as well as its own gradient information to update its local threshold. Threshold updating completes an iteration of the algorithm at that node. Another observation period can start immediately and the host computer will execute the balancing policy with the new threshold value until the next iteration. Each host computer has a communication server that takes care of the job transfers between computers. The incremental delay due to job

transfers, required by the threshold updating algorithm, is approximated by the average communication delay μ_c.

We can make the following interesting observation regarding the application of the estimation techniques discussed in sections 3 and 4 in the decentralized threshold scheduling policy. If the initial threshold value at a particular node is large enough, so that this host never reaches its threshold during the observation period, then the incremental delay with respect to the local job flow cannot be determined using formula (2.6). If this derivative is assumed to be 0, due to the lack of information, this will probably lead to unnecessary increase of the threshold value during the next iteration. Therefore, a heuristic modification must be made; namely, if a host does not hit its threshold during the observation interval, then it automatically decrements its threshold value by one.

We present numerical examples for a particular set of system parameters. However, we have observed similar results for a wide range of system parameters. We first consider a system with five host computers with the same utilization $u_1 = u_2 = u_3 = u_4 = u_5 = 0.5$. All the processors have the same job processing rate ($\mu_1 = ... = \mu_5 = 1.0$). The average communication delay of a job is assumed to be 10% of the mean job service time. The initial threshold values are $T_1^{init} = ... = T_5^{init} = 10$. Figure 5.1 shows the threshold value at host 1 of the simulated distributed system with respect to the number of iterations of the algorithm at that host. Each observation period consists of 50 busy cycles. We observed similar behavior of the algorithm for all the host computers of the system. It turns out that the algorithm converges very quickly to the optimal (or near-optimal) threshold value. Obviously, when the initial thresholds are large, it takes more iterations for the algorithm to converge and this is due to the fact that the optimal thresholds are usually small values [2]. Figure 5.2 shows the average response time of a job in the system of the five hosts as a function of time. The two curves correspond to different initial threshold values. The solid curve is derived for initial values $T_1^{init} = ... = T_5^{init} = 10$ while the dotted curve is derived for initial threshold values $T_1^{init} = ... = T_5^{init} = 15$. As we can expect the system with the larger initial threshold values converges slower. A comparison is made with a system with no load balancing at all (NLB), where all the hosts are modeled as M/M/1 systems with utilization 0.5.

In Figures 5.3 and 5.4 we consider experiments where host computers have different

utilizations. In Figure 5.3 hosts 1, 2 and 3 have utilization 0.9 and hosts 4 and 5 have utilization 0.5. The average response time of a job is shown as a function of time. Two curves are presented corresponding to different initial threshold values and a comparison is made with a system with no load balancing. The same experiment is repeated for utilizations $u_1 = u_2 = u_3 = 1.2$ and $u_4 = u_5 = 0.5$ (Fig. 5.4). Obviously, hosts 1, 2 and 3 are saturated without load balancing. Since these three nodes are overloaded, a busy cycle termination occurs very rarely and so does a threshold update. Therefore, starting with large initial threshold values at the overloaded nodes results in larger average response time of a job (Fig. 5.4).

We next study the *adaptivity* of the threshold updating algorithm in a dynamically changing system environment. In such a case, when there is an imbalance in incremental delays due to a change in the system environment, we expect that the algorithm corrects the imbalance properly so that the system performance may be improved. Figures 5.5 and 5.6 consider an example where hosts in the system have the same processing power ($\mu_1 = ... = \mu_5 = 1.0$). Initially all the five hosts have the same utilization of 0.5. Right after the 2500-th time unit (simulation time) the utilization of hosts 1, 2 and 3 changes to 0.9 (Figure 5.5) and 1.2 (Figure 5.6) respectively. Right after the 5000-th time unit (simulation time) the utilization of hosts 1, 2 and 3 returns to 0.5. As it turns out from the simulation results, the algorithm adapts smoothly to a change of increasing or decreasing workload. After a short transient time the average response time converges to the value we had observed for the corresponding system in a static environment (Figures 5.2, 5.3 and 5.4). In Figure 5.6 the solid curve indicates the behavior of the algorithm in a system changing environment when the initial threshol values are 10 for all the nodes. The dotted curve represents its behavior in the same environment but in this case threshold value is kept constant $T = 2$ for all the nodes.

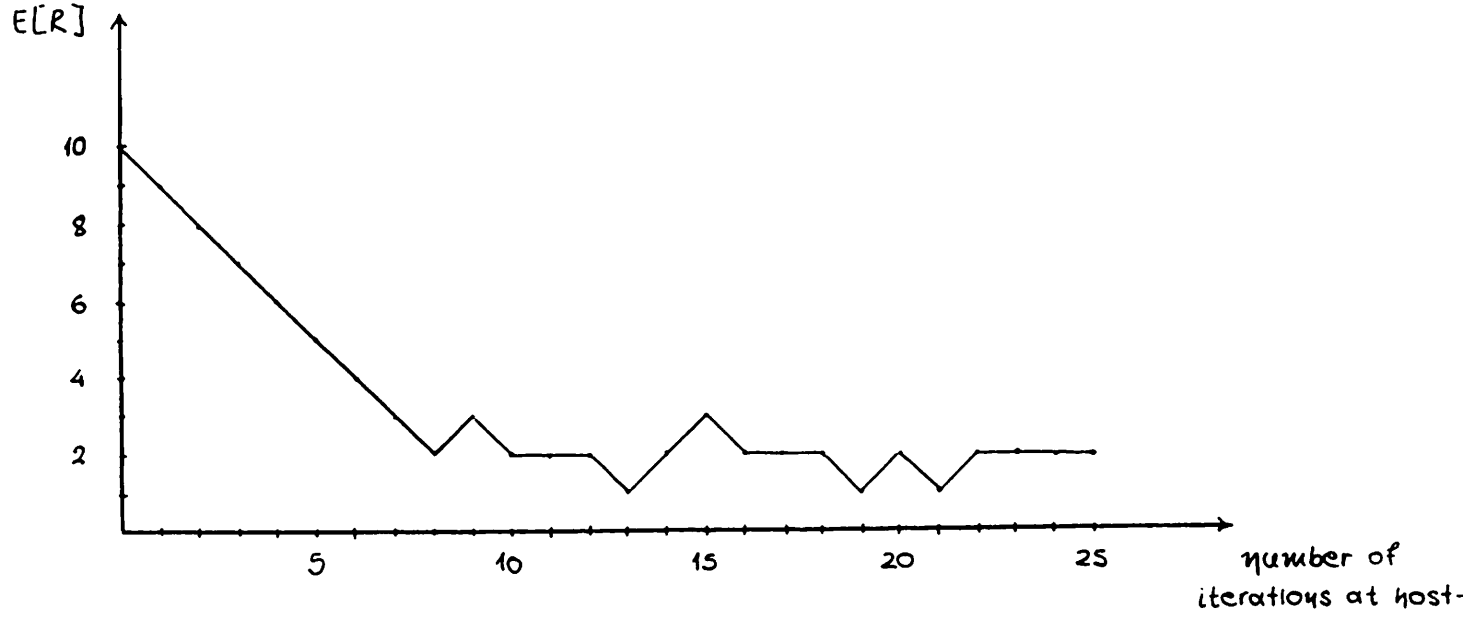


Fig. 5.1 - Behavior of the threshold updating algorithm at host-1 of a distributed system with 5 hosts ($T_1^{init} = ... = T_5^{init} = 10$ and $u_1 = ... = u_5 = 0.5$).

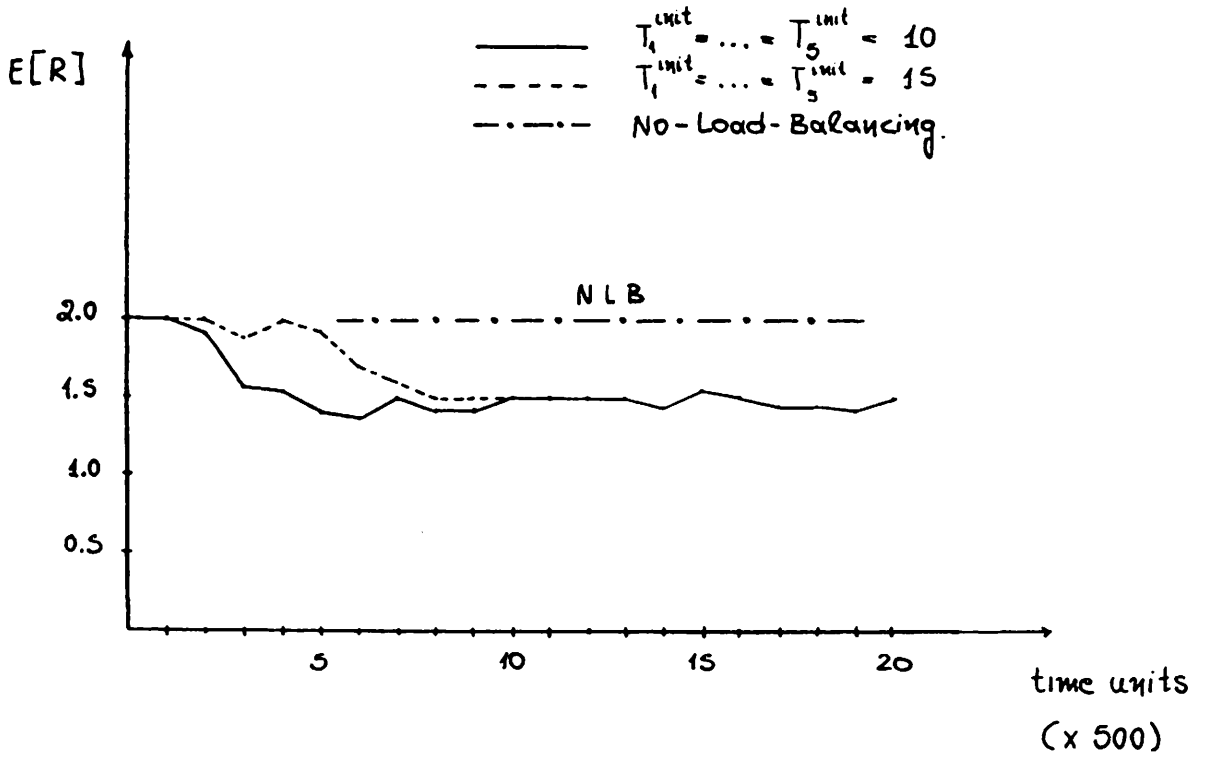


Fig. 5.2 - Average response time a job in the system of five hosts as a function of time ($u_1 = \ldots = u_5 = 0.5$).

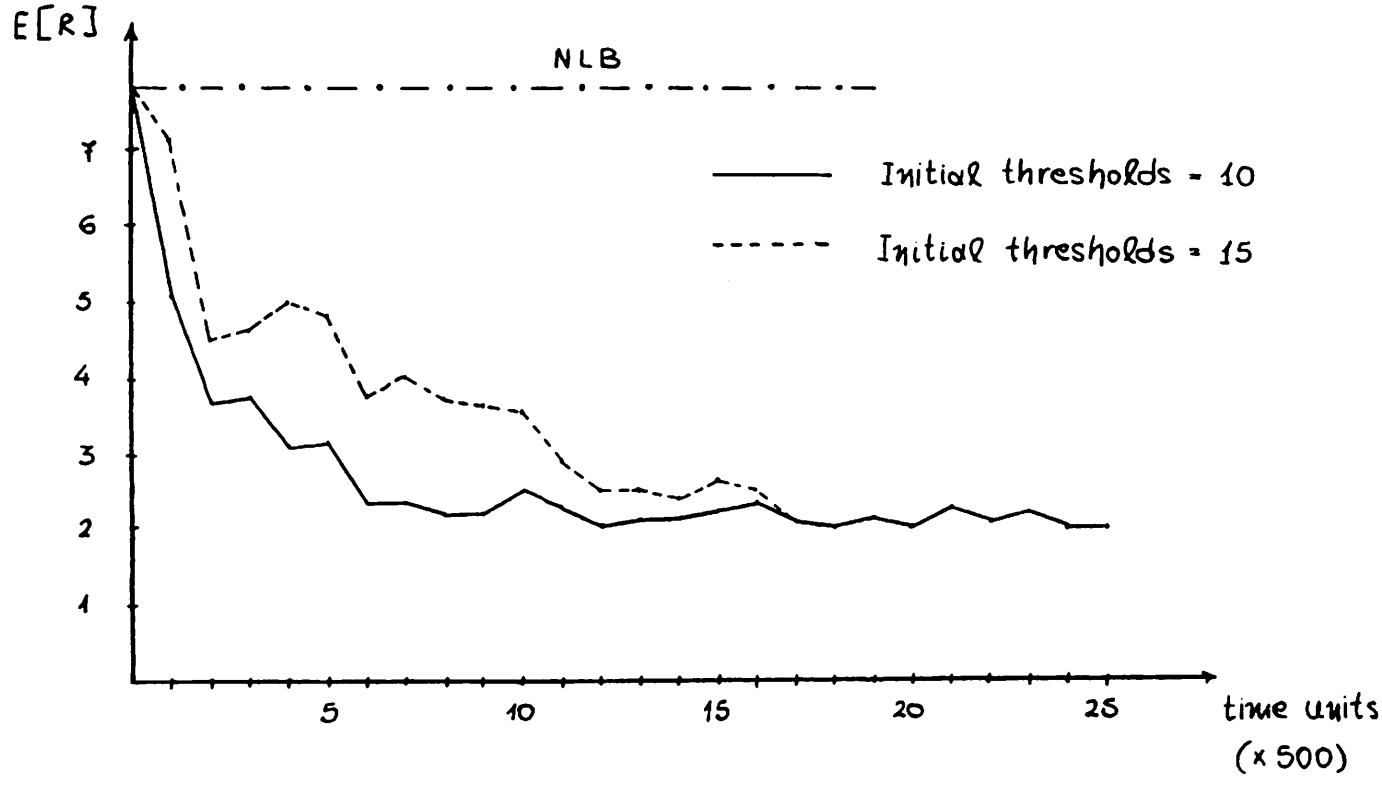


Fig. 5.3 - Average response time a job in the system of five hosts as a function of time ($u_1 = u_2 = u_3 = 0.9$ and $u_4 = u_5 = 0.5$).

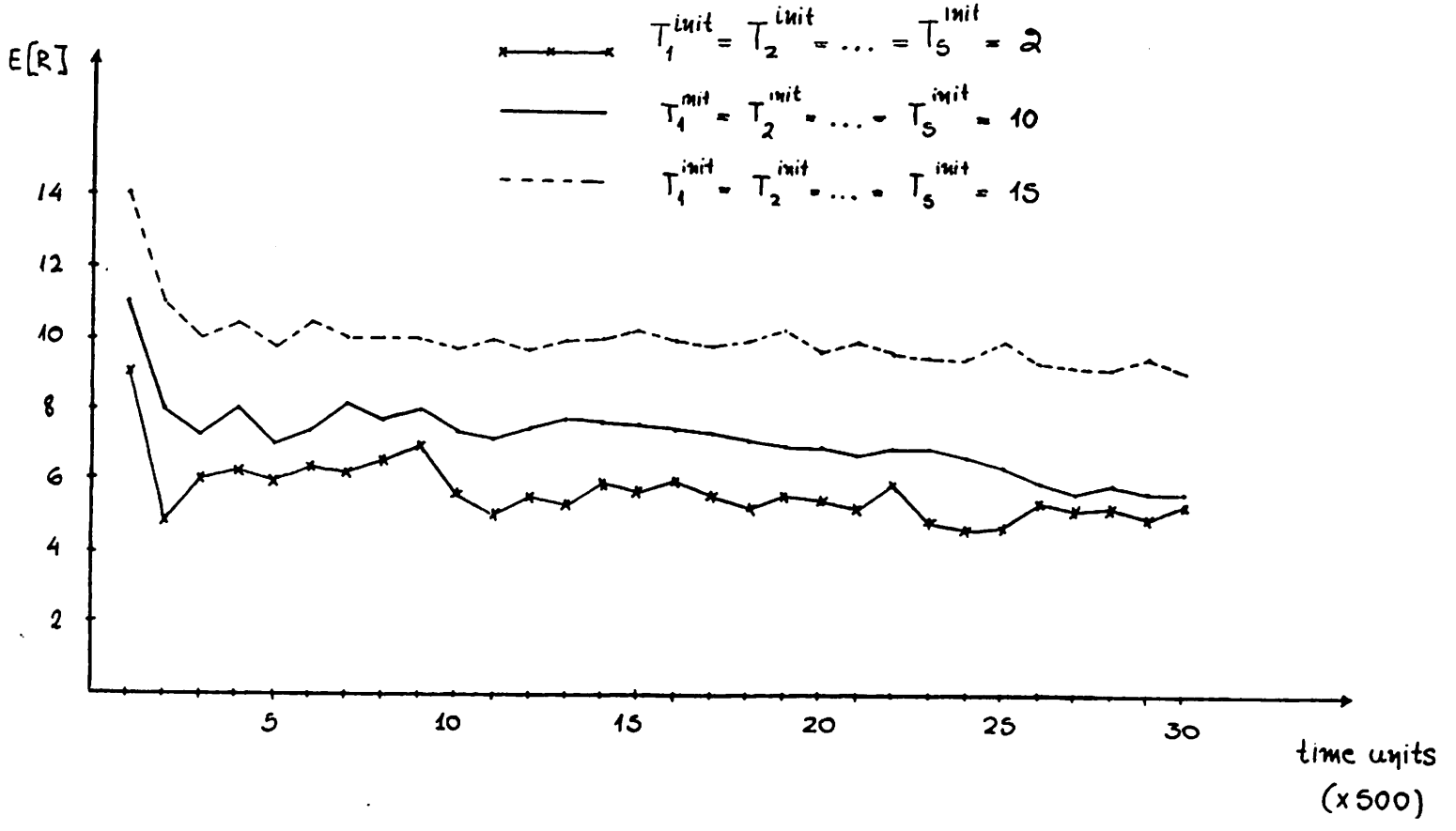


Fig. 5.4 - Average response time of a job in the system of five hosts as a function of time ($u_1 = u_2 = u_3 = 1.2$ and $u_4 = u_5 = 0.5$).

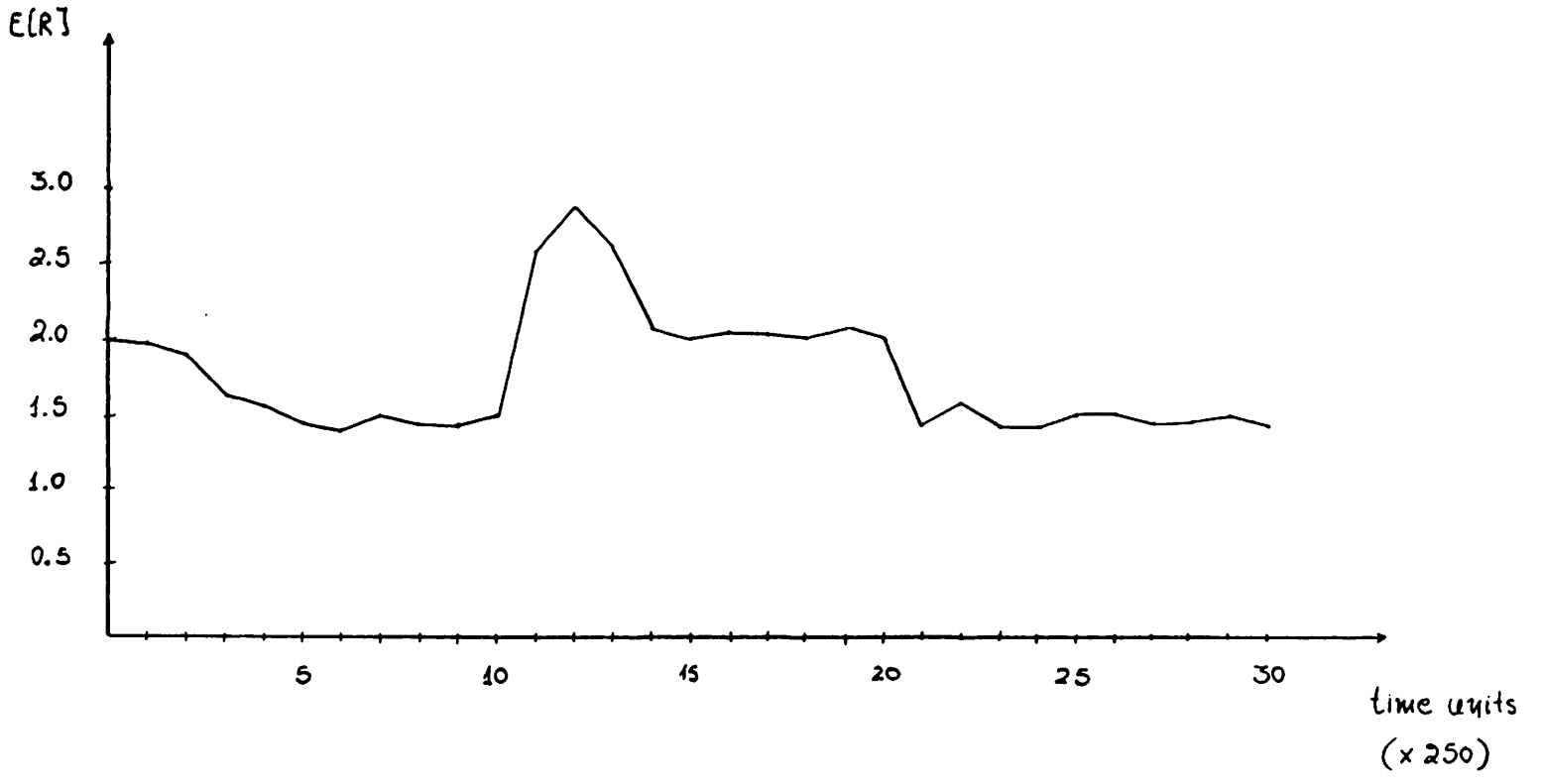


Fig. 5.5 - Behavior of the threshold updating algorithm
in a changing system environment
(utilization at nodes 1, 2 and 3 changes from
50% to 90% and back to 50%).

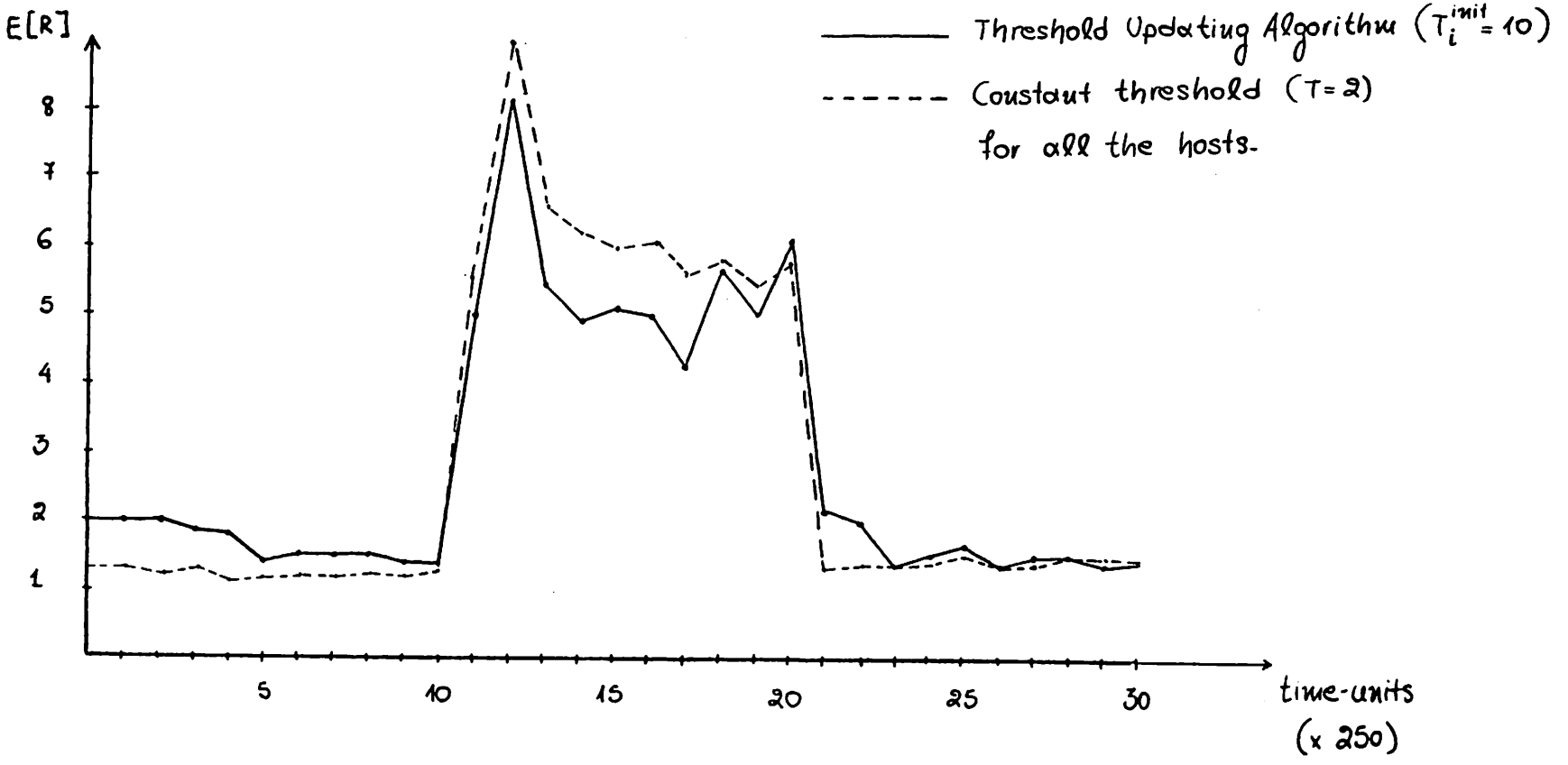


Fig. 5.6 - Behavior of the threshold updating algorithm
in a changing system environment
(utilization at nodes 1, 2 and 3 changes from
50% to 120% and back to 50%).

6. CONCLUSIONS

In this work we considered the problem of improving the performance of a distributed system by using *load balancing* techniques to smooth-out periods of high congestion at individual nodes. Specifically, we adopted a class of load balancing techniques, that use a threshold policy at each host to make decisions for job transfers. We put emphasis on the problem of determining the optimum values of threshold parameters in order to yield optimal or near-optimal performance. An efficient distributed algorithm has been adapted to the specific load balancing policy for that purpose. The algorithm is iterative in nature and threshold parameters are updated at each iteration. Between updates, each host computes and exchanges with other hosts gradient information. This information is then used to update load balancing parameters in the next iteration.

Obviously, the above distributed algorithm is based on the knowledge of performance sensitivity information. Each host must be able to estimate the change in throughput and expected queue length due to a change in either the threshold or the job arrival rate. Two different methods towards estimating the above gradients have been proposed in this work. Both of them effectively provide performance sensitivity estimates while the actual system is running (on-line estimation), by using real data gathered during an observation interval. In both the methods the arrival rate is estimated during the execution of the estimation algorithm.

Our approach towards estimating the gradients with respect to the threshold relies on the assumption that either the arrival process or the service time are exponentially distributed. The performance of the proposed estimation technique has been investigated through simulations. As it turns out from the simulation results, the estimation algorithm converges very fast even for short observation periods (i.e. 200 job completions). Furthermore, the memory requirements and the increase in running time are very low.

A different technique is proposed for estimating gradients with respect to the arrival rate. The method is extremely well suited to regenerative systems. In our case, the beginning of a new busy cycle is considered as the regeneration point. By evaluating the above estimation technique through simulations, we realized that the observation period must be long enough (e.g. 2000 busy cycles) in order for the estimator to give consistent

estimates of the desired gradient. However, the method can be effectively applied to systems involved in the decentralized threshold scheduling policy discussed in this work, where we only need to compare derivatives without worrying about their absolute value.

Both the proposed estimation techniques have been imbedded in the updating threshold algorithm and simulation results have been produced for a system with five host computers modeled as single server queueing systems. It turns out that after a finite number of algorithm iterations, the behavior of the system, in a static environment is confined in a neighborhood of the optimal performance. A great improvement in the performance measure (average response time of a job) has been observed in the system executing the distributed load balancing algorithm compared to a system with no load balancing at all. Although we used short observation periods (50 or 100 busy cycles) in most of our experiments, the performance of the algorithm was not affected. It appears that accuracy in gradient estimation is not very crucial, since it is the *relative* values of the gradients, not their *absolute* values, which are more significant. Since the optimal thresholds are usually small values (e.g. less than six), it takes more iterations for the algorithm to converge if the initial thresholds are large. A heuristic modification has been made in order to improve the performance of the technique estimating gradients with respect to threshold, when the initial threshold values are large enough so that the nominal system does not even reach its threshold. It is interesting to notice that the overhead of exchanging gradients between the hosts of the distributed system is small since they are not an istantaneous information such as queue length. An interesting problem, is the *adaptivity* of the algorithm in a dynamically changing system environment. In such a case, when there is an imbalance in incremental delays due to a change in the system environment, we expect that the algorithm corrects the imbalance properly so that the system performance may be improved. Simulation results pointed out that the algorithm adapts smoothly to a change of the workload. When the system environment changes over time, this algorithm can run in the background so that it can track the system variation in a quasi-static manner [2].

REFERENCES

[1]. K. J. LEE, D. TOWSLEY, "Quasi-static decentralized load balancing with site constraints", submitted to ACM Sigmetrics Conf., 1987.

[2]. K. J. LEE, Load Balancing in Distributed Computer Systems, Ph.D. Thesis, Dept. Elect. & Comp. Eng., Univ. of Massachusetts, 1987.

[3]. K. J. LEE, D. TOWSLEY, "A comparison of priority-based decentralized load balancing policies", Proc. of Performance 1986 and ACM Sigmetrics Conf. 1986.

[4]. K. J. LEE, D. TOWSLEY, "Distributed Algorithms for decentralized load balancing in star-configured systems", submitted to *IEEE Trans. on Computers*, 1986.

[5]. D. EAGER, E. LAZOWSKA, J. ZAHORJAN, "Adaptive load sharing in homogenious distributed systems", *IEEE Trans. on Software Eng.*, SE-12, pp. 662-675, May 1986.

[6]. A. TANTAWI, D. TOWSLEY, "Optimal static load balancing in distributed computer systems", *Journal of ACM*, vol. 32, pp. 445-465, April 1985.

[7]. Y. WANG, R. MORRIS, "Load sharing in distributed systems", *IEEE Trans. on Computers*, vol. C-34, pp. 204-217, March 1985.

[8]. D. LUENBERGER, Introduction to Linear and Nonlinear Programming, Addison-Wesley, 1973.

[9]. C. CASSANDRAS, Y. HO, "An event domain formalism for sample path perturbation analysis of discrete event dynamic systems", *IEEE Trans. on Automatic Control*, AC-30, pp. 1217-1221, 1985.

[10]. R. SURI, M. ZAZANIS, "Perturbation Analysis gives strongly consistent sensitivity estimates for the M/G/1 queue", *Operations Research*, 1985.

[11]. C. CASSANDRAS, "On-line optimization of a flow control strategy", *IEEE Trans. on Automatic Control*, AC-31, Feb. 1986.

[12]. M. REIMAN, A. WEISS, "Sensitivity Analysis for Simulations via Likelihood Ratios", AT&T Bell Labs Report, 1987.