The Effect of Scheduling Discipline on Sender-Initiated and Receiver-Initiated Adaptive Load Sharing in Homogeneous Distributed Systems†

Sivarama P. Dandamudi
School of Computer Science, Carleton University, Ottawa, Ontario K1S 5B6, Canada
sivarama@scs.carleton.ca

ABSTRACT

Load sharing is a technique to improve the performance of distributed systems by distributing the system workload from heavily loaded nodes to lightly loaded nodes in the system. Previous studies have considered two adaptive load sharing policies: *sender-initiated* and *receiver-initiated*. In the sender-initiated policy, a heavily loaded node attempts to transfer work to a lightly loaded node and in the receiver-initiated policy a lightly loaded node attempts to get work from a heavily loaded node. Most of the previous studies have assumed the first-come/first-served (FCFS) node scheduling policy; furthermore, analyses and simulations in these studies have been done under the assumption that the job service times are exponentially distributed and the job arrivals form a Poisson process. The goal of this paper is to fill the void in the existing literature. We study the impact of these assumptions on the performance of the sender-initiated and receiver-initiated policies. We consider three node scheduling policies – first-come/first-served (FCFS), shortest job first (SJF), and round robin (RR) policies. Furthermore, we also look at the impact of variance in the inter-arrival times and in the job service times. Our results show that:

(i) When non-preemptive node scheduling policies (FCFS and SJF) are used, the receiver-initiated policy is more sensitive to variance in inter-arrival times than the sender-initiated policies and the sender-initiated policies are relatively more sensitive to the variance in job service times.

(ii) When the preemptive node scheduling policy (RR) is used, in most cases, the sender-initiated policy provides a better performance than the receiver-initiated policy. This is in contrast to the prevailing intuition that is based on the FCFS node scheduling policy.

(iii) When service time variance is high, sender-initiated load sharing policy provides the best performance among the policies considered here when the round robin node scheduling policy is used.

Index Terms: Adaptive load sharing, scheduling policies, performance evaluation, homogeneous distributed systems, sender-initiated policies, receiver-initiated policies.

1. INTRODUCTION

Load sharing is a technique to improve the performance of distributed systems by distributing the system workload from heavily loaded nodes, where service is poor, to lightly loaded nodes in the system. Load sharing policies may be either static or adaptive (see [Har90] for a survey). Static policies only use information about the average system behaviour while the adaptive policies use the current or recent system state information. The chief advantage of the static policies is their simplicity (i.e., no need to collect system state information). The disadvantage of these policies is that they cannot respond to changes in system state (thus the performance improvement is limited). On the other hand, adaptive policies react to the system state changes dynamically and therefore provide substantial performance improvements over the static policies. However, this additional performance benefit is obtained at a cost - the cost of collecting and maintaining the system state information. This paper considers adaptive load sharing in homogeneous distributed systems in which the nodes are identical and the workload of each node is similar.

Two important components of an adaptive policy are a transfer policy and a location policy [Eag86a]. The transfer policy determines whether a job is processed locally or remotely and the location policy determines the node to which a job, selected for possible remote execution, should be sent. Typically, transfer policies use some kind of load index threshold to determine whether the node is heavily loaded or not. Several load indices such as the CPU queue length, time averaged CPU length, CPU utilization, the amount of available memory etc. have been

†A previous version of this paper appears in *Proc. Int. Conf. Distributed Computing Systems*, Vancouver, 1995.

proposed/used [Zho88, Kun91, Shi92]. It has been reported that the choice of load index has considerable effect on the performance and that the simple CPU queue length load index was found to be the most effective [Kun91].

The adaptive policies can employ either a centralized or a distributed location policy. In the centralized policy, state information is collected by a single node and other nodes would have to consult this node for advice on the system state. In the distributed policy, system state information is distributed to all nodes in the system. The centralized policy has the obvious disadvantage of diminished fault-tolerance and the potential for performance bottleneck (although some studies have shown that the node collecting and maintaining the system state need not be a performance bottleneck for reasonably large distributed systems [The88]). The use of the centralized policy is often limited to a cluster of nodes in a large distributed system [Zho93]. The distributed control eliminates these disadvantages associated with the centralized policy; however, distributed policy may cause performance problems as the state information may have to be transmitted to all nodes in the system. (Previous studies have shown that this overhead can be substantially reduced [Eag86a, Eag86b]).

Location policies can be divided into two basic classes: *sender-initiated* or *receiver-initiated*. In sender-initiated policies, congested nodes attempt to transfer work to lightly loaded nodes. In receiver-initiated policies, lightly loaded nodes search for congested nodes from which work may be transferred. It has been shown that, when the first-come/first-served (FCFS) scheduling policy is used, sender-initiated policies perform better than receiver-initiated policies at low to moderate system loads [Eag86b]. This is because, at these system loads, the probability of finding a lightly loaded node is higher than that of finding a heavily loaded node. At high system loads, on the other hand, receiver-initiated policies are better because it is much easier to find a heavily loaded node at these system loads. There have also been proposals to incorporate the good features of both these policies [Shi90, Shi92].

In the remainder of this section, we briefly review related work and outline the paper. There is a lot of literature published on this topic (an excellent overview of the topic can be found in [Shi92]) [Alo88, Cho90, Eag86a, Eag86b, Hac87, Hac88, Kru88, Kun91, Lin92, Lit88, Liv82, Rom91, Sch91, Shi90, The88, Wan85, Zho88]. Here, for the sake of brevity, we will only review a subset of this literature that is directly relevant to our study.

Eager et al. [Eag86a, Eag86b] provide an analytic study of adaptive load sharing policies. They showed that the sender-initiated location policy performs better at low to moderate system loads and the receiver-initiated policy performs better at high system loads. They have also shown that the overhead associated with state information collection and maintenance under the distributed policy can be reduced substantially (by probing only a few randomly selected nodes about their system state as opposed to all nodes in the system). Shivaratri and Krueger [Shi90] have proposed and evaluated, using simulation, two location policies that combine the good features of the sender-initiated and receiver-initiated location policies. Schaar et al. [Sch91] consider the impact of communication delay on the performance of load sharing policies.

Hac and Jin [Hac87] have implemented a receiver-initiated algorithm and evaluated its performance under three workload types: CPU-intensive, IO-intensive, and mixed workloads. They compare the performance of their load balancing policy with that when no load balancing is employed. They conclude that, for all the three types of workload, load balancing is beneficial. Unfortunately, they do not compare the performance of various load balancing policies that have been proposed in the literature.

Dikshit, Tripathi, and Jalote [Dik89] have implemented both sender-initiated and receiver-initiated policies on a five node system connected by a 10Mb/s communication network. As a part of their study they have conducted an experiment on the impact of service time variance, but the coefficient of variation is less than or equal to 1 (taken from exponential and uniform distributions).

A common thread among the previous studies, with a few exceptions, is that they assume the FCFS node scheduling policy. Furthermore, analyses and simulations in these studies have been done under the assumption that the job service times are exponentially distributed and the the job arrivals form a Poisson process (i.e., job inter-arrival times are exponentially distributed). In this context, it should be noted that the service time distribution of the data collected by Leland and Ott [Lel86] from over 9.5 million processes has been shown to have a coefficient of variation of 5.3 [Kru88].

The goal of this paper is to fill the void in the existing literature. As a result, this paper concentrates

on the impact of node scheduling policies on the performance of the sender-initiated and receiver-initiated adaptive load sharing policies. We consider three node scheduling policies - first-come/first-serve (FCFS), shortest job first (SJF), and round robin (RR) policies. Furthermore, we also study the impact of variance in the inter-arrival times and in the job service times.

The prevailing intuition, based on the FCFS node scheduling policy, is that the sender-initiated policies are superior only at low to moderate system loads while the receiver-initiated policies are better at high system loads. This has prompted several researchers to devise hybrid policies that behave like the sender-initiated policy at low to moderate system loads and like the receiver-initiated policy at high system loads [Shi90, Mir89, Dik89].

In comparing the performance of the sender-initiated and receiver-initiated policies using these three node scheduling policies, our results show that (for the parameter values considered here):

When non-preemptive node scheduling policies (FCFS and SJF) are used:

- receiver-initiated policies are highly sensitive to variance in the inter-arrival times (i.e., to clustered arrival of jobs) and this sensitivity can be reduced considerably by using reinitiation;
- sender-initiated policies are relatively more sensitive to variance in job service times.

When the preemptive node scheduling policy (RR) is used:

- the sender-initiated policy is insensitive to the system load, variance in the job service times for a given variance in the inter-arrival times;
- the receiver-initiated policy, on the other hand, is sensitive to the system load, variance in service time and the variance in inter-arrival time;
- when the variance in service time is high, the sender-initiated policy provides the best performance among the policies considered in this study. Note that the experimental data in [Lel86] shows that the service time variance can be high.

The remainder of the paper is organized as follows. Section 2 discusses the load sharing and node scheduling policies. Section 3 describes the workload and system models we have used in performing this study. The results are discussed in Sections 4 and 5. Section 6 is a summary.

2. LOAD SHARING AND NODE SCHEDULING POLICIES

This section presents the load sharing policies and the node scheduling policies. Section 2.1 describes the sender-initiated and receiver-initiated load sharing policies. The three node scheduling policies are described in Section 2.2.

2.1. Load Sharing Policies

Our load sharing policies are slightly different from those reported in the literature. These subtle changes are motivated by the following aspects of locally distributed systems. First, workstations use some kind of round robin scheduling policy for efficiency and performance reasons. Thus, it is necessary to have a number of jobs that is related to the degree of multiprogramming (T_m). In the implementation of Hac and Jin [Hac87], they have used a similar parameter (they call it capacity C). Second, it is expensive to migrate active processes and, for performance improvement, such active process migration is strictly not necessary. It has been shown that, except in some extreme cases, active process migration does not yield any significant additional performance benefit [Eag88, Kru88]. Load sharing facilities like Utopia [Zho93] will not consider migrating active processes. As a result, we assume that each node has two queues that hold the jobs: a *job queue*, which is a queue of all the jobs (i.e., active jobs) that are to be executed locally and a *job waiting queue* that will hold those jobs that are waiting to be assigned to a node. Note that only jobs in the job waiting queue are eligible for load distribution.

The assumption of job waiting queue is reasonable from implementation point of view. We quote two example implementations that have used job waiting queues. The implementation in [Hac87] uses a similar job queue (they call it the local process queue). The implementation described in [Dik89] has a long-time scheduler that maintains a queue of jobs before they are submitted to the UNIX operating system for execution. The degree of multiprogramming is used by the long-term scheduler to control the number of jobs that are being executed at any given time. They have implemented the long-term scheduler for the same reasons that we have described here (i.e., to avoid process migration). Job waiting queue is also used in [Sue92].

We will study two adaptive load sharing policies - sender-initiated and receiver-initiated policies. In both these policies, we use the same threshold-based transfer policy (which is different from the one reported in the

literature for reasons described above). When a new job arrives at a node, the transfer policy looks at the job queue length at the node. It transfers the new job to the job queue (i.e., for local execution) if the job queue length is less than the degree of multiprogramming T_m. Otherwise, the job is placed in the job waiting queue of the node for a possible remote execution. The location policy, when invoked, will actually perform the node assignment. The two location policies are described below.

Sender-initiated policy

When a new job arrives at a node, the transfer policy described above would decide whether to place the job in the job queue or in the job waiting queue. If the job is placed in the job waiting queue, the job is eligible for transfer and the location policy is invoked. The location policy probes (up to) a maximum of probe limit P_l randomly selected nodes to locate a node with the job queue length less than T_S. If such a node is found, the job is transferred for remote execution. The transferred job is directly placed in the destination node's job queue when it arrives. Note that probing stops as soon as a suitable target node is found. If all probes fail to locate a suitable node, the job is moved to the job queue to be processed locally. When a transferred job arrives at the destination node, the node must accept and process the transferred job even if the state of the node at that instance has changed since probing. In this paper, we assume that $T_m = T_S$ for the sender-initiated load sharing policy.

Receiver-initiated policy

When a new job arrives at node *S*, the transfer policy would place the job either in the job queue or in the job waiting queue of node *S* as described before. The location policy is invoked by nodes only at times of job completions. The location policy of node *S* attempts to transfer a job from its job waiting queue to its job queue if the job waiting queue is not empty. Otherwise, if the job queue length of node *S* is less than T_R, it initiates the probing process as in the sender-initiated policy to locate a node *D* with a non-empty job waiting queue. If such a node is found within P_l probes, a job from the job waiting queue of node *D* will be transferred to the job queue of node *S*. In this paper, as in the previous literature [Eag86b, Shi92], we assume that T_R = 1. That is, load distribution is attempted only when a node is idle (Livny and Melman [Liv82] call it 'poll when idle' policy). The motivation is that a node is better off avoiding load distribution when there is work to do. Furthermore, several studies have shown that a large percentage (up to 80% depending on time of day) of workstations are idle [Mut87, Nic87, Lit88, The88, Kru91]. Thus the probability of finding an idle workstation is high.

In this policy, if all probes fail to locate a suitable node to get work from, the node could wait for the arrival of a local job. Thus, in this case, job transfers are initiated at most once every time a job is completed. This receiver-initiated policy is used for the results reported in Section 4.

This receiver-initiated policy could cause performance problems because the processing power is wasted until the arrival of a new job locally. This poses severe performance problems if the load is not homogeneous (e.g. if only a few nodes are generating the system load) [Shi92]. As suggested in [Shi92], this adverse impact on performance can be remedied by, for example, reinitiating the load distribution activity after the elapse of a predetermined time if the node is still idle. The effect of reinitiation on the performance of receiver-initiated policies is discussed in Section 5.

2.2. Scheduling Policies

Scheduling strategies can be broadly divided into two classes: *non-preemptive* strategies and *preemptive* strategies. (Note that scheduling strategies can be classified along other dimensions as well. For example, whether the policy requires job characteristics or not leading to *job-independent* and *job-dependent* policies.) We consider two non-preemptive strategies (one job-independent policy and one job-dependent policy) and one preemptive policy.

Non-preemptive Strategies

- *First-Come-First-Serve (FCFS):* This is a non-preemptive (job-independent) scheduling strategy that schedules jobs in the order of their arrival into the job queue. This node scheduling policy has been used in most of the previous studies.
- *Smallest Job First (SJF)*: This strategy takes job service times into consideration in making scheduling decisions. When a node is looking for a job, the node simply schedules itself that job (from the node's job queue) that requires the smallest service time. To implement this policy, *a priori* knowledge about job service times is needed. This requirement causes implementation problems as such knowledge is often not available in practice. However, policies such as this one are studied to understand how much benefit one can obtain if job

service time knowledge is available in advance. The SJF policy reduces the job response times substantially (as compared to the FCFS scheduling policy). This, however, is achieved by giving priority to jobs with smaller service demands. In this sense, this strategy in not fair unlike the FCFS policy. This may lead to starvation, particularly at high system loads. Note, however, our interest here is to use this policy as an yardstick to compare the performance of the other implementable policies. Our results show that, under certain conditions, the performance of the SJF policy (even with accurate *a priori* job service demand knowledge) is worse than that of the round robin policy (discussed next).

Preemptive Policy

- We consider the round robin (RR) preemptive scheduling policy. In the RR policy, each scheduled job is given a quantum Q and, if the job is not completed within that quantum, it is preempted and placed at the end of the associated job queue. This would eliminate a particularly large service demand job from monopolizing the processor. Note that each preemption would cause a context switch overhead *CSO*.

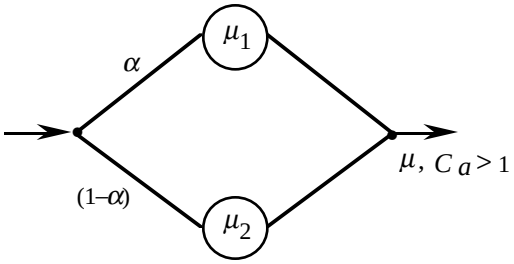
3. System and Workload Models

In the simulation model, a locally distributed system is represented by a collection of nodes. In this paper we consider only homogeneous nodes. We also model the communication network in the system at a higher level. We model communication delays without modelling the low-level protocol details. An Ethernet-like network with 10 Mbits/sec is assumed. The communication network is modelled as a single server. Each node is assumed to have a communication processor that is responsible for handling communication with other nodes. Similar assumptions are made by other researchers [Mir89]. The CPU would give preemptive priority to communication activities (such as reading a message received by the communication processor, initiating the communication processor to send a probe message etc.) over the processing of jobs.

The CPU overheads to send/receive a probe and to transfer a job is modelled by T_{probe} and T_{jx}, respectively. Actual job transmission (handled by the communication processor) time is assumed to be uniformly distributed between U_{jx} and L_{jx}. Probing is assumed to be done serially, not in parallel. For example, the implementation [Dik89] uses serial probing.

The system workload is represented by four parameters. The job arrival process at each node is characterized by a mean inter-arrival time $1/\lambda$ and a coefficient of variation C_a. Jobs are characterized by a processor service demand (with mean $1/\mu$) and a coefficient of variation C_s. We study the performance sensitivity to variance in both inter-arrival times and service times (the CV values are varied from 0 to 4). As in most previous studies, we model only CPU-intensive jobs in this study. (Exceptions do exist, including [Hac88], that consider the disk service demand.)

We use a two-stage hyper-exponential model (shown below) to generate service time and interarrival time CVs greater than 1 [Kob81].



where μ_1 is the mean service time of stage 1, α is the selection probability of this stage, and μ_2 is the mean service time of stage 2. The values of μ_1 and μ_2 can be computed as follows:

$$\mu_1 = \mu - \mu(1-\alpha)\left(\frac{C_a^2 - 2}{2\alpha(1-\alpha)}\right)^{\frac{1}{2}}$$

$$\mu_2 = \mu + \mu(1-\alpha)\left(\frac{C_a^2 - 2}{2\alpha(1-\alpha)}\right)^{\frac{1}{2}}$$

where μ is the mean service time and *Ca* is the desired coefficient of variation. The value of α should satisfy the following relationship:

$$\frac{1+\alpha}{1-\alpha} > C_a^2$$

We have used $\alpha = 0.95$ in the simulation experiments. Note that a specific value of α only affects the absolute values of the results but not the relative performance that is of interest in this paper.

We use the mean response time as the chief

performance metric to compare the performance of the various scheduling policies. In addition, we also obtain the probe rate per node and job transfer rate per node statistics from the simulation. These two rates are computed as follows:

Probe rate/node =

$$\frac{\text{number of probes during simulation}}{(T * \text{number of nodes})}$$

Transfer rate/node =

$$\frac{\text{number of job transfers during simulation}}{(T * \text{number of nodes})}$$

where T is the simulation run length.

4. PERFORMANCE ANALYSIS

This section presents the simulation results and discusses the performance sensitivity of the sender-initiated and receiver-initiated load sharing policies and the impact of scheduling policies on their performance. Unless otherwise specified, the following default parameter values are assumed. The distributed system has N = 32 nodes interconnected by a 10 megabit/second communication network. The average job service time is one time unit (e.g., 1 second). The size of a probe message is 16 bytes. The CPU overhead to send/receive a probe message T_{probe} is 0.003 time units and to transfer a job T_{jx} is 0.02 time units. Job transfer communication overhead is uniformly distributed between L_{jx} = 0.009 and U_{jx} = 0.011 time units (i.e., average job transfer communication overhead is 1% of the average job service time). Since we consider only non-executing jobs for transfer, 1% is a reasonable value. Note that transferring active jobs would incur substantial overhead as the system state would also have to be transferred. Furthermore, it is shown in Section 4.6 that the transfer cost does not change the relative performance of the policies. Transfer policy threshold T_m is 2 and location policy threshold T_S (for the sender-initiated policies) is 2 and T_R (for the receiver-initiated policies) is 1 (as explained in Section 2.1). Similar threshold values are used in previous studies [Eag86a/b, Dik89]. Section 4.5 discusses the impact of this parameter. Probe limit P_l is 3 for all policies. Section 4.4 considers the impact of the probe limit. For the RR policies, quantum size Q is 0.1 time unit and the context switch overhead CSO is 1% of the quantum size Q. In practice, this is a reasonable value. For example, DYNIX uses 100 msec quantum with 0.75 msec reallocation time (i.e., about 0.75%).

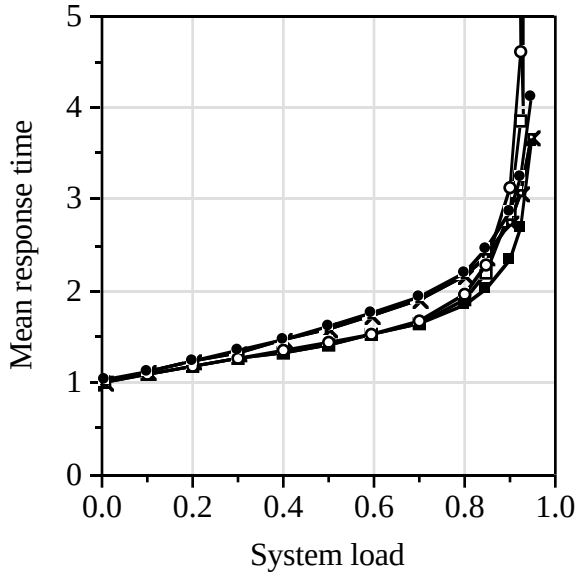
Batch strategy has been used to compute confidence intervals (at least 30 batch runs were used for the results reported here). This strategy produced 95% confidence intervals that were less than 1% of the mean response times when the system utilization is low to moderate and less than 5% for moderate to high system utilization (in most cases, up to about 90%).

Section 4.1 discusses the performance as a function of system load. The sensitivity of the performance to variance in inter-arrival times and service times is presented in Sections 4.2. and 4.3, respectively. Sections 4.4 and 4.5 discuss the impact of probe limit and threshold parameters on the performance. Performance sensitivity to the transfer cost is given in Section 4.6.

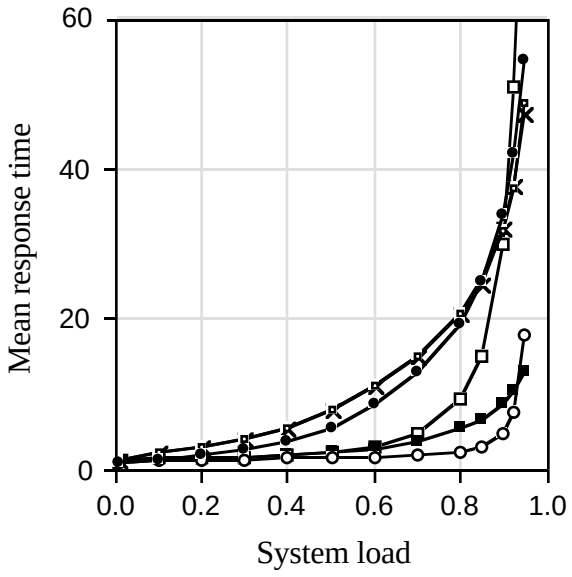
4.1. Performance as a function of system load

Figure 4.1 shows the mean response time of the two load sharing policies under the three node scheduling policies as a function of offered system load. Note that the offered system load is given by λ/μ. Since μ = 1 for all the experiments, offered system load is equal to λ. The results in Figure 4.1a correspond to an inter-arrival CV (C_a) and service time CV (C_s) of 1 and those in Figure 4.1b were obtained with the inter-arrival time CV and service time CV of 4. Note that the y-axis scale in Figure 4.1b is 12 times that used in Figure 4.1a. As can be expected, higher service and inter-arrival CVs have a substantial impact on all policies. When C_a =1 and C_S =1, scheduling policy has very little impact on the receiver-initiated policies. For the sender-initiated policies, performance depends on the scheduling policy only when the system load is high. At these system loads, the SJF policy performs better by giving priority to shorter jobs. Furthermore, when FCFS or RR policies are used, receiver-initiated policy performs better than the sender-initiated policy at high system loads. This observation agrees with the previous results that used the FCFS node scheduling policy [Eag86b, Shi92].

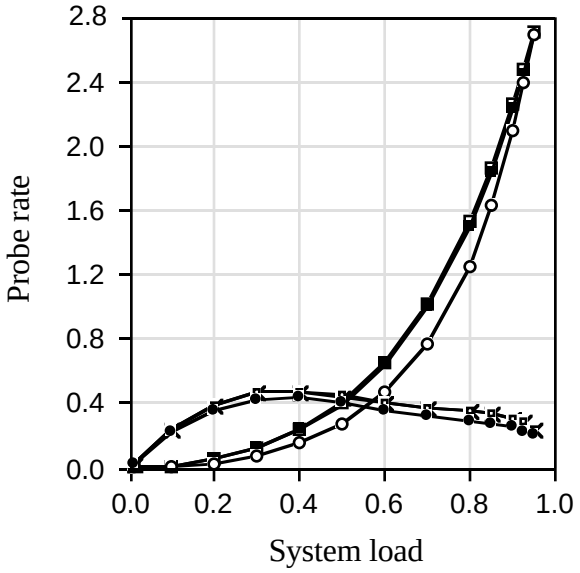
When $C_a = C_S$ = 4, sender-initiated policy performs substantially better than the receiver-initiated policy except at high system loads and when the FCFS node scheduling policy is used. The performance superiority of the sender-initiated policies is mainly due to the high sensitivity of receiver-initiated policies to inter-arrival CV (a detailed explanation can be found in the next section, which discusses the impact of inter-arrival CV on the performance).



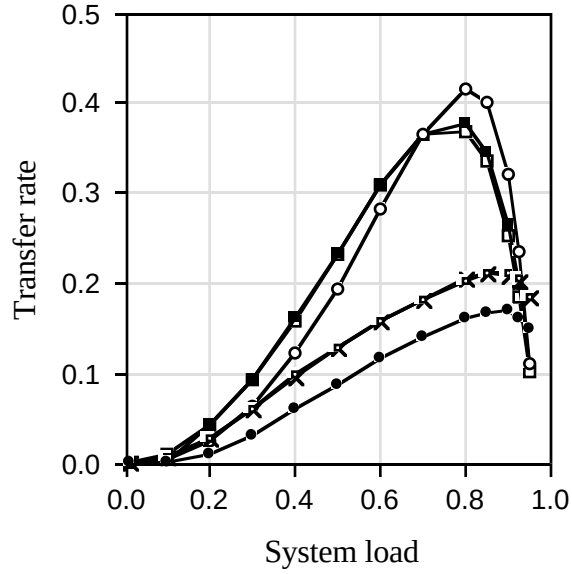
(a) Inter-arrival CV *Ca* = Service CV *Cs* = 1



(b) Inter-arrival CV *Ca* = Service CV *Cs* = 4



(c) Inter-arrival CV *Ca* = Service CV *Cs* = 4



(d) Inter-arrival CV *Ca* = Service CV *Cs* = 4

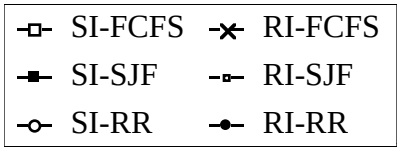


Figure 4.1 Performance sensitivity as a function of offered system load

(N = 32 nodes, μ = 1, λ is varied to vary system load, T_m = 2, T_S = 2, T_R = 1, P_l = 3, transfer cost = 1%, Q = 0.1, CSO = 1% of Q)

Receiver-initiated policies

For the receiver-initiated policy, FCFS and SJF node scheduling policies provide similar performance. This is due to the threshold value (T_m = 2) selected for this experiment. Note that, when T_m=2, SJF node scheduling policy behaves like the FCFS policy because there is only one job in the job queue waiting for the processor. The impact of increasing the threshold value is discussed in Section 4.5.

The RR policy performs slightly better than the FCFS/SJF policies due to the preemptive nature of the RR policy. Even though the RR policy introduces overhead in terms of context switch, it also tends to reduce other system overheads associated with probes and job transfers. As shown in Figures 4.1c and 4.1d, RR policy results in a lower probe rate and lower job transfer rate. This is because RR policy tends to complete small jobs faster and therefore, more jobs can join the job queue as compared to the situation with FCFS/SJF policies. Thus the performance advantage of the RR policy increases with increasing variance service times (further discussed in Section 4.3).

Sender-initiated policies

The node scheduling policy has substantial impact on the sender-initiated policy. In general, RR policy tends to provide the best performance among the three node scheduling policies considered here (followed by SJF and FCFS policies). While the worst performance of the FCFS policy is expected, the performance superiority of RR policy over the SJF policy is surprising. This superiority of the RR policy is partly due to the preemptive nature of the RR policy and partly due to the limited choice for the SJF policy because of the short job queue length. (Note, however, that increasing the threshold value results in decreased number of attempts at load distribution and, therefore, deteriorates performance - see Section 4.5.)

In particular, the RR policy is relatively insensitive to the offered system load for up to about 80%. The SJF policy dominates the RR policy only at very high system loads. For similar reasons as in the receiver-initiated policy, RR policy generates fewer probes than when FCFS/SJF policies are used (Figure 4.1c). For reasons explained before, the job transfer rate of RR policy is lower than that with the FCFS/SJF policies for up to about 70% system load as shown in Figure 4.1d. However, at higher system loads, RR policy transfers more jobs. This is because, at higher system loads, the probability of finding a receiver node is higher with the RR policy because the RR policy tends to complete smaller jobs faster (therefore, the probability of the node queue length falling below the threshold increases with the RR policy).

4.2. Sensitivity to variance in inter-arrival times

This section considers the impact of inter-arrival CV C_a on the performance of the sender-initiated and receiver-initiated policies. Figure 4.2 shows the mean response time, average probe rate and the job transfer rate when the offered system load is fixed at 80% (i.e., λ = 0.8) and service time CV at 1. All other parameter values are set as in Figure 4.1. *The main observation from this figure is that the impact of the variance in inter-arrival times is much more significant on the receiver-initiated policies than on the sender-initiated policies.*

Note that higher inter-arrival CV implies clustered nature of job arrivals at each node in the system; the higher the CV the more clustered the job arrival process is. The sender-initiated policies are relatively less sensitive to the inter-arrival CV. The receiver-initiated policies, on the other hand, exhibit increased sensitivity to Ca. The main reason for this is that the receiver-initiated policies attempt to distribute load only when a job is completed (see Section 5 for the impact of reinitiation). Furthermore, increased CV implies the clustered nature of job arrivals into the system; this decreases the probability of finding a node with no job to execute and hence results in reduced probe rate and job transfer rate. Thus, increasing the inter-arrival CV moves the system based on the receiver-initiated policy towards a no load sharing system. For example, assume that Cs = 0. If Ca is also zero, each node invokes the probing process to locate a sender node after the completion of each job because local job arrival would be some time after the completion of the previous job (because of no variations either in service times or in inter-arrival times). Now increasing C_a means clustered arrival of jobs. Suppose 4 jobs have arrived into the system as a cluster. Then, the node would not initiate any load sharing activity until it completes these four jobs (that is, the node would initiate load sharing only if no job arrives at this node within next $4/\mu$ time units after the arrival of the first job in the cluster). Thus, on average, the probability of finding no job is a decreasing function of Ca (see Figure 4.2b).

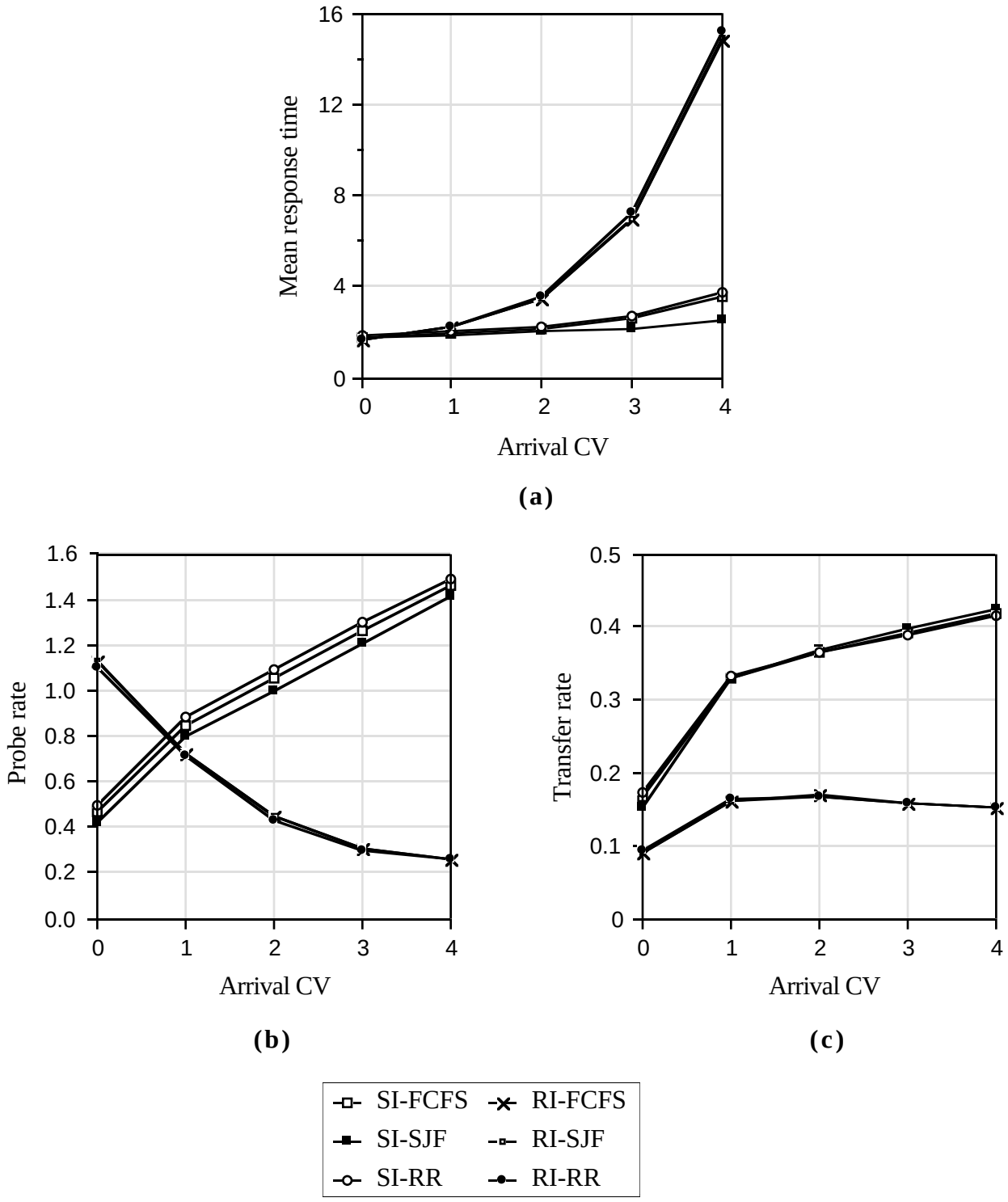


Figure 4.2 Performance sensitivity to inter-arrival time CV

(In these figures, all three lines for the SI and RI policies are close together.)

(N = 32 nodes, λ = 0.8, Ca is varied from 0 to 4, μ = 1, Cs = 1, T_m = 2, T_S = 2, T_R = 1, P_l = 3, transfer cost = 1%, Q = 0.1, CSO = 1% of Q)

On the other hand, clustered arrival of jobs fosters increased load sharing activity in sender-initiated policies. Thus the probe and job transfer rate increase with increasing Ca (see Figures 4.2b and 4.2c). This results in increased overhead costs; however, if these overheads are well controlled, as should be if load sharing is to be effective, this increase in load sharing overhead is well compensated for by the reductions obtained in the mean job response times. Note that these results assume a job transfer cost of 1% of job execution time (i.e., job transfer induces considerable overhead). Note that in sender-initiated policies, when Ca = 4, more than 50% of the jobs are executed remotely (as the transfer rate is more than 0.4 when the job arrival rate λ is 0.8). The corresponding value for the receiver-initiated policy is about 20%. An implication of this is that increasing job transfer costs will have more impact on the sender-initiated policies than on the receiver-initiated policies. Performance sensitivity to transfer cost is discussed in Section 4.6.

For the receiver-initiated policy, scheduling policy does not make any difference. It has been observed in Section 4.1. that SJF and FCFS behave the same way due to the threshold value used in the experiments. If a higher threshold value were used, then SJF would show performance improvements (discussed in Section 4.5). The RR policy performs marginally worse than the FCFS/SJF policies because of low service time CV (which is 1 for this experiments). Note that RR policy improves performance when there is high variance in service times. For the same reason, RR and FCFS perform similarly in sender-initiated policy. However, SJF policy improves performance with increasing Ca. This is because clustered arrival of jobs tends to increase the job queue length beyond 2 (the threshold value used here) because a job that could not be transferred would be moved to the local job queue for processing. Furthermore, any transferred jobs would have to be moved to the local job queue even if that places the node above threshold. Thus, the performance of the SJF improves with increasing variance in inter-arrival times.

4.3. Impact of variance in service times

This section considers the impact of service time CV on the performance of the sender-initiated and receiver-initiated policies. Figure 4.3 shows the mean response time, average probe rate and the job transfer rate when the offered system load is fixed at 80% (i.e., λ = 0.8) and the inter-arrival time CV at 1. All other parameter values are set as in Figure 4.1. Note that, when Cs = 0, FCFS and SJF behave similarly. There are two main observations from this figure:

(i) *When non-preemptive node scheduling policies (FCFS or SJF) are used, sender-initiated policies exhibit more sensitivity to Cs than the receiver-initiated policies.* At high Cs (for example, when Cs = 4) receiver-initiated policies perform substantially better than the sender-initiated policies as shown in Figure 4.3a. The sensitivity of the FCFS policy to service time CV is an expected result because larger jobs can monopolize the processing power that results in increased probe rates (Figure 4.3b). However, this increase in probing activity is useless because, at high system loads, the probability of finding a receiver node is small. As shown in Figure 4.3c, the job transfer rate remains constant independent of the service time CV (for CVs ≥ 1). The SJF performs better than the FCFS by giving priority to smaller jobs. This results in smaller probe rates, even though the job transfer rate remains the same as that in the FCFS policy. For reasons discussed in Section 4.1, the performance of the RI-FCFS and RI-SJF is similar. At high Cs, these receiver-initiated policies perform substantially better than the sender-initiated policies mainly because, at high system loads, the probability of finding an overloaded node is high. Thus, in this case, increased probing activity results in increased job transfers (thereby increasing load sharing). This results in a better performance and results in reduced sensitivity to Cs.

(ii) *When preemptive RR policy is used, sender-initiated policy provides the best performance unless the service time CV is low.* Furthermore, SI-RR policy exhibits negligible sensitivity to service time CV (particularly for CV > 1). This is mainly due to the preemptive nature of the RR scheduling policy. It can be seen from Figure 4.3b that the probe rate associated with SI-RR policy decreases marginally with increasing Cs. This is because higher Cs implies the presence of large number of small jobs and small number of large jobs. Because of its preference to complete smaller jobs, RR policy tends to move more jobs to the corresponding node's local job queue and thus initiates fewer load distribution activities. This tendency increases with increasing Cs. This, however, does not affect the load sharing capability as the job transfer rate remains the same as that in the FCFS/SJF policies (except when Cs < 1). On the other hand, RI-RR policy is sensitive to service time CV. For example, the

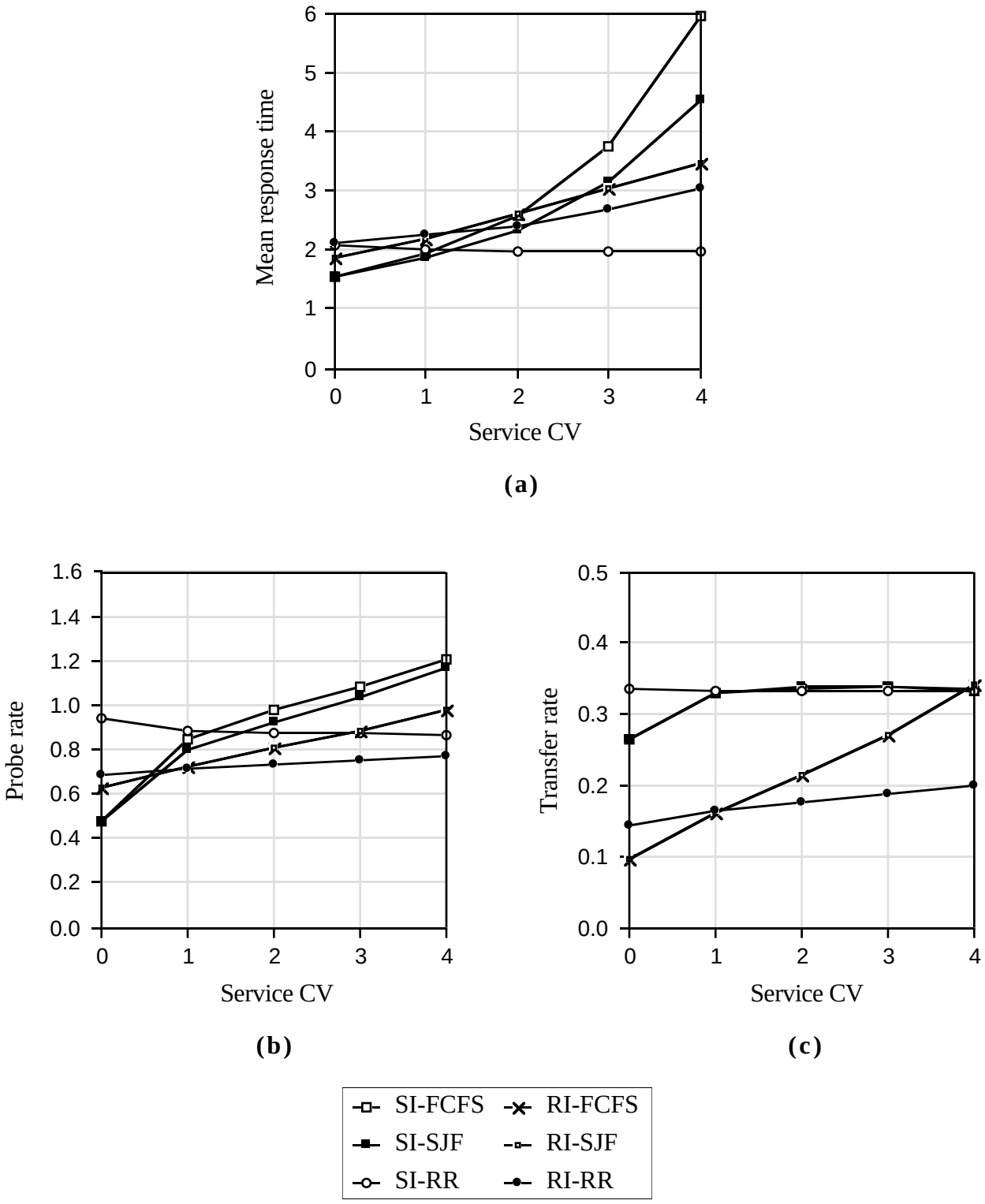


Figure 4.3 Performance sensitivity to service time CV (In (c) SI-FCFS and SI-SJF lines overlap.)
(N = 32 nodes, λ = 0.8, Ca = 1, μ = 1, Cs is varied from 0 to 4, T_m = 2, T_S = 2, T_R = 1, P_l = 3, transfer cost = 1%, Q = 0.1, CSO = 1% of Q)

response time increases from about 2 to 3 as Cs increases from 0 to 4. This can be explained as follows. As indicated above, an increase in Cs results in a large number of small jobs and, therefore, the probability of a node becoming idle increases. This results in an increase in the probe rate as Cs increases (Figure 4.3b), which causes increased job transfers (Figure 4.3c). However, this job transfer rate is substantially less than that in RI-FCFS/SJF policies because the RR policy tends to complete smaller jobs quickly and this reduces the probability of finding a node in the sender state. For example, assume that a node has one large job and one small job. Under FCFS policy, the job queue length would be two until it completes the large job as opposed to in RR policy that completes the smaller by sharing the processing power and thus reduces the job queue length to one after that. Thus the probability of this example node being in the sender state increases when using the FCFS policy. As a result, RI-FCFS/SJF policies tend to transfer more jobs than the RI-RR policy.

Rommel [Rom91] uses a measure called probability of load balancing success (PLBS) and shows that, among other things, the PLBS depends on the service time distribution. The higher the service time CV, the higher the PLBS.

4.4. Performance sensitivity to probe limit

The previous three sections have assumed a probe limit of 3. This section discusses the impact of the probe limit on the performance of the sender-initiated and receiver-initiated policies. Figure 4.4 shows the mean response time, average probe rate and the job transfer rate when the offered system load is fixed at 80%. The service time CV and the inter-arrival time CV are fixed at 4. All other parameter values are set as in Figure 4.1. *A major observation from this figure is that the receiver-initiated policies are more sensitive to probe limit P_l than the sender-initiated policies.*

For the receiver-initiated policies, all three curves are close together. Changing the probe limit does not introduce significant performance differences among the three node scheduling policies under the receiver-initiated policy. The data presented in Figure 4.4a shows that the response time is extremely sensitive to the probe limit. For example, the mean response time decreases from about 45 to about 10 when the probe limit is increased from 1 to 6 (almost 20% of the nodes in the system). Further increase in probe limit to 10 results in a response time of about 6. Even though the probe limit is set at 10, the resulting probe rate is still about 1 (much less than that in the sender-initiated policies even with a small probe limit). For reasons explained in the previous sections, the RI-RR policy generates fewer probes and job transfer rates. As reported elsewhere, at higher system loads and for higher probe limits, receiver-initiated policies perform better than SI-FCFS policy and the performance of the receiver-initiated policies approaches that of SI-SJF policy [Dan95]. However, their performance is still worse than the SI-RR policy.

For the sender-initiated policies, SI-SJF and SI-RR policies exhibit (relatively) less sensitivity to the probe limit. The SI-FCFS policy results in substantial performance degradation if too small a probe limit is used. However, for all these sender-initiated policies, a fairly small probe limit (say 3 or 4) is sufficient. Setting probe limit too high would actually result in marginal performance deterioration because of the increased probe overhead and the uselessness of further probing (see Figures 4.4b and 4.4c). This effect is clearly observable at higher system loads [Dan95]. For reasons explained in the previous sections, the SI-RR policy results in fewer probes but transfers more jobs.

4.5. Performance sensitivity to threshold

The previous experiments have assumed a threshold of $T_m = T_s = 2$ for the sender-initiated policies and $T_m = 2$ and $T_R = 1$ for the receiver-initiated policies. This section discusses the impact of threshold on the performance of the sender-initiated and receiver-initiated policies. Figure 4.5 shows the mean response time, average probe rate and the job transfer rate when the offered system load is fixed at 80%. The service time CV and the inter-arrival time CV are fixed at 4. All other parameter values are set as in Figure 4.1.

Sender-initiated policies

In general, increasing the threshold value decreases the scope of load sharing. This manifests in decreasing probe rates as the threshold is increased. It should be noted that $T_m = T_s = \infty$ corresponds to a system with no load sharing. At $T_m = T_s = 1$, all job arrivals, when the system is busy, would result in initiating load distribution activity but the probability of finding an idle node is very low at high system loads. This, therefore, results in poor performance at high system loads as shown in Figure 4.5.

The SI-FCFS policy is more sensitive to the threshold value than the other two sender-initiated policies. All three sender-initiated policies show perfor-

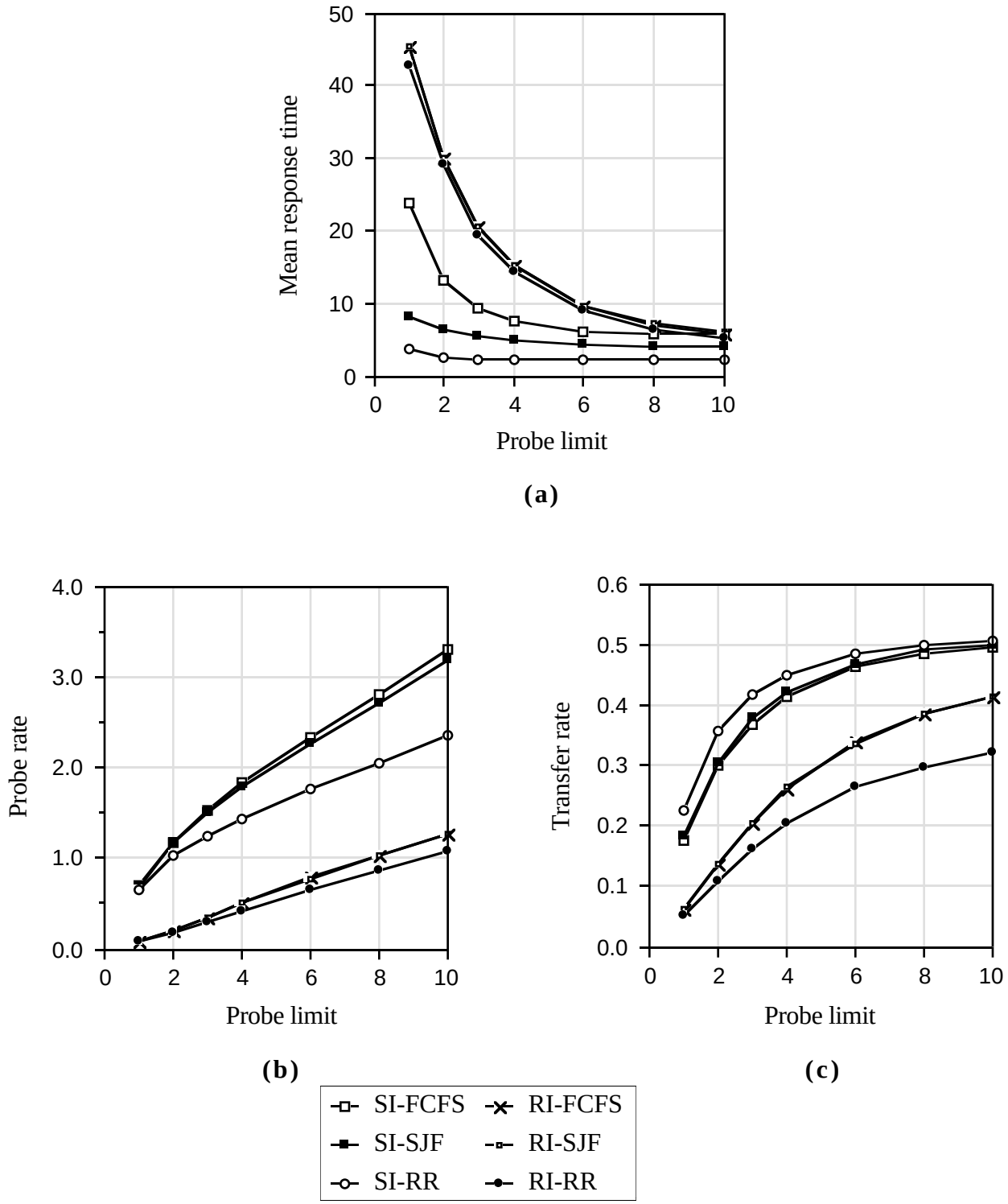


Figure 4.4 Performance sensitivity to probe limit (In (a) all three lines for the RI policies are close together.) (N = 32 nodes, λ = 0.8, Ca = 4, μ = 1, Cs = 4, T_m = 2, T_S = 2, T_R = 1, P_l is varied from 1 to 10, transfer cost = 1%, Q = 0.1, CSO = 1% of Q)

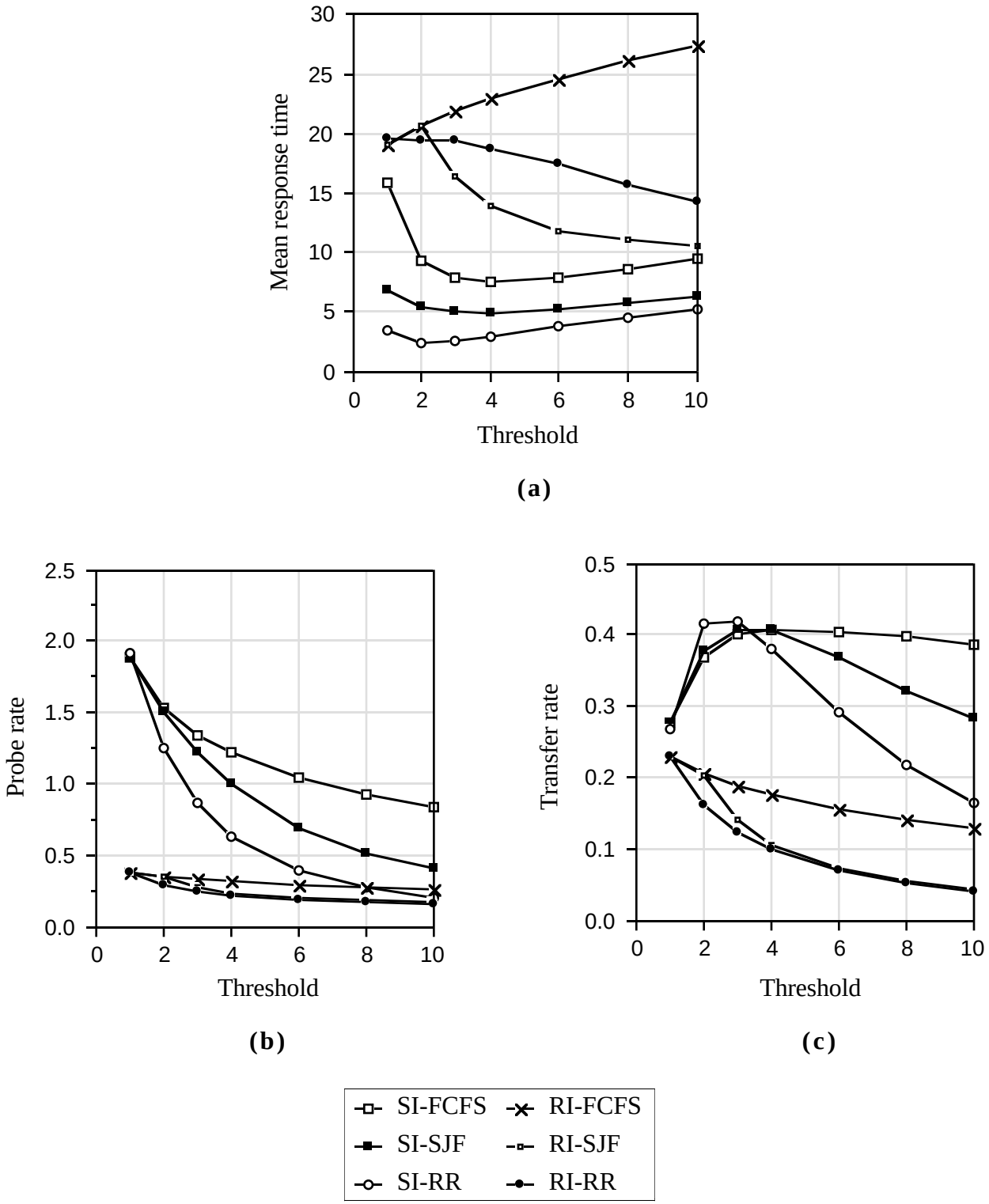


Figure 4.5 Performance sensitivity to the threshold parameter

(N = 32 nodes, λ = 0.8, Ca = 4, μ = 1, Cs = 4, T_m and T_S are varied, T_R = 1, P_l =3, transfer cost = 1%, Q = 0.1, CSO = 1% of Q)

mance improvements initially and further increase in threshold would result in performance deterioration as shown in Figure 4.5a. The reason for the initial performance improvement is that increasing the threshold from 1 increases the probability of finding a node that is below the threshold and therefore results in increased job transfer rate even though the probe rate is reduced (Figures 4.5b and 4.5c). Further increase in the threshold value results in decreasing the number of attempts at load distribution. This causes reduced probe rates as well as reduced job transfer rates. The effect is much more pronounced in SI-RR and SI-SJF policies than in the SI-FCFS policy (Figures 4.5b and 4.5c). The SI-SJF policy generates substantially fewer probes than does the SI-FCFS policy as the threshold increases because there will be more jobs in the job queue to select a smaller job. This causes more jobs to be moved into the local job queue and the probability of finding a node in the sender state decreases. The combined effect of these two phenomena results in fewer probes and fewer job transfers. The SI-RR policy further reduces the probe rate for similar reasons except that the impact would be much greater because of the preemptive nature of the RR policy.

Note that, when $T_m = T_s = 1$, the performance of the three sender-initiated policies is significantly different. The main reason for this behaviour is that, even though the threshold is 1, the job queue length could be higher than 1 due to (i) the transfer of local jobs to the local job queue because no receiver node has been found and (ii) the transfer of jobs that have been transferred from another node. Note that the policies dictate that a transferred job has to be processed at the destination node regardless of the state of the node at the time of its arrival.

Receiver-initiated policies

The performance of the RI-FCFS policy deteriorates with increasing threshold value *Tm*. The main reason for this is that the system moves towards diminished load sharing system and the service time CV is high. It is well-known that the FCFS scheduling discipline is sensitive to the service time CV. The behaviour of RI-SJF is similar to that of the RI-FCFS policy for up to $T_m = 2$. For higher threshold values, RI-SJF provides performance improvements by giving priority to smaller jobs. (Note that the service time CV for this experiment is 4.) RI-RR policy also shows improvements in performance with increasing threshold value as there is increased scope for giving priority to smaller jobs. As can be expected, the improvement with RI-RR is not as substantial as that obtained with the RI-SJF policy.

Rommel [Rom91] has shown that increasing the number of nodes greatly reduces the effect of threshold value.

4.6. Performance Sensitivity to transfer cost

This section presents the impact of the job transfer cost. Previous sections assumed a cost of 1% of the job service demand. This is reasonable as load sharing is often used only for long running jobs (e.g., simulation programs). Figure 4.6 shows the effect of job transfer cost. The x-axis is the transfer cost expressed as a percentage of the job service time. It can be seen from this figure that the transfer cost does not alter the relative performance of the policies under consideration. Furthermore, lower transfer costs have only a negligible influence on the response times. As the transfer cost approaches 10% of the job service demand, response times increase. The increase is much more pronounced on the sender-initiated policies than on the receiver-initiated policies. This is mainly due to communication network saturation as explained below.

Recall that the communication network is modelled as a single server. The utilization of the communication server due to job transfers is given by *N*Tx*Tcost*, where *Tx* is the transfer rate and *Tcost* is the transfer cost. For the 32-node system considered here, the network saturates at about when the *Tx* is 7.8% of job service demand for the SI policies, which have a transfer rate of about 0.4. Thus the performance of the sender-initiated policies deteriorates as the transfer cost approaches 10%. As a result of the network congestion, the probe rate and the transfer rate drop substantially at 10% transfer cost. The effect is not as severe as the transfer rate of receiver-initiated policies is about half of that of the sender-initiated policies.

Mirchandaney et al. [Mir89] have presented the effects of delays in transferring jobs. Using exponential service times of jobs and Poisson job arrivals at the nodes they have shown that the performance of the sender-initiated and receiver-initiated policies is virtually identical at moderately high delays (e.g., at job transfer delay that is twice the job service time) and under FCFS node scheduling policy. However, such long delays are unrealistic unless very small jobs are made eligible for transfer.

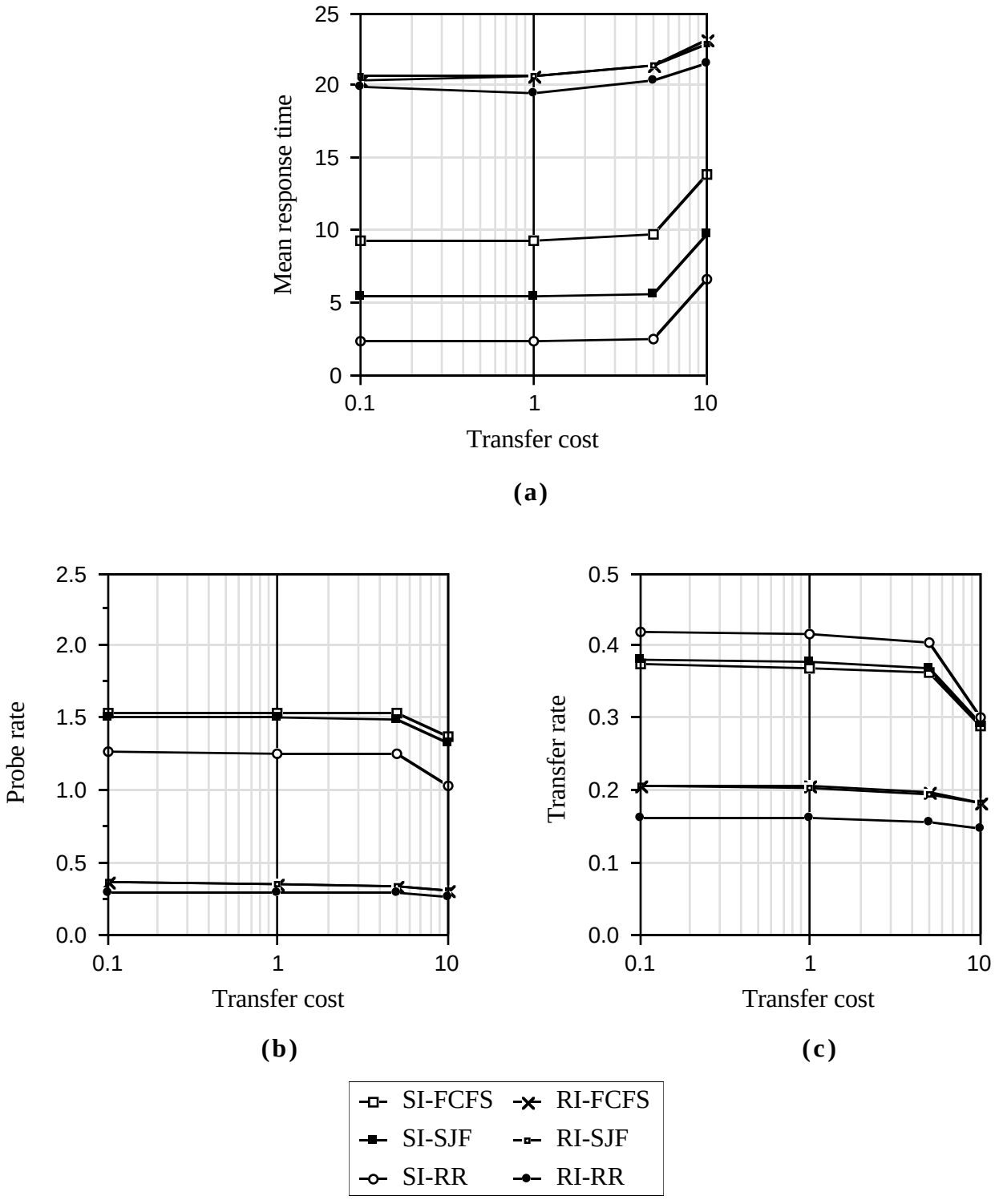


Figure 4.6 Performance sensitivity to transfer cost

(N = 32 nodes, λ = 0.8, Ca = 4, μ = 1, Cs = 4, T_m = 2, T_S = 2, T_R = 1, P_l =3, Q = 0.1, CSO = 1% of Q)

5. EFFECT OF REINITIATION ON RECEIVER-INITIATED POLICIES

The receiver-initiated policies considered so far initiate load distribution activity whenever a node becomes idle as a result of a job completion. (Recall that our receiver-initiated policies have assumed a threshold value of 1.) As a result, the receiver-initiated policies exhibit high sensitivity to inter-arrival times (see Section 4.2). This sensitivity can be reduced by reinitiating load distribution if a node is idle for an extended period of time. This section studies the impact of such reinitiation strategy.

5.1. Sensitivity to reinitiation period

Figure 5.1 shows the response time, probe and job transfer rates as a function of load distribution reinitiation period. The offered system load is fixed at 80% and the service time and the inter-arrival time CVs are fixed at 4. All other parameter values are set as in Figure 4.1. The reinitiation period is varied from 0.5 to 6.

As can be expected, the response time increases with the reinitiation period (see Figure 5.1a). For example, when the reinitiation period is increased from 0.5 to 6, the response time increases from about 4 to 11.5 for the RR scheduling policy. For the FCFS policy, the corresponding values are about 5 and 12.5. We have not shown the data for the SJF scheduling policy as the performance of this policy is similar to that of the FCFS policy.

The high sensitivity to reinitiation period is due to high variance in inter-arrival and service times. In particular, the response time is much more sensitive to inter-arrival time CV than the service time CV as discussed in the next two sub-sections.

It can also be seen from the data presented in Figure 5.1 that the RR scheduling policy maintains its performance superiority over the FCFS policy for the range of reinitiation periods considered here. This performance superiority of the RR policy can be attributed to the fact that the service time CV for this data is 4. For reasons discussed in Section 4, the RR scheduling policy generates fewer probes (Figure 5.1.b) and causes fewer job transfers (Figure 5.1c).

5.2. Sensitivity to variance in inter-arrival times

This section presents the effect of variance in inter-arrival times on the performance of receiver-initiated policies when the reinitiation period is fixed at 1, which is the average job service demand. Figure 5.2 shows the response time, probe and job transfer rates as a function of inter-arrival CV. As in Section 4.2, the service time CV is fixed at 1 and the offered system load at 80%.

For reference, we have also presented the data when load distribution is not reinitiated (as in Section 4). From Figure 5.2a, it can be seen that reinitiation has substantial impact on the response time when the inter-arrival CV is high. For example, when the inter-arrival CV is 4, the response time reduces from about 15.5 to 4.6 when reinitiation is used. However, these values are still higher than those obtained for the sender-initiated policies as shown in Figure 4.2.

Note that when reinitiation is not used, for reasons discussed in Section 4.2, the probe rate decreases rapidly with increasing inter-arrival CV (see Figure 5.2b). When reinitiation is used, the probe rate settles to 1 as we have used a reinitiation period of 1 for this data. This increased probe rate results in increases job transfers as shown in Figure 5.2c.

The data presented in Figure 5.2 show that the node scheduling policy does not have any effect on the performance of the receiver-initiated policies, whether or not reinitiation is employed.

5.3. Sensitivity to variance in service times

This section presents the effect of variance in service times on the performance of receiver-initiated policies when the reinitiation period is fixed at 1 as in the last section. Figure 5.3 shows the response time, probe and job transfer rates as a function of service time CV. As in Section 4.3, the inter-arrival time CV is fixed at 1 and the offered system load at 80%.

The data presented in Figure 5.3 suggest that reinitiation of load distribution has relatively less significant effect on response times when service time CV is varied (as compared to the variance in inter-arrival times). For example, when the service time CV is 4 and the node scheduling policy is RR, the response time decreases from about 3 to 2.4 (about 20% improvement) when reinitiation is employed. The corresponding values for the FCFS policy are 3.5 and 2.8 (again about 20% improvement).

It can also be seen from Figure 5.3 that reinitiation affects the performance of the scheduling policies similarly. It follows from this statement that the RR scheduling policy maintains its performance superiority over the FCFS policy for higher service time CVs, whether or not reinitiation is employed.

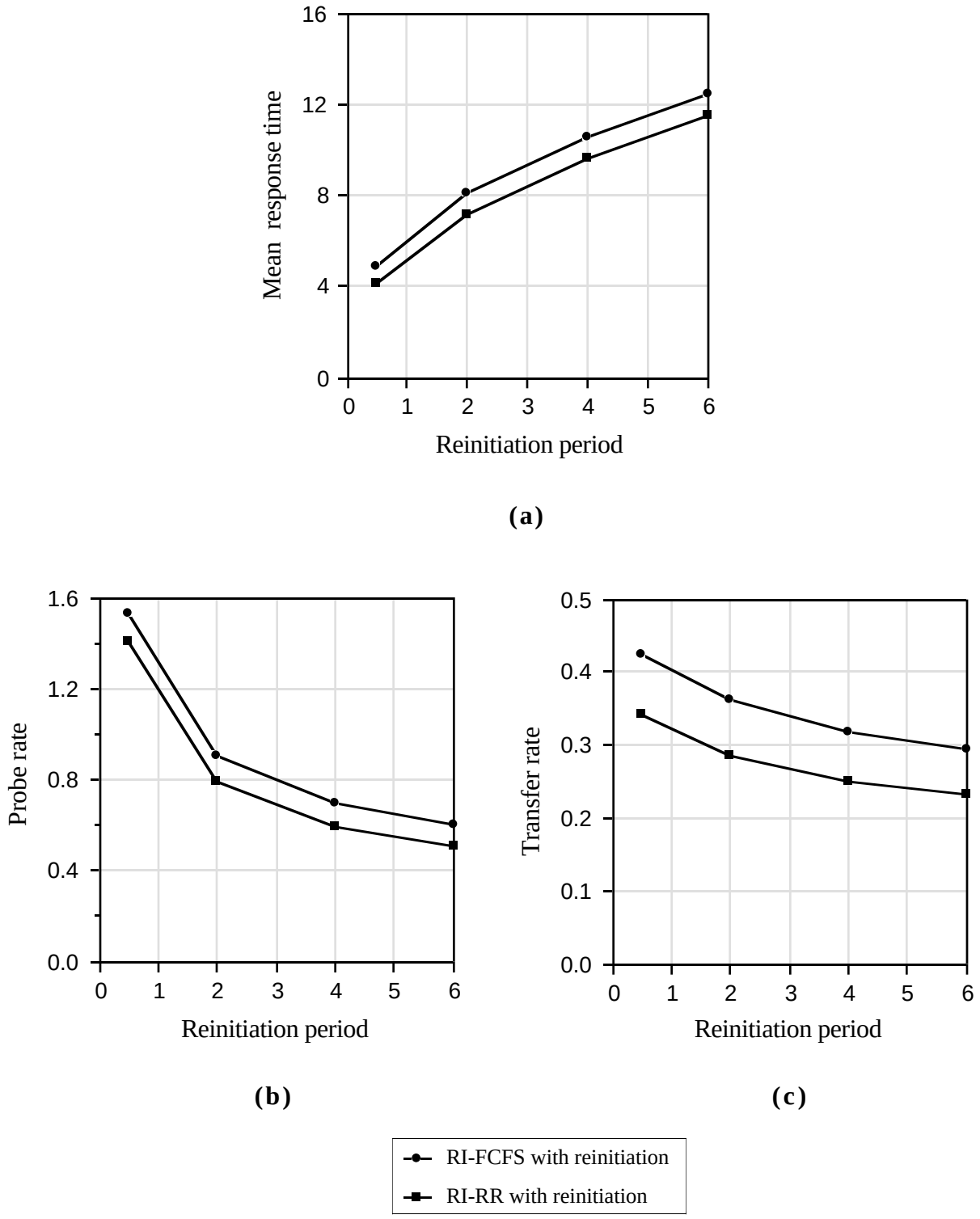


Figure 5.1 Performance sensitivity of the receiver-initiated policies to the reinitiation period (N = 32 nodes, λ = 0.8, Ca = 4, μ = 1, Cs = 4, T_m = 2, T_S = 2, T_R = 1, P_l =3, Q = 0.1, CSO = 1% of Q)

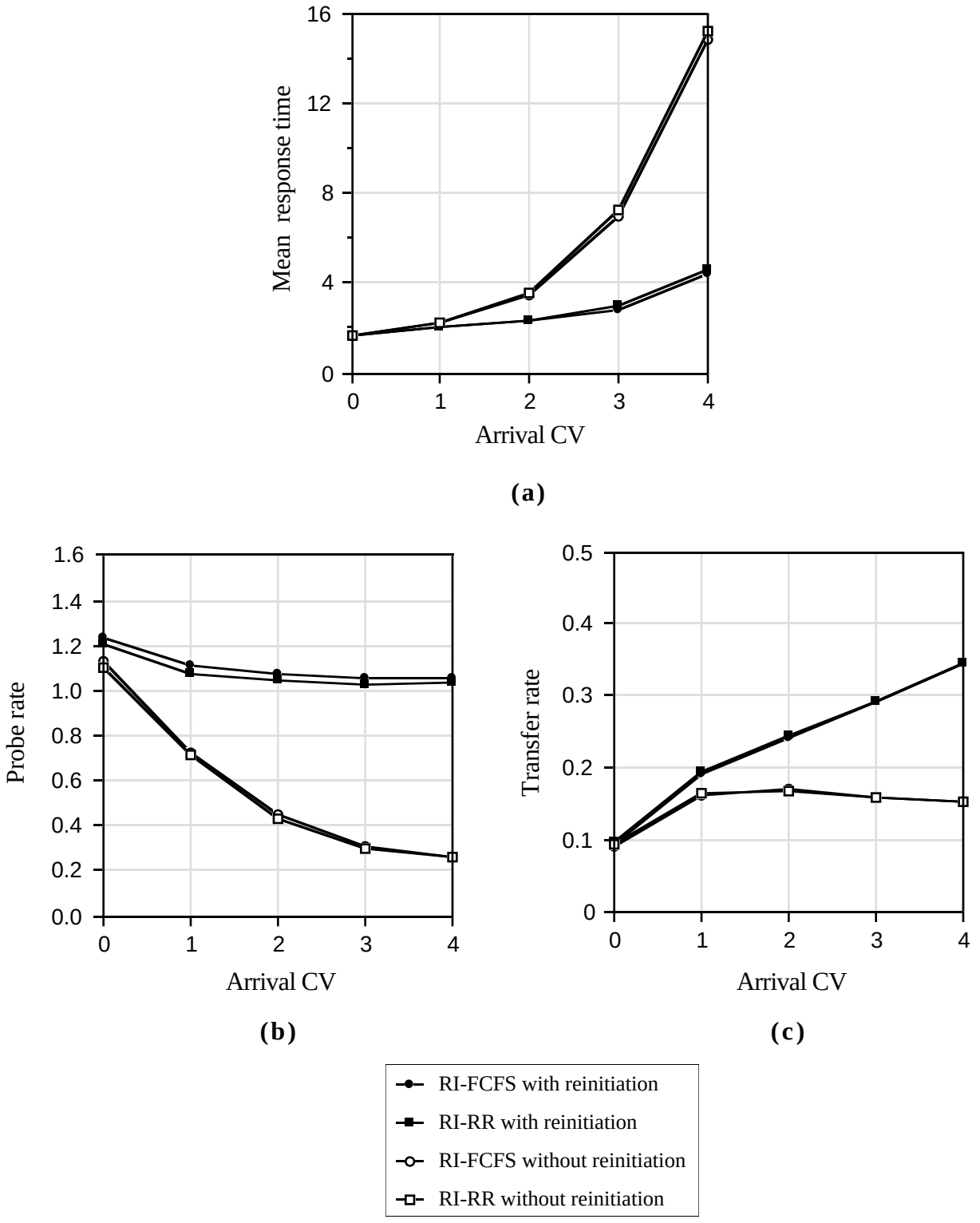


Figure 5.2 Performance sensitivity to variance in inter-arrival times
(N = 32 nodes, λ = 0.8, Ca = varied, μ = 1, Cs = 1, T_m = 2, T_S = 2, T_R = 1, P_l =3, Q = 0.1, CSO = 1% of Q, reinitiation period = 1)

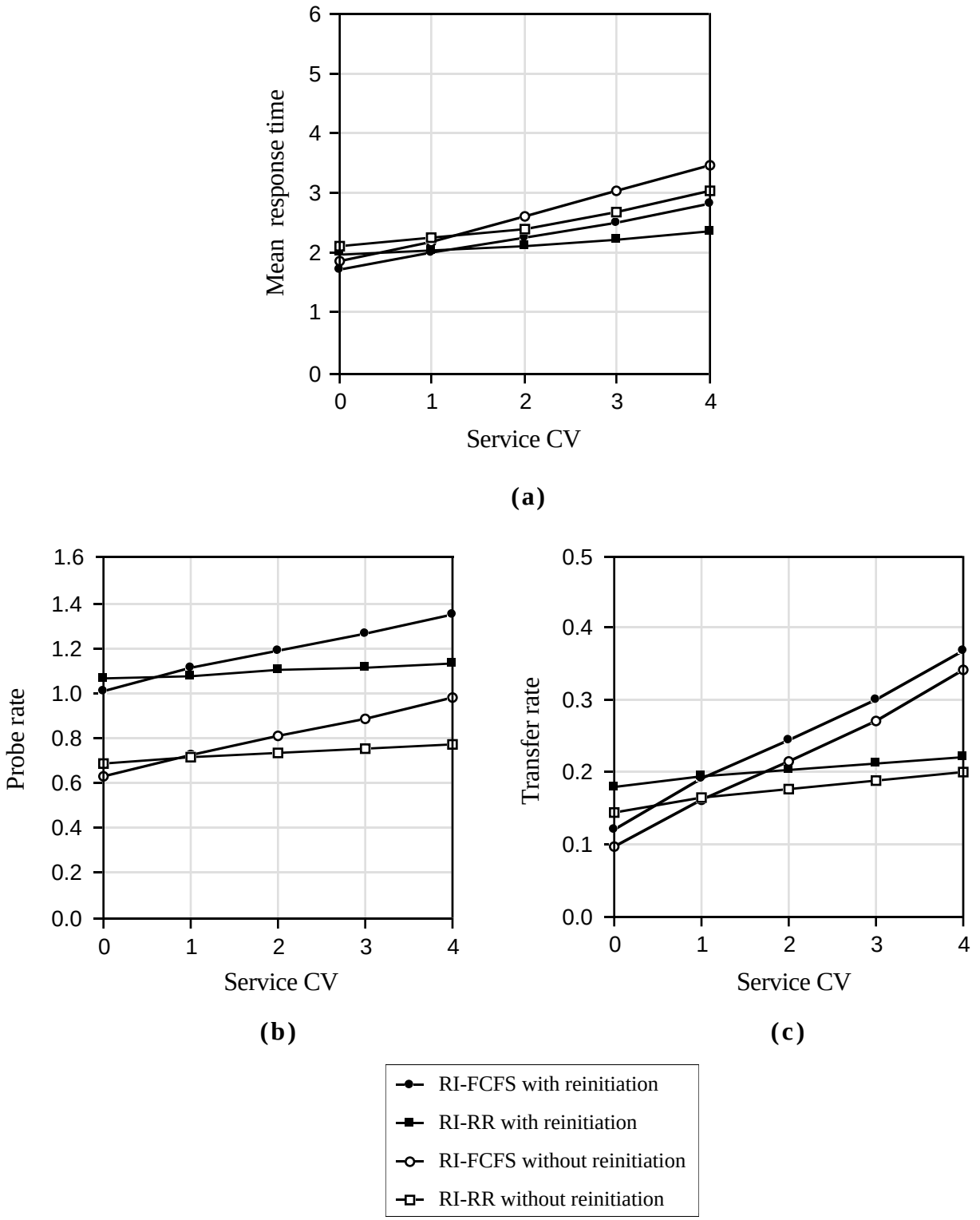


Figure 5.3 Performance sensitivity to variance in service times
(N = 32 nodes, λ = 0.8, Ca = 1, μ = 1, Cs = varied, T_m = 2, T_S = 2, T_R = 1, P_l =3, Q = 0.1, CSO = 1% of Q, reinitiation period = 1)

6. CONCLUSIONS

Load sharing is a technique to improve the performance of distributed systems by distributing the system workload from heavily loaded nodes to lightly loaded nodes in the system. Previous studies have considered two adaptive load sharing policies: sender-initiated and receiver-initiated. In the sender-initiated policy, a heavily loaded node attempts to transfer work to a lightly loaded node and in the receiver-initiated policy a lightly loaded node attempts to get work from a heavily loaded node.

A common thread among the previous studies is the assumption of the FCFS node scheduling policy. Furthermore, analyses and simulations in these studies have been done under the assumption that the job service times are exponentially distributed and the the job arrivals form a Poisson process (i.e., job inter-arrival times are exponentially distributed). Experimental data suggest that service time variance can be mush higher than that of the exponential distribution. The goal of this paper is to fill the void in the existing literature. As a result, this paper has considered the impact of node scheduling policies on the performance of the sender-initiated and receiver-initiated adaptive load sharing policies. We have considered three node scheduling policies - first-come/first-serve (FCFS), shortest job first (SJF), and round robin (RR) policies. Furthermore, we have also studied the impact of variance in the inter-arrival times and in the job service times. In comparing the performance of the sender-initiated and receiver-initiated policies using these three node scheduling policies, we have shown that (for the parameter values considered here):

When non-preemptive node scheduling policies (FCFS and SJF) are used:

- receiver-initiated policies are highly sensitive to variance in the inter-arrival times (i.e., to clustered arrival of jobs) and this sensitivity can be reduced considerably by using reinitiation;
- sender-initiated policies are relatively more sensitive to variance in job service times.

When the preemptive node scheduling policy (RR) is used:

- the sender-initiated policy is insensitive to the system load, variance in the job service times for a given variance in the inter-arrival times;
- the receiver-initiated policy, on the other hand, is sensitive to the system load, variance in service time and the variance in inter-arrival time;
- when the variance in service time is high, the sender-initiated policy provides the best performance among the policies considered in this study.

We have not considered the impact of system and workload heterogeneity [Mir90]. System heterogeneity refers to non-homogeneous nodes (nodes with different processing speeds, for example) and the workload heterogeneity refers to non-homogeneous job characteristics. We intend to extend our work to study these issues in the near future.

ACKNOWLEDGEMENTS

I thank Siu-Lun Au for his help with the simulation program. I gratefully acknowledge the financial support provided by the Natural Sciences and Engineering Research Council of Canada and Carleton University.

REFERENCES

[Alo88] R. Alonso and L. L. Cova, “Sharing Jobs Among Independently Owned Processors,” *IEEE Int. Conf. Dist. Computing Systems,* 1988, pp. 282-288.

[Cho90] S. Chowdhury, “The Greedy Load Sharing Algorithm,” *J. Parallel and Distributed Computing,* Vol. 9, 1990, pp. 93-99.

[Dan85] S. P. Dandamudi, “Performance Impact of Scheduling Discipline on Adaptive Load Sharing in Homogeneous Distributed Systems,” *IEEE Int. Conf. Dist. Computing Systems,* 1995.

[Dik89] P. Dikshit, S. K. Tripathi, and P. Jalote, “SAHAYOG: A Test Bed for Evaluating Dynamic Load-Sharing Policies,” *Software - Practice and Experience*, Vol. 19, No. 5, May 1989, pp. 411-435.

[Eag86a] D. L. Eager, E. D. Lazowska, and J. Zahorjan, "Adaptive Load Sharing in Homogeneous Distributed Systems," *IEEE Trans. Software Engng.*, Vol. SE-12, No. 5, May 1986, pp. 662-675.

[Eag86b] D. L. Eager, E. D. Lazowska, and J. Zahorjan, "A Comparison of Receiver-Initiated and Sender-Initiated Adaptive Load Sharing," *Performance Evaluation*,

Vol. 6, March 1986, pp. 53-68.

[Eag88] D. L. Eager, E. D. Lazowska, and J. Zahorjan, "The Limited Performance Benefits of Migrating Active Processes for Load Sharing," *ACM Sigmetrics Conf.*, 1988, pp. 63-72.

[Hac87] A. Hac and X. Jin, "Dynamic Load Balancing in a Distributed System Using a Decentralized Algorithm," *IEEE Int. Conf. Dist. Computing Systems*, 1987, pp. 170-177.

[Hac88] A. Hac and T. J. Johnson, "Dynamic Load Balancing Through Process and Read-Site Placement in a Distributed System," *AT&T Technical Journal*, 1988, pp. 72-85.

[Har90] A. J. Harget and I. D. Johnson, "Load Balancing Algorithms in Loosely-Coupled Distributed Systems: A Survey," in *Distributed Computing Systems*, (Ed) H. S. M. Zedan, Butterworths, London, 1990, pp. 85-108.

[Kob81] H. Kobayashi, *Modeling and Analysis: An Introduction to System Performance Evaluation Methodology*, Addison-Wesley, Reading 1981.

[Kru88] P. Krueger and M. Livny, "A Comparison of Preemptive and Non-Preemptive Load Distributing," *IEEE Int. Conf. Dist. Computing Systems*, 1988, pp. 123-130.

[Kru91] P. Krueger and R. Chawla, "The Stealth Distributed Scheduler," *IEEE Int. Conf. Dist. Computing Systems*, 1991, pp. 336-343.

[Kun91] T. Kunz, "The Influence of Different Workload Descriptions on a Heuristic Load Balancing Scheme," *IEEE Trans. Software Engng.*, Vol. SE-17, No. 7, July 1991, pp. 725-730.

[Lel86] W. E. Leland and T. J. Ott, "Load Balancing Heuristics and Process Behavior," *Proc. PERFORMANCE 86* and *ACM SIGMETRICS 86*, 1986, pp. 54-69.

[Lin92] H.-C. Lin and C. S. Raghavendra, "A Dynamic-Load Balancing Policy With a Centralized Dispatcher (LBC)," *IEEE Trans. Software Engng.*, Vol. SE-18, No. 2, February 1992, pp. 148-158.

[Lit88] M. J. Litzkow, M. Livny, and M. W. Mutka, "Condor - A Hunter of Idle Workstations," *IEEE Int. Conf. Dist. Computing Systems*, 1988, pp. 104-111.

[Liv82] M. Livny and M. Melman, "Load Balancing in Homogeneous Broadcast Distributed Systems, *Proc. ACM Computer Network Performance Symp.*, 1982, pp. 47-55.

[Mir89] R. Mirchandaney, D. Towsley, J. A. Stankovic, "Analysis of the Effects of Delays on Load Sharing," *IEEE Trans. Computers.*, Vol. 38, No. 11, November 1989, pp. 1513-1525.

[Mir90] R. Mirchandaney, D, Towsley, and J. A. Stankovic, "Adaptive Load Sharing in Heterogeneous Distributed Systems," *J. Parallel and Distributed Computing*, Vol. 9, 1990, pp. 331-346.

[Mut87] M. Mutka and M. Livny, "Prtofiling Workstation's Available Capacity for remote Execution," *Proc Performance 87*, Brussels, Belgium, 1987, pp. 529-544.

[Nic87] D. Nichols, "Using Idle Workstations in a Shared Computing Environment," *Proc. ACM Symp. Operating System Principles*, Austin, Teas, 1987, pp. 5-12.

[Rom91] G. Rommel, "The Probability of Load Balancing Success in a Homogeneous Network," *IEEE Trans. Software Engng.*, Vol. SE-17, No. 9, September 1991, pp. 922-933.

[Sch91] M. Schaar, K. Efe, L. Delcambre, L. N. Bhuyan, "Load Balancing with Network Cooperation," *IEEE Int. Conf. Dist. Computing Systems*, 1991, pp. 328-335.

[Shi90] N. G. Shivaratri and P. Krueger, "Two Adaptive Location Policies for Global Scheduling Algorithms," *IEEE Int. Conf. Dist. Computing Systems*, 1990, pp. 502-509.

[Shi92] N. G. Shivaratri, P. Krueger, and M. Singhal, "Load Distributing for Locally Distributed Systems," *IEEE Computer*, December 1992, pp. 33-44.

[Sue92] T. T. Y. Suen and J. S. K. Wong, "Efficient Task Migration Algorithm for

Distributed Systems, *IEEE Trans. Parallel and Dist. Syst.*, Vol. 3, No. 4, July 1992, pp. 488-499.

[The88] M.M. Theimer and K. A. Lantz, "Finding Idle Machines in a Workstation-Based Distributed System," *IEEE Int. Conf. Dist. Computing Systems*, 1988, pp. 112-122.

[Wan85] Y. T. Wang and R. J. T. Morris, "Load Sharing in Distributed Systems," *IEEE Trans, Computers*, Vol. C-34, March 1985, pp. 204-217.

[Zho88] S. Zhou, "A Trace-Driven Simulation Study of Dynamic Load Balancing," *IEEE Trans. Software Engng.*, Vol. SE-14, No. 9, September 1988, pp. 1327-1341.

[Zho93] S. Zhou, X. Zheng, J. Wang, and P. Delisle, "Utopia: a Load Sharing Facility for Large, Heterogeneous Distributed Computer Systems," *Software - Practice and Experience*, Vol. 23, No. 12, December 1993, pp. 1305-1336.