IIE Transactions (2001) **33**, 203–217

Efficient distributed simulation through dynamic load balancing

MURALI S. SHANKER[1], REMA PADMAN[2] and W. DAVID KELTON[3]

[1]*Department of Management and Information Systems, Kent State University, Kent, OH 44242, USA*
E-mail: mshanker@kent.edu
[2]*H. John Heinz III School of Public Policy and Management, Carnegie Mellon University, Pittsburgh, PA 15213, USA*
E-mail: rpadman@andrew.cmu.edu
[3]*Department of Quantitative Analysis and Operations Management, College of Business Administration, University of Cincinnati, Cincinnati, OH 45221-0130, USA*
E-mail: david.kelton@uc.edu

Received December 1995 and accepted July 2000

With recent advances in parallel computation, distributed simulation has become a viable way of dealing with time-consuming simulations. For distributed simulations to run efficiently, care must be taken in assigning the tasks (work) in the simulated system to the available physical processors in the computer system. An inefficient assignment can result in excessive communication times between processors and unfavorable load conditions. This leads to long run times, possibly giving performance worse than that with a uniprocessor sequential event-list implementation. This paper establishes the feasibility, and in some cases the necessity, of using dynamic task allocation (rather than *a-priori* static allocation) in distributed simulation. A dynamic reallocation strategy is developed, and experiments on an iPSC/2 Hypercube indicate that significant improvements in run time can be achieved at low cost.

1. Introduction

With recent advances in parallel computation, Distributed Simulation (DS) has become a viable way of dealing with time-consuming simulations. For example, consider a simulation of large circuit-switched communication networks that consist of nodes and links between node pairs. A link consists of a number of trunks, each having the capacity needed to carry exactly one call. To place a call between a pair of switches, a path in the network is identified, and on each link in the path, a trunk is allocated for the sole use of the call. These trunks are then held for the duration of the call. During periods of high load, some calls may be blocked since the number of trunks on each link is limited.

Designers of circuit-switched networks are concerned about how different factors like network routing and traffic patterns affect call blocking. A typical requirement is that call blocking not exceed a prespecified threshold, say, 0.01%, except during emergencies. Simulations of such networks are time-consuming because the number of simulated events per unit of real time is large, especially in emergency situations. For example, a useful simulation of the AT&T network would involve at least on the order of 10^9 events (Eick *et al.*, 1993). Fortunately, such simulations offer great potential for speedup using DS.

In DS, unlike sequential event-list simulation, there is no global event list or global simulation clock. Here the physical system to be simulated is represented by a collection of Logical Processes (LPs) that communicate via timestamped messages along directed channels in the logical system. For example, the logical system of Fig. 1 could represent a work center with 10 machines, with each LP modeling a machine. The timestamped messages in DS represent events in a sequential simulation. In Fig. 1, timestamped messages originate from LP_1 (the source), and leave the system from LP_{10} (the sink). At the branch point LP_3, messages go to LP_4 and LP_8 with equal probabilities.

An important factor affecting the performance of DS models, like the Chandy and Misra model for DS (Chandy and Misra, 1979, 1981; Misra, 1986), is the allocation of LPs to available (physical) processors. The primary objective in making an assignment is to reduce the realized run time of the simulation. An inefficient assignment can result in excessive communication times between processors and unfavorable load conditions. This leads to long run times, possibly giving performance worse than that with a uniprocessor sequential event-list implementation.

The problem of assigning LPs to processors to minimize the run of the simulation is one instance of the

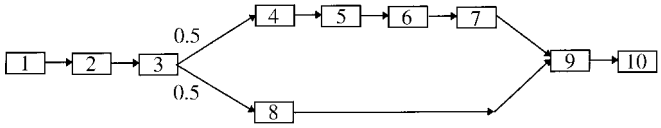

Fig. 1. A logical system with 10 LPs.

Task-Allocation Problem (TAP) found in distributed systems, and it is NP-complete (Garey and Johnson, 1979). Similar problems exist in other fields. For example, a shop floor in a manufacturing environment can be viewed as a group of work centers. Each work center is composed of manufacturing cells containing machines, robots, and tools. Activity planning is a complex problem in which activities (jobs) must be selected, tools must be assigned to machines, and resources (machines) must be allocated and scheduled to meet production goals. With the introduction of robotics and other flexible technologies in manufacturing, the number of ways in which a product can be manufactured has increased. This in turn introduces additional complexity in activity planning, as more options have to be considered to determine the best schedule. This activity planning is a TAP, where an objective may be to maximize the throughput of jobs (Bourne and Fox, 1984). An analogy to the current research is to view jobs as messages, tools as LPs, and machines and robots as processors. Our research in distributed simulation task allocation should provide insight for analysis and management of such industrial systems.

Various *static* and *dynamic load-sharing* strategies have been proposed for TAPs (Stone, 1977; Bokhari, 1979; Livny and Melman, 1982; Ni and Hwang, 1985; Nicol and Reynolds, 1985; Tantawi and Towsley, 1985; Eager *et al.*, 1986; Iqbal *et al.*, 1986; Lu and Carey, 1986; Chu and Lan, 1987; Baumgartner and Wah, 1989; Shin and Chang, 1989; Nandy and Loucks, 1992; Goswami *et al.*, 1993; Woodside and Monforton, 1993), but their potential is limited for Distributed Simulation Task-Allocation (DSTA) problems (Shanker, 1990; Shanker *et al.*, 1993). The results of previous studies on solving TAPs in distributed systems indicate that state-dependent (dynamic) load-sharing strategies perform better than static strategies, but have higher overhead and are sensitive to inadequate or inaccurate status information. While dynamic strategies are more appropriate for task allocation in DS, existing schemes tend to assume knowledge of many parameters that in practice are not known. Unlike previous studies where strategies are set up for externally generated messages (tasks), in DS the rate of message generation depends internally on the strategy in effect (Shanker, 1990; Shanker *et al.*, 1993). For example, the arrival rate of messages for the logical system of Fig. 1 depends on the processor to which LP_1 is assigned, so the overall load in the system can be affected by changing this assignment. Other complicating factors in DS include unknown precedence relationships, reallocation of LPs rather than individual messages, and the fact that relatively slow inter-processor communication times may imply that a balanced load across processors may not be the proper strategy if the objective is to minimize run time. For these reasons, there is room for improvement of dynamic metrics and schemes for TAPs as applied to DS.

In this paper, we present a dynamic reallocation strategy, based on the congestion measure developed in Shanker *et al.* (1993), that is suitable for DS problems where the objective is to minimize the *run time* of the simulation. By run time we mean a *realization* that is a function of the underlying simulation's probabilistic structure and the strategy adopted for implementation. For simulations using the same random numbers for identical purposes, different strategies could lead to different realizations of run time. Any strategy that consistently lowers this run time would lead to a lower expected run time. Experimental results suggest that the dynamic strategy provides us with an intuitive way of reallocating load to reduce simulation run time. As in Shanker *et al.* (1993), we use the Bryant/Chandy/Misra model (Bryant, 1977; Chandy and Misra, 1979, 1981; Misra, 1986) in our experiments.

The next section presents the reallocation strategy. Experimental design is discussed in Section 3, and results are in Section 4. Limitations of the strategy are in Section 5, and conclusions are in Section 6.

2. Strategy for reallocation

Two factors must be evaluated each time dynamic reallocation is considered: the cost of making the reallocation, and the benefit that can be achieved due to it. To carry out a dynamic scheme, time is spent collecting statistics, determining the new allocation, and reallocating the load.

To determine the cost effectiveness of a scheme we must know the benefit and cost of implementing it for each reallocation. While the cost can be estimated accurately, it is harder to determine the benefit since run time is known only at the end of the simulation. An equivalent but more tangible measure is δ_s, the expected departure rate of messages from the system. As run time is related inversely to δ_s, an assignment that increases δ_s will reduce run time if the cost of reassignment does not offset the improvement in the departure rate. While it may not be possible to predict the exact run time of the simulation, by observing the change in δ_s due to a reallocation, the apparent benefit may be predicted. Unlike run time, δ_s can be measured during the simulation, and by observing the change in it, the relative effects of an assignment can be estimated.

Towards this end, we use the *message utilization* π_j of processor P_j, developed in Shanker *et al.* (1993), as a measure in our dynamic scheme. Specifically, letting d be the number of available processors, we define for processor P_j, $j = 0, \ldots, d-1$,

$$\pi_j = \lambda_j / \mu_j, \tag{1}$$

where λ_j = the arrival rate of *new messages* and μ_j = the service rate of messages.

A message to LP_i on processor P_j is a new message if it either comes from an LP that is assigned to a different processor from the one to which LP_i is assigned, or if LP_i is a source LP. For example, any message generated at LP_1 (Fig. 1), a source LP, regardless of the assignment of LPs to processors, is a new message to the processor to which LP_1 is assigned. Similarly, if LP_1 and LP_2 are assigned to different processors, any message from LP_1 to LP_2 would be a new message to the processor to which LP_2 is assigned. A new message to processor P_j may visit many LPs on that processor before leaving it. At each LP, the message experiences a delay. A message is delayed because of waiting in queue until its precedence relationships are satisfied and the processor can execute it, and because of the execution time incurred at that LP. The total execution time incurred by a message on P_j is the sum of the execution times incurred at the various LPs visited by that message. Then $\psi_j = 1/\mu_j$ is the expected execution time incurred by a new message to P_j.

The metric (1) is based on the observation that, given λ_j and μ_j, the maximum departure rate δ_j of messages from P_j can be predicted. When $\lambda_j < \mu_j$, the maximum departure rate is limited by λ_j. Similarly, when $\lambda_j \geq \mu_j$, the maximum departure rate is limited by μ_j. In fact,

$$\delta_j \leq \min(\lambda_j, \mu_j). \tag{2}$$

As discussed in Shanker *et al.* (1993), the metric has important ramifications for any allocation scheme using it as a measure to minimize run time.

- π_j reflects the load on processor P_j. If $\pi_j > 1$ then P_j is receiving messages faster than it can process them. This is an unstable condition that sometimes arises in distributed-simulation environments and it is important to recognize and correct such situations.
- Assignments where precedence relationships are not being satisfied in a timely manner can be identified by using the metric as a measure of load. Reallocation can then be considered to minimize this effect.
- The effect of heterogeneous processors and positive communication times can be captured by dynamic schemes using the metric.
- The potential benefit of a *proposed* reassignment during the simulation can be estimated *before* it is implemented. Thus, the cost effectiveness of any proposed allocation, as compared to the present one, can be estimated. The following assumptions are made in estimating the potential benefit:
  - We either know or can collect observations to predict expected arrival and service rates of messages. (However, no assumption is made about their distributional forms).
  - We can collect observations to estimate expected communication rates between two processors for each message type.
  - The simulation at any point will behave like its recent past. This is an important assumption because allocation schemes are based on the observations collected.

The dynamic scheme developed below builds on the characteristics of the metric, and is implemented in three phases: *transfer*, *identification*, and *location* phases.

- The *transfer* phase determines when and at which processor(s) a reallocation should take place.
- The *identification* phase identifies the LPs that should be reallocated away from the affected processor(s) determined in the transfer phase.
- The *location* phase identifies the processors to which the LPs determined in the identification phase are reallocated.

2.1. *Transfer phase: identifying processors for reallocation*

This phase identifies the processors for reallocation, and the times at which a reallocation should be considered. As a reallocation involves moving LPs from one processor to another and not just individual messages, it is important that a reallocation be done only when the state of the affected processor has changed. The *state* of a processor is defined by the value of π_j in relationship to three thresholds: T_u, T_l, and T_a. The *upper* threshold limit is T_u, the *lower* threshold limit T_l, and T_a denotes the *availability* of a processor.

A processor P_j is *overloaded* and a candidate for reallocation if $\pi_j > T_u$. Ideally, we would like $\pi_j \approx 1$, $\forall j$, as this would indicate that all processors are being utilized to the fullest (though, because of communication costs this may not always be the best allocation for minimizing run time). But, as the arrival process of messages is usually unknown, and because precedence relationships for messages have to be satisfied before the messages can be executed, even when $\lambda_j < \mu_j$, P_j may experience an increasing queue of pending messages leading to a full buffer during the simulation. As any processor having $\pi_j > T_u$, $T_u < 1$, is a candidate for reallocation, T_u provides us with a means of controlling the maximum allowable load on a processor.

A processor P_j is *underloaded* if $\pi_j < T_l$. This state indicates that not enough work is coming into P_j, and is more likely to occur in *fine-grained* applications, where communication time becomes significant. For example, in a parallel add of two vectors, as the execution time needed

for each add operation is likely to be small, communication time becomes significant. In such situations, LPs could be transferred to P_j to increase its work, or the LPs in P_j could be distributed among other processors to reduce the communication among the processors, thereby reducing run time. Especially in the latter case, LPs should be assigned to reduce the overall run time and not just the communication time. Though the value of T_l depends on the computer system and the application, for most *coarse-grained* applications where communication is insignificant compared to execution time, like those in this paper, no processor is likely to be underloaded.

To determine a new allocation (discussed below), we say P_j is *available* if $\pi_j < T_a < T_u$. A threshold value below T_a signifies that the processor can receive some additional LPs without becoming overloaded. We can also view T_a as the desired threshold for all processors to maintain during the simulation. In our discussion, we do not necessarily differentiate in notation between parameters and their finite-horizon estimates. For example, π_j refers to the message utilization of processor P_j, but when calculated during the simulation will refer to the estimate of the message utilization. In most cases, it will be clear from the context if the notation refers to the parameter or to the estimated value.

The frequency of reallocation requests depends on the sample size used to estimate π_j, which is calculated by collecting observations for a fixed number of messages leaving P_j (Shanker, 1990). The sample size used in the scheme is a factor controlled by the modeler. If the sample size is small, but adequate for inference, it will ensure that the true conditions of the system are closely mirrored by the observations, but at the same time it will introduce communication traffic overhead as state information has to be passed among the processors. On the other hand, infrequent requests for reallocation reduce traffic overhead, but also reduce sensitivity to state changes. The optimal sample size will depend to a large extent on the system being simulated and the costs involved in observing state information. For simulations like those considered here, a sample size between 400 and 1000 messages works satisfactorily. Note that we are more interested in reallocating based on the *current* state rather than on the *true* state of the system. For example, assume that the system is inherently stable, i.e., $\pi_j < T_u$, $\forall j$. We are then interested in time periods when (the current finite-horizon estimate of) $\pi_j \geq T_u$ for some j. Though the true state implies stability, a reallocation would still be considered to smooth out the load. The frequency of reallocation would thus depend more on the cost effectiveness of the allocation scheme than on the statistical validity of the presumed state of the node. If reallocations could be performed at no cost, a new assignment could conceivably be implemented for each message arrival even though one observation will not provide much clue to the true state.

The scheme followed in this phase is as follows: A processor is a candidate for reallocation if it is either overloaded or underloaded. The processor ultimately chosen, say P_j, denoted as the *critical* processor, is the most overloaded processor. That is, $\pi_j > \pi_i, i \neq j$, and $\pi_j > T_u$. If no overloaded processors exist, the processor with the least load is deemed critical. That is, if P_j is that processor, then $\pi_j < \pi_i, i \neq j$, and $\pi_j < T_l$.

As will be discussed in the location phase, a reallocation is performed by moving LPs from an overloaded processor to an available processor. The levels of T_u and T_a thus affect the reallocation. High values of T_u would mean that fewer processors become critical, and therefore decrease the frequency of reallocation. But, low values of T_u often lead to unnecessary reallocations. The choice of T_u is therefore an important issue, and for simulations like those in this paper a value between 0.90 and 0.95 performs satisfactorily. If we had more knowledge of the system, T_u could be refined accordingly.

As the level of T_a denotes the availability of a processor, a low value would decrease the number of available processors for reallocation, but would guarantee their capacity to absorb additional load without becoming overloaded. A high value, on the other hand, would increase the chances of finding a suitable available processor for reallocation, but the available processor may not have enough capacity to accept additional load. For simulations like those considered in this study a value of T_a between 0.80 and 0.85 appears to work well.

It is important to note the distinction between the two thresholds, T_u and T_a. The point below which a processor has capacity to accept additional work without becoming overloaded is signified by T_a. If $T_u = T_a$, it could happen that any available processor P_j ($\pi_j < T_a = T_u$), could become overloaded the moment it accepts a single LP from another processor. As significant time is spent checking for a suitable reallocation it is advisable to keep $T_a < T_u$. Thus the difference is maintained to control unnecessary checking.

2.2. *Identification phase: identifying LPs for reallocation*

An important result from existing studies on task allocation has been that simple dynamic load-sharing algorithms generally achieve substantial performance improvement over static algorithms (Livny and Melman, 1982; Eager *et al.*, 1986; Mirchandaney *et al.*, 1989). As task allocation is done during run time, there is a need to keep the dynamic scheme itself simple and efficient. The number of LPs is typically much greater than the number of processors, so each processor will contain many LPs. Only those LPs that have predecessor LPs on different processors, called *beginning LPs*, or successor LPs on different processors, called *ending LPs*, will be considered for reallocation. This reduces the number of LPs to be considered and helps maintain the structure of the logical

system, implicitly reducing communication costs. Consider the logical system of Fig. 1. Mapping 1 in Fig. 2 shows the initial mapping to a computer with two identical processors. Mappings 2 and 3 show the same logical system with different assignments. In Mapping 2, an ending LP from P_0 (LP_8) has been moved to P_1, and hence the structure of the system and communication pattern remain unaltered, while the same is not true for Mapping 3. Here, communication has doubled (at least in distance travelled by a message), and the structure of the system is no longer maintained. Thus, it is probable that Mapping 2 will reduce run time compared to Mapping 3.

The scheme followed in this phase is given in Algorithm 1. At each step, beginning and ending LPs are moved from the critical processor P_j to an *available* neighboring processor (discussed below). The algorithm terminates when the critical processors's performance is within available threshold limits, or there are no more assignments possible.

2.3. *Location phase: identifying processors to which LPs are reallocated*

This phase identifies the processors *to* which LPs from the identification phase are reallocated.

To reinforce the identification phase strategy of the previous section, those processors that contain the predecessors of beginning LPs or successors of ending LPs identified for reallocation will be chosen first. Such processors are called *neighboring processors*. Specifically, in this phase the *first available* neighboring processor is chosen. This strategy is illustrated in Fig. 2. Mapping 5 shows the initial mapping of the earlier logical system to a computer with three identical processors. In Mapping 6 an ending LP, LP_4, has been moved to the neighboring processor P_1. The communication pattern and structure remain unaffected. In Mapping 7, LP_4 has instead been moved to P_2, increasing communication costs among processors.

Thus, the following strategy is considered in the location phase: Move the LPs (from P_j) identified in the identification phase to an available processor P_k. If $\pi_k^1 < T_u$, the allocation is kept, otherwise a new allocation is sought. The quantity π_k^1 is the *predicted* threshold of P_k under the proposed allocation. As shown in Shanker *et al.* (1993), π_k^1 can be estimated from observed data under the assumption that the simulation at any point will behave like its recent past. That is, the arrival rate of messages between LPs and the service rate of messages at LPs will remain unaffected by a different allocation. While this assumption could be violated in many instances, π_k^1 can still be estimated satisfactorily for purposes of choosing an alternative assignment (Shanker *et al.*, 1993). The scheme followed in the location phase is given in Algorithm 1.

Algorithm 1

```
/*Assumption: P_j is the critical processor*/
/*end LP refers to either a beginning or an ending LP*/
if (π_j > T_u) then /*overloaded processor*/
  for each end LP LP_i ∈ P_j DO
    move LP_i to available neighboring processor P_k
    /*neighboring processors that are also generating
    processors are considered last*/
    evaluate π_k^1
    if π_k^1 < T_u then
      keep allocation
    endif
  until [(π_j < T_a) or (no more end LPs)]
else if (π_j < T_l) then /*underloaded processor*/
  for each end LP LP_i ∈ P_j DO
    move LP_i to available neighboring processor P_k
    evaluate π_k^1
    if π_k^1 < T_u then
      keep allocation
    endif
  until [(π_j ≤ 0) or (no more end LPs)]
endif
```

The purpose of seeking a new allocation is to improve the departure rate of messages from the system. While the above scheme does not guarantee that δ_s will always increase, it is possible it may, especially when the strategy of moving beginning and ending LPs is coupled with the strategy of assigning LPs to neighboring processors. To understand why, consider the following examples:

Example 1. Consider the two different assignments shown in Mappings 1 and 2 in Fig. 2. For illustration purposes, assume that P_0 under Mapping 1 is an overloaded critical processor ($\pi_0 > T_u$). The objective is to improve the departure rate δ_s by reallocating LPs from P_0. Let λ_j^1 and μ_j^1 denote the arrival and service rates, respectively, of messages to P_j under a different assignment.

Consider the new assignment in Mapping 2. Here LP_8 (an ending LP) has been moved from P_0 to P_1. Therefore, the load at P_1 has increased, causing the service rate to decrease there, i.e., $\mu_1 \geq \mu_1^1$. At the same time, processor P_0's service rate has increased ($\mu_0^1 \geq \mu_0$). As $\pi_0 (= \lambda_0/\mu_0) > T_u$ and $\delta_0 \leq \min\{\lambda_0, \mu_0\}$, it is possible that since $\mu_0^1 \geq \mu_0$, the departure rate δ_0^1 under the new mapping has also increased ($\delta_0^1 \geq \delta_0$). Assuming negligible

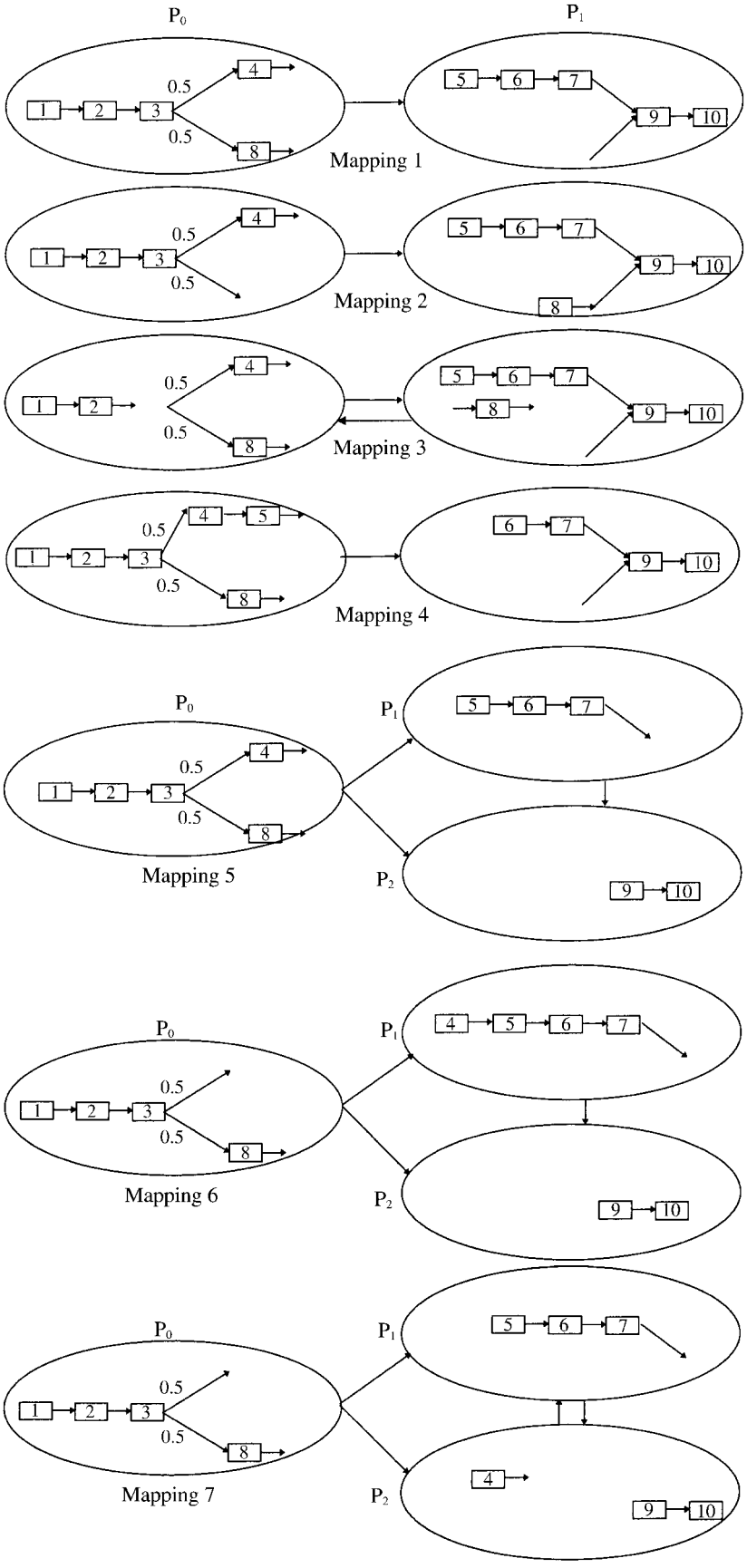

Fig. 2. Mappings 1–7.

communication times, $\lambda_1^1 \geq \lambda_1$, as the departure rate from P_0 will be the arrival rate into P_1. The reallocation has increased the arrival rate of messages to P_1, and at the same time decreased the service rate there.

A mapping would have been considered in the first place only if P_1 were an available processor. Therefore, since $\pi_1 < T_a < 1$ and $\delta_1 \leq \min\{\lambda_1, \mu_1\}$, the departure rate from P_1 is limited by the arrival rate. Under the new mapping, $\lambda_1^1 \geq \lambda_1$, and $\mu_1^1 \leq \mu_1$. But, as long as $\lambda_1^1 < \mu_1^1$, the departure rate from P_1 under the new allocation is limited by the arrival rate. As an allocation is kept only if $\pi_1^1 < T_u < 1$, the arrival rate will continue to be the limiting factor. Since $\lambda_1^1 \geq \lambda_1$, the departure rate $\delta_1^1 \geq \delta_1$. The reallocation has potentially increased the departure rate.

Example 2. As a different example, consider Mappings 1 and 4 (Fig. 2). Now let P_1 be an overloaded processor and P_0 an available processor in Mapping 1. In Mapping 4, a beginning LP from P_1 has been moved to P_0. By moving LP_5 to P_0, the service rate at P_0 has decreased, but the arrival rate is unaffected because LP_5 is not a beginning LP on P_0. As long as $\pi_0^1 < T_u$ the arrival rate is still the limiting factor for the departure rate at P_0, but now P_1, by reducing its load, has increased its service rate without significantly affecting the run time of the system. While an increase in departure rate is not always assured, the above strategy inherently balances the system such that each processor receives only as much work as it can process efficiently (by controlling T_u). If an increase in the departure rate must be guaranteed each time a reallocation is done, we would have to calculate δ_j^1, $\forall j$, a time-consuming process. As the departure rates are predicted from observed values there is no way of determining the accuracy of a prediction beforehand. Thus, the effort is seldom worthwhile.

An alternative approach to reallocate load is to use the fact that messages in a DS are generated internally. In a DS, new messages originate at source LPs (for the logical system of Fig. 1 the source LP would be LP_1). The rate at which these messages are generated depends on the assignment of LPs to processors. If a source LP is assigned to an overloaded processor, the rate of message generation will be slow (assuming that a new message is generated only when the previous message has finished its processing on that processor). Alternatively, if the source LP resides on a relatively free processor, then the rate of message generation could be high. Thus, processors having source LPs, called *generating processors*, govern the rate of message generation. As generating processors usually remain stable (they adjust the arrival rate to match their service capacity – unless they also experience arrivals from other processors, in which case they could become unstable), they are handled differently. When a reallocation is considered, even if a generating processor is the first available neighboring processor, LPs are reallocated to them only as a last resort, as a reallocation will reduce the rate at which new messages are generated, and hence will affect the overall load. This fact, coupled with the earlier strategy of reallocating only to neighboring processors, provides an effective way of reallocating load.

3. Experimental analysis

An experimental study evaluated the effect on run time of using the dynamic scheme on tasks such as those found in DS. The study considered three logical systems (Fig. 3) that are based on systems used in previous studies (Shanker *et al.*, 1989; Shanker, 1990) and suitable for simulating on the iPSC/2 Hypercube. The logical systems, by their structures, also introduced different degrees of difficulty in finding new assignments during the simulation. The experimental design included the following factors:

A: *The scheme used*: The performance of the dynamic scheme was compared to a static strategy chosen to achieve the best possible run time for the simulation under the assumption that the initial load remained unchanged. The static strategy was chosen by performing exploratory experiments evaluating different assignments of LPs to processors under initial load conditions.

B: *Increase in load*: This factor controlled the change in processing time for a message at an LP, and was considered at two levels. The processing time for a message at all LPs was the same initially.

C: *The time of load increase*: This factor specified the number of messages that were executed before an

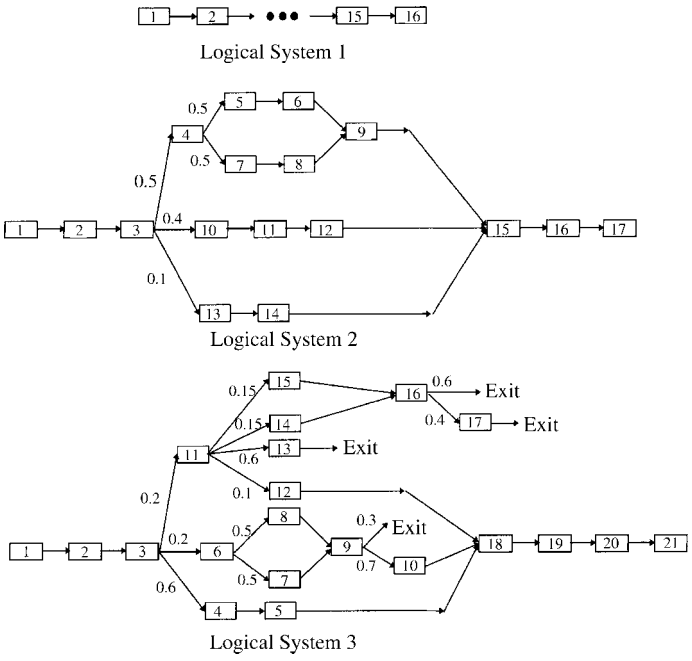

Fig. 3. Logical systems 1–3.

increase in processing was required at an LP. This was considered at two levels.

D: *The location of load increase*: While factors B and C specify the amount and time of load increase, this factor indicates the location of load increase. Specifically, the LP experiencing the load increase is defined here. This also was considered at two levels.

In addition, the following factors might also affect the performance of the schemes:

- *Simulation run length*: Technological constraints on the message-buffer space of the hypercube limited the maximum simulation run length to between 900 to 5000 messages depending on the logical system.
- *Frequency of reallocation requests*: This specifies the sample size needed for making inferences about the system state. Small sample sizes lead to a high frequency of reallocation requests and system overhead, but they also closely mirror the true conditions of the system. Large sample sizes, have less overhead, but are also less sensitive to system states. In our experiments, the sample size varied from 400 to 1000 messages depending on the logical system.
- *Threshold values*: The values of T_u, T_a, and T_l were set based on preliminary simulations done to study their effect on performance. For the simulations like ours, T_u ranged from 0.92 to 0.95, T_a was set at 0.85, and T_l from 0.20 to 0.25. While finding the best values of the parameters may be difficult, for most simulations it is relatively easy to determine good values if one has some prior knowledge about the system. In addition, for our stochastic simulations, as the load varies from run-to-run, the threshold values tend to be robust – a range of values for the thresholds work well. In this case the stochastic nature of the simulations is in fact an advantage in determining appropriate threshold values as it gives a wider latitude in setting them.
- *Initial allocation*: Assignment provided by the static strategy mentioned earlier.
- *Initial load value*: This specifies the initial amount of computation required by each LP. Higher values imply a greater computation time for a message at that LP.
- *Logical system*: An important assumption made in implementing the dynamic scheme is that "future observations resemble the recent past". As discussed in the following sections, this assumption is violated when there is a strong dependency among LPs in a system, as in closed networks. As such, all logical systems considered here are open networks.

The experiments were conduced on an iPSC/2 Hypercube, which is a distributed system of connected *nodes*. Each node is a self-contained computer, and any information to be shared between nodes is communicated via messages through a high-speed network connecting the nodes. These nodes are usually accessed from the outside world by means of a *host* computer. The host could be any computer system, e.g., a Unix workstation, and is used as an interface between the user and the Hypercube.

Two different sets of experiments were conducted, differing in how the Hypercube was used for dynamic reallocation. In the first set of experiments, all calculations pertaining to the scheme were done on the nodes. Here the host computer was used only for starting and closing the cube, i.e., to provide the user with feedback about the simulation. In the second set of experiments, calculations for determining a new allocation were done on the host while the nodes continued simulating. This favors the dynamic scheme as the overhead associated with calculating the reallocation is no longer present. It is important to note that in both sets of experiments there is still the overhead of actually implementing the reallocation. In all cases a 2^4 full factorial design was used with the four factors mentioned previously, and all data were generated by using non-overlapping random-number streams. Each experiment was replicated until statistically significant results were obtained at the 90% confidence level.

4. Results

The results of the experiments, shown as a comparison between the dynamic and static scheme, are in Tables 1, and 2. There are two levels for each of the four design factors A (Scheme), B (Load), C (Time), and D (Location). As the results in Tables 1, and 2 are comparisons between the performance of the dynamic and static schemes (factor A), we have eight pairs of design points. In the interest of brevity, complete treatment levels are not described here, but specific levels will be mentioned for illustration purposes. Further details may be found in Shanker (1990).

4.1. *Experiment 1: reallocation using only the nodes*

The run time achieved by using the dynamic scheme was in many cases substantially improved over those resulting

Table 1. Experiment 1: percent change in run time

Design point	*Logical system*		
	1	*2*	*3*
1	−6.27	−23.69	0.41
2	−20.24	−26.62	0.20
3	−1.47	−33.51	−0.16
4	−16.67	−43.17	−0.07
5	6.56	−21.11	−49.70
6	9.13	−27.22	−35.82
7	5.87	−20.02	−33.55
8	6.35	−35.29	−30.24

Table 2. Experiment 1: average thresholds and deviations in load

Design point	*Logical system 1*		*Logical system 2*		*Logical system 3*	
	Dynamic	*Static*	*Dynamic*	*Static*	*Dynamic*	*Static*
A: Average thresholds						
1	0.7578	0.6846	0.5602	0.5902	0.7333	0.7329
2	0.7694	0.6852	0.5934	0.6123	0.7509	0.7487
3	0.7571	0.6890	0.5517	0.5910	0.7359	0.7298
4	0.7674	0.6882	0.5699	0.6136	0.7494	0.7413
5	0.7638	0.7703	0.5473	0.5900	0.7454	0.7252
6	0.7686	0.7689	0.5344	0.6125	0.7376	0.7122
7	0.7647	0.7696	0.5555	0.5909	0.7552	0.7360
8	0.7691	0.7706	0.5350	0.6137	0.7589	0.7244
B: Average deviations in load						
1	0.0374	0.0734	0.0681	0.0751	0.0405	0.0404
2	0.0210	0.0828	0.0405	0.0982	0.0405	0.0407
3	0.0453	0.0710	0.0213	0.0760	0.0403	0.0407
4	0.0251	0.0852	0.0537	0.0985	0.0406	0.0404
5	0.0131	0.0110	0.0483	0.0758	0.0227	0.0561
6	0.0187	0.0169	0.0265	0.0982	0.0261	0.0502
7	0.0135	0.0116	0.0569	0.0758	0.0234	0.0629
8	0.0157	0.0150	0.0298	0.0986	0.0252	0.0566

from the static scheme. Table 1 shows the percent change in run time, $100 \times [d - s]/s$, where d and s are the average run times observed when using the dynamic and static schemes, respectively. For logical system 3 (LS_3), the run time improvement was between 0 and 50% using the dynamic scheme, and for LS_2 the improvement was between 20 and 44%. Though the improvement for LS_1 was not as great (Table 1), it was nevertheless significant for half the simulation runs (design points 1–4). In fact, in a few cases for LS_1, the dynamic scheme increased the average run time. There were two main reasons for this increase: the topology of the logical system simulated, and the strategy used for task allocation. The dynamic scheme attempts to improve the run time by taking a *localized* view of the problem. That is, tasks from processor P_i are reallocated to P_j if that reallocation will increase the net departure rate of messages from the above two processors, without considering the effect on the overall system. This strategy could fail when there is a high degree of dependency among LPs (even if precedence relationships are satisfied immediately), as it is in the pipelined topology of LS_1. Here, any change in processing at LP_i has an immediate and significant effect on LPs downstream, and observations collected in the recent past may not accurately reflect future values under a different assignment. Thus, a reallocation may merely shift the bottleneck process rather than eliminate it. This was exactly the case for design points 5–8 where LP_{12} was transferred to P_3 during the simulation. Here, the bottleneck process shifted from P_2 to P_3 (the Hypercube had four available processors), and as such no reduction in run time was observed.

The other factor that hindered performance of the dynamic scheme for LS_1 was that the initial allocation was a balanced load across the processors (this allocation provided the best run time under initial conditions of the simulation). Thus, any change in load at an LP has an immediate effect on performance. For such systems, if an improvement in run time is sought when load changes, the load needs to be balanced again across the system. This may not always be possible as LPs, and not individual messages, are being reallocated. An alternative approach to balancing the load is to lower the overall load in the system, thereby eliminating the bottleneck, rather than shifting the bottleneck. This is done by reducing the rate at which new messages are generated. This approach is illustrated in Fig. 4 for LS_1, showing on the left the arrival and service rates in LS_1 under the initial allocation. The rates were calculated using the algorithm given in Shanker *et al.* (1993) and an initial load value of 15. For example, consider the calculation of the rates for P_0 under the initial allocation. As each new message generated in P_0 is executed at all four LPs there, the total average execution time incurred by a message in P_0 is $\psi_0 = 60$. Assuming that a new message is generated at LP_1 only when no other messages in P_0 require processing, $\lambda_0 = \mu_0 = 1/60$. Then, using (2), an estimate of δ_0 is $1/60$. During the simulation, the load at LP_6 changed to 29. The effect of this is shown in the center of Fig. 4. The dynamic strategy then moved LP_5 to P_0 to "balance" the load in the system. As P_0 is a generating processor, the overall load in the system changes. This effect is shown on the right in Fig. 4, where now the bottleneck has been removed from the system by altering the overall

rate at which messages are generated. While this strategy removes the bottleneck, δ_s also decreases. But in many cases the bottleneck caused by a full buffer leads to unpredictable run times. While this does not necessarily imply longer run times, from limited experiments it appears to be so. The success of this strategy therefore

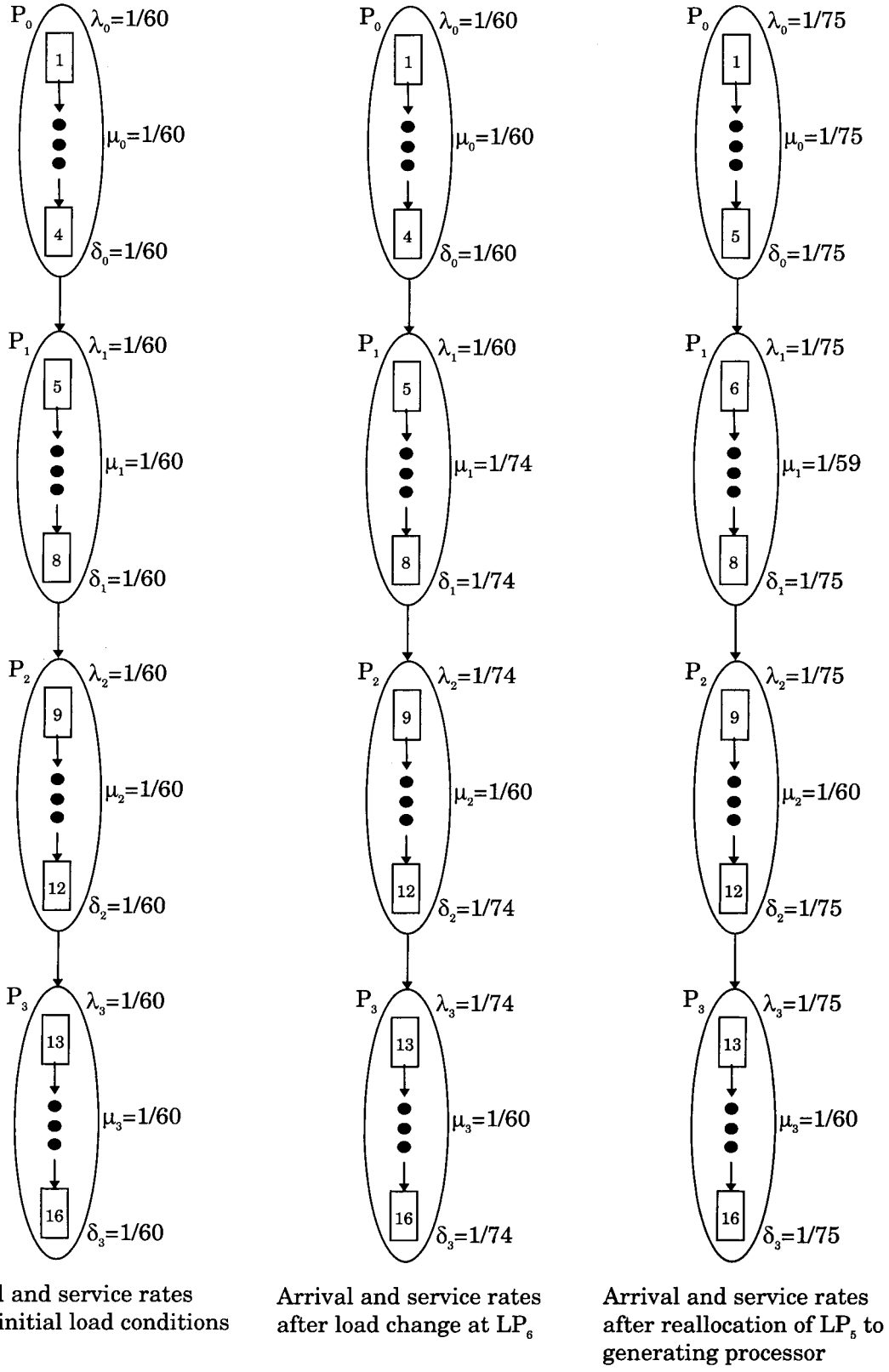

Fig. 4. Expected arrival and service rates for LS_1.

depends on the overhead involved in resolving a full buffer.

The difference in the performance of the dynamic scheme on logical systems with less dependency among LPs than LS_1 can be seen readily by considering LS_2 and LS_3. For LS_2, the dynamic scheme does better than the static scheme in all cases, and for LS_3, in 50% of the cases. In no case does the dynamic scheme perform worse than the static scheme (Table 1). In addition to the fact that the LPs are less dependent in LS_2 and LS_3 than in LS_1, the initial assignment is also not as well balanced as it was for LS_1. Thus, processors are able to absorb small changes in load without becoming overloaded. This can be seen by considering LS_3. When load changes at LP_5 the dynamic scheme does not consider a new allocation. As only about 60% of all messages pass through LP_5 (Fig. 3), small increases in load will have minimal effect on performance. But, when load changes at LP_{20}, there is a greater effect. This is because nearly 75% of all messages pass through LP_{20}.

Table 2A shows the estimated average threshold values, i.e., an estimate of message utilization π, for the three logical systems. If π_j^{ik} is the estimated threshold value for P_j at the end of run i for design point k, then the average threshold value $\bar{\pi}^k = \sum_{j=0}^{d-1} \bar{\pi}_j^k / d$, where $\bar{\pi}_j^k = \sum_{i=1}^{N_k} \pi_j^{ik} / N_k$, and N_k is the number of replications at design point k. Unlike utilization, π_j^{ik} is calculated for only as long as messages are arriving or being processed, and any idle time at the end of the simulation for a processor (when other processors are still working) is not taken into account. Thus, π_j represents the utilization as long as messages are being processed. For LS_1 and LS_3 (Table 2A) the average threshold tends to be higher for the dynamic scheme, but for LS_2, not only did the dynamic scheme have a lower average threshold, but the balance of load among processors, i.e., the deviation (variance) in threshold values among processors, is lower while using the dynamic scheme (Table 2B). This suggests that the dynamic scheme lowers the run time by using the processors more efficiently and is therefore able to absorb a greater degree of load change in the system. In general, for all logical systems the dynamic scheme achieved a better balance of load among the processors, leading to a lower run time when compared to the static strategy.

4.2. *Experiment 2: reallocation using the host*

Another set of experiments was conducted to determine the effectiveness of using the dynamic scheme when the host, instead of the nodes, was used for calculating the new assignment. Though the factors were the same as in the previous experiment they were considered at different levels, and only LS_1 and LS_2 were simulated (Shanker *et al.*, 1993).

The results from this study are similar to the previous one. Again the dynamic scheme did better, or at least as well, in all cases compared to the static scheme. One difference from the previous study was in the pattern observed for average threshold and deviation in load among processors for LS_1. For points 1–4, the dynamic scheme achieved a lower average threshold value in contrast to the previous experiment. Also, while the average deviation in load among processors was clearly lower for the dynamic scheme in the previous experiment (Table 2B), no such pattern could be detected here. Unlike the earlier experiment, load was not always reallocated by moving LPs to a generating processor. This was reflected in lower threshold values for the dynamic scheme, and is similar to the results observed for logical system 2 where load is usually reallocated to processors other than generating processors leading to lower threshold values for the dynamic scheme. As no reallocation was done at design points 5–8 there was not much difference in either the average threshold or deviation values between the static and dynamic scheme for LS_1. Even though a lower run time was achieved by the dynamic scheme for LS_1 by moving LPs to processors other than the generating processor, it could be that the bottleneck has only been shifted, and not removed, a symptom possibly masked by the small run length. Thus if the simulation had continued, a reallocation to a generating processor may have been necessary to reduce the run time. This would also explain the discrepancy in the pattern observed for the two experiments. Again, such problems are more likely to occur in systems like LS_1 where the dependency among LPs is high.

For LS_2, the results mirrored those of the earlier experiment. The run time with the scheme was lower in all cases, and so were the average threshold values. In addition, the dynamic scheme achieved a better balance of load among processors, supporting the contention that the scheme uses the processors more efficiently.

4.3. *Experiment 3: effect of system characteristics on run time*

The above experiments show that the dynamic scheme was often effective in reducing run time, and generally the scheme with a lower deviation in load among processors led to the lower run time. This supports earlier studies where the variance of load distribution among processors has been used as the minimization measure (Lu and Carey, 1986). At face value, this implies that regardless of the problem and the computer it is better to distribute the load uniformly across the available processors. This would not be correct since the number of processors used should depend on the logical system as well as the computer. The following experiment shows us an intuitive way of assigning load along these lines.

LS_1 was considered. The average execution time (estimated through preliminary experiments) for a message on

Table 3. Approximate values for δ_0 ($1/\zeta \approx 0.3333$)

Assignment	*Available processors*	δ_0
1	P_0	0.1923
2-B	P_0, P_1	0.3846
2-UB	P_0, P_1	0.3419
4	$P_0 - P_3$	0.7692
8	$P_0 - P_7$	1.5385

all LPs was the same and ≈ 0.325 ms. The communication time between processors was empirically estimated to be ≈ 3 ms. Five different static assignments were considered (Table 3). In each assignment, except one, 2-UB (2 processors, unbalanced load), the LPs were distributed uniformly over the available processors. In 2-UB, the LPs are allocated so that the bottleneck (the most utilized resource) occurs at the processors and not in the communication channels, while in 2-B (2 processors, balanced load), 4 and 8, the bottleneck is in the communication channels, a result of distributing the load uniformly. Specifically, in 2-UB, LPs are allocated to P_0 until $\delta_0 \approx 1/\zeta$, where ζ is the expected communication time between processors. The remaining LPs are then assigned to P_1. The primary interest was to determine the relative performance of the above allocations.

In the first set of experiments the simulation was run for 800 messages. The results are given in Table 4. Clearly 2-UB is the best allocation. Consider the simulation under assignment 2-B. Here P_0 sends messages to the communication channel at a faster rate than it can handle, and hence the I/O buffer becomes full. When this happens, P_0 stops its processing and waits for the buffer to clear before continuing, thus wasting time. This situation arises when using 4 and 8 processors also because $\delta_0 > 1/\zeta$. But in 2-UB, P_0 executes messages at a rate comparable to the communication rate between processors, and under such circumstances no time is lost because of a full buffer. Note that, because of the logical structure and mappings considered the bottleneck will either be at P_0 or in the communication channel from P_0 to P_1, and hence the results can be explained with reference to δ_0 and ζ.

The implications of choosing the appropriate number of processors for a system with limited I/O space can be demonstrated even more emphatically by increasing the run length to 2000 messages, and hence increasing the chance of filling up the I/O buffer. Now there is a sharp decrease in the departure rate of messages for assignments 2-B, 4, and 8, while the results for 2-UB and 1 are comparable to the previous experiment. Therefore, when communication time is significant, a balanced load may not always lead to the best run time, and even if the load is balanced it is essential that a proper subset of processors be chosen for the assignment. One intuitive way of assigning LPs is to balance the departure rate of messages from the processor to the communication rate so that the impact of limited I/O space is neutralized. The dynamic scheme tries to do just this. In the earlier experiments in Section 4.1, a balanced load generally led to a lower run time because the simulations were coarse-grained, and hence communication was not the bottleneck.

All simulations considered in Sections 4.1 and 4.2 experience an increase in load, and the results show that even for small increases in load it is cost-effective to use the dynamic scheme. The percent overhead ([reallocation cost / run time] $\times$ 100) in implementing the dynamic scheme for the three simulations of Section 4.1 is shown in Table 5, which also shows the overhead of the dynamic scheme when the simulations of LS_2 experience no increase in load. Clearly, the overhead in all cases is quite low. The overhead in using the scheme is less than 4%, while the potential improvement could be as high as 44% (see Table 1). In addition, in most simulations it would be difficult to determine the best static assignment *a-priori*, as has been done for the experiments of Sections 4.1 and 4.2, and in such cases it may be cost-effective to use the dynamic scheme to smooth out the load during the simulation. In addition to the different factor levels chosen in Section 4.2 (compared to those in Section 4.1), the results in Table 5 could also explain why no significant benefit was derived from calculating the new assignment on the host in Section 4.2 when compared to the experiments performed only on the nodes.

Table 4. Average departure rate of messages from the system

Number of processors	*Average departure rate*	
	800 messages	*2000 messages*
1	0.1955	0.1992
2-B (balanced load)	0.2484	0.1149
2-UB (unbalanced load)	0.2933	0.3019
4	0.2373	0.1173
8	0.1331	0.0836

Table 5. Percent overhead

Design point	*Logical system*			
	1	*2*	*3*	*2 (no load)*
1	1.190	2.381	0.097	1.735
2	1.330	3.750	0.093	1.700
3	1.135	2.536	0.090	1.735
4	1.387	3.764	0.104	1.768
5	1.419	2.049	0.071	1.692
6	1.239	2.844	0.078	1.840
7	1.455	1.859	0.089	1.753
8	1.322	2.612	0.129	1.778

5. Limitations

From the experiments of the previous section, it is clear that the dynamic scheme often proved superior to the static strategy. But many characteristics can be identified that affect the performance of the scheme. A few are discussed below.

The dynamic strategy uses a localized approach to task allocation. That is, LPs from processor P_i are reallocated to P_j if that reallocation will increase the net departure rate of messages from the above two processors, without considering the effect on the overall system. As seen in Section 4, this strategy sometimes leads to inefficient allocations for LS_1 (Fig. 3), where there is a high degree of dependency among the LPs. But, as dynamic task allocation is based on *observed* values, and involves significant cost during the simulation, a more globalized approach would require greater efficiency to remain cost effective.

A related aspect is the assumption that future observations will behave like the recent past. This assumption is the basis under which a new allocation is calculated by the dynamic scheme. When future values are significantly affected by a new allocation, the dynamic scheme, partly because of its localized approach, is likely to perform poorly. From our experiments, these situations occur in well-structured systems like LS_1 (Fig. 3), or when there are immediate feedback loops at an LP. This assumption, because of the cyclic dependency among LPs, is also usually violated in closed networks. While it is possible to modify the scheme to account for such situations, it is likely to increase the overhead significantly. And even then, a better allocation cannot be guaranteed as future values will still be predicted from values observed in the past.

Another factor affecting performance is the setting of threshold values T_u , T_a, and T_l. For the simulations in this paper, the threshold values were set based on preliminary experiments, and remained fixed for the duration of the simulation. While a range of values worked well in our experiments, it may be difficult to find the best values for all simulations. An alternative approach is to view the thresholds as variables, with their values defined as a function of the system's state. For example, T_u could be a function of the overall load. If the load on all processors was high, it might be better to keep T_u high to prevent unnecessary calculations (as there would be no available processors, and therefore no reallocation would be done). While this strategy has obvious advantages, determining a suitable function for the thresholds may be difficult. Further research is needed for choosing appropriate values before simulations can be run efficiently in a manner transparent to the end user.

Another aspect that affects performance is the handling of precedence relationships among LPs. Currently, the scheme can recognize when messages are spending excessive time waiting in queue, thus implying that precedence relationships are not being satisfied, but has no provision to correct the situation directly. Precedence relationships among LPs can be represented by a network of queues with multiple arrival streams. For example, consider the logical system in Fig. 1. Using the Bryant/Chandy/Misra model, messages will be chosen by LP_9 from the two channels, i.e., from the channels from LP_7 and LP_8 to LP_9, based on the timestamp of messages. Thus the precedence relationship at LP_9 can be modelled as a single-server system with two arrival streams. The queue discipline there is dictated by the timestamp of messages arriving in the two channels, as the server (LP_9) will always pick the message with the smallest timestamp. The problem is in predicting the throughput for such systems. While limited results exist (Kumar and Shorey, 1994), to our knowledge no results yet exist for systems with multiple general arrival streams and general service discipline (i.e., $\sum GI/G/1$ systems), and queue disciplines defined by precedence relationships like those in DS. Thus, while it is easy to recognize when precedence relationships are not being satisfied, it is difficult to correct them as the effect of a reallocation on precedence relationships cannot as yet be predicted accurately.

6. Conclusions

While TAPs in distributed systems have been studied extensively, as mentioned earlier, most problems make assumptions that limit their utility to DS. Thus, solutions developed in other fields are usually unsuitable (in terms of reducing run time) for task allocation in DS. Also, while some work has been done in the area of dynamic task allocation for DS (Nicol and Reynolds, 1985; Reiher and Jefferson, 1990), most modelers generally rely on guesswork to determine a good allocation, and that too a static one.

This paper establishes the feasibility, and in some cases the necessity, of using dynamic task allocation in DS, and provides an intuitive and simple way of reallocating load during the simulation to reduce its run time. More importantly, the implications of this research could lie in its potential application to TAPs in other domains. For example, by modeling the manufacturing environment mentioned in Section 1 as a distributed simulation, it is possible to evaluate the effect of the dynamic scheme on throughput. While the manufacturing model is more complex than the examples here, our results may nevertheless hold. From a practical standpoint, for the success of the scheme we need to assume that observations collected in the recent past represent the near-term future. While this appears to be overly restrictive, the predictions of a potential assignment of LPs to processors can be made robust to moderate departures from the above

assumption. Thus, unless there is a strong dependency among LPs, thereby possibly severely violating the assumption, the dynamic scheme is likely to do well.

Acknowledgments

We would like to thank Sartaj Sahni for his valuable contributions to this work, and The Minnesota Supercomputer Institute and Argonne National Laboratory for providing computational support. We would also like to thank an anonymous referee and the guest editor David Goldsman for their many valuable suggestions.

References

Baumgartner, K.M. and Wah, B.W. (1989) GAMMON: a load balancing strategy for local computer systems with multiaccess networks. *IEEE Transactions on Computers*, **38**(8), 1098–1109.

Bokhari, S.H. (1979) Dual processor scheduling with dynamic reassignment. *IEEE Transactions on Software Engineering*, **SE-5**(4), 341–349.

Bourne, D.A. and Fox, M.S. (1984) Autonomous manufacturing: automating the job-shop. *IEEE Computer*, **17**(9), 76–86.

Bryant, R.E. (1977) Simulation of packet communication architecture computer systems. Technical Report LCS, TR-188, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge MA.

Chandy, K.M. and Misra, J. (1979) Distributed simulation: a case study in design and verification of distributed programs. *IEEE Transactions on Software Engineering*, **SE-5**(5), 440–452.

Chandy, K.M. and Misra, J. (1981) Asynchronous distributed simulation via a sequence of parallel computations. *Communications of the ACM*, **24**, 198–206.

Chu, W.W. and Lan, M.-T. (1987) Task allocation and precedence relations for distributed real-time systems. *IEEE Transactions on Computers*, **C-36**(6), 667–679.

Eager, D.L., Lazowska, E.D. and Zahorjan, J. (1986) Adaptive load sharing in homogeneous distributed systems. *IEEE Transactions on Software Engineering*, **SE-12**(5), 662–675.

Eick, S.G., Greenberg, A.G., Lubachevsky, B.D. and Weiss, A. (1993) Synchronous relaxation for parallel simulations with applications to circuit-switched networks. *ACM Transactions on Modeling and Computer Simulation*, **2**(4), 287–314.

Garey, M.R. and Johnson, D.S. (1979) *Computers and Intractability*: *A Guide to the Theory of NP-Completeness*, W.H. Freeman and Company, San Francisco, CA.

Goswami, K.K., Devarakonda, M. and Iyer, R.K. (1993) Prediction-based dynamic load-sharing heuristics. *IEEE Transactions on Parallel and Distributed Systems*, **4**(6), 638–648.

Iqbal, A.M., Saltz, J.H. and Bokhari, S.H. (1986) A comparative analysis of static and dynamic load balancing strategies, in *Proceedings of the International Conference on Parallel Processing*, Pennsylvania State University Press, University Park, PA, pp. 1040–1047.

Kumar, A. and Shorey, R. (1994) Stability of event synchronization in distributed discrete event simulation, in *Proceedings of the 8th Workshop on Parallel and Distributed Simulation*, Simulation Council, San Diego, CA, pp. 65–72.

Livny, M. and Melman, M. (1982) Load balancing in homogeneous broadcast distributed systems, in *Proceedings ACM Computer Network Performance Symposium*, ACM Press, New York, NY, pp. 47–55.

Lu, H. and Carey, M.J. (1986) Load-balanced task allocation in locally distributed computer systems, in *Proceedings of the International Conference on Parallel Processing*, Pennsylvania State University Press, University Park, PA, pp. 1037–1039.

Mirchandaney, R., Towsley, D. and Stankovic, J.A. (1989) Analysis of the effects of delays on load sharing. *IEEE Transactions on Computers*, **38**(11), 1513–1525.

Misra, J. (1986) Distributed discrete-event simulation. *Computing Surveys*, **18**(1), 39–65.

Nandy, B. and Loucks, W. (1992) An algorithm for partitioning and mapping conservative parallel simulation onto multicomputers, in *Proceedings of the 6th Workshop on Parallel and Distributed Simulation*, Simulation Council, San Diego, CA, pp. 139–146.

Ni, L.M. and Hwang, K. (1985) Optimal load balancing in a multiple processor system with many job classes. *IEEE Transactions on Software Engineering*, **SE-11**(5), 491–496.

Nicol, D.M. and Reynolds, Jr., P.F. (1985) An optimal repartitioning decision policy, in *Proceedings of the 1985 Winter Simulation Conference*, Institute of Electrical and Electronics Engineers, Inc., New York, NY, pp. 493–497.

Reiher, P. and Jefferson, D. (1990) Dynamic load management in the time warp operating system. *Transactions of the Society for Computer Simulation*, **7**(2), 91–120.

Shanker, M.S. (1990) Resource Utilization Through Dynamic Task Allocation. Ph.D. thesis, University of Minnesota, and working paper 90-5, Department of Operations and Management Science, Minneapolis, MNn.

Shanker, M.S., Kelton, W.D. and Padman, R. (1989) Adaptive distribution of model components via congestion measures, in *Proceedings of the 1989 Winter Simulation Conference*, Institute of Electrical and Electronics Engineers, Inc., New York, NY, pp. 640–647.

Shanker, M.S., Kelton, W.D. and Padman, R. (1993) Measuring congestion for dynamic task allocation in distributed simulation. *ORSA Journal on Computing*, **5**(1), 54–68.

Shin, K.G. and Chang, Y.-C. (1989) Load sharing in distributed real-time systems with state-change broadcasts. *IEEE Transactions on Computers*, **38**(8), 1124–1142.

Stone, H.S. (1977) Multiprocessor scheduling with the aid of network flow algorithms. *IEEE Transactions on Software Engineering*, **SE-3**(1), 85–93.

Tantawi, A.N. and Towsley, D. (1985) Optimal static load balancing in distributed computer systems. *Journal of the ACM*, **32**(2), 445–465.

Woodside, C.M. and Monforton, G.G. (1993) Fast allocation of processes in distributed and parallel systems. *IEEE Transactions on Parallel and Distributed Systems*, **4**(2), 164–174.

Biographies

Murali S. Shanker is an Associate Professor in the College of Business at Kent State University. He received his Ph.D. from the Department of Operations and Management Science at the University of Minnesota. His research interests are in simulation, parallel processing, and artificial neural networks. His current research is in developing task-allocation strategies for distributed simulation models, and in conducting statistical analysis of the behavior of neural networks as applied to classification and prediction problems.

Rema Padman is an Associate Professor of Operations and Information Management at the Heinz School of Public Policy and Management at Carnegie Mellon University, She has a B. Tech., in Chemical Engineering from the Indian Institute of Technology at Kanpur, India, an M.S. in Operations Research from the University of Texas at Dallas and a Ph.D. in Operations Research from the University of Texas at

Austin. Prior to joining Carnegie Mellon University, she was on the faculty at the Carlson School of Management at the University of Minnesota. Her primary research interests are in operations planning and management and healthcare information systems. Her work has addressed a range of issues related to the application of data mining and metaheuristic strategies using distributed, connectionist, and evolutionary programming approaches for operational decision making in areas such as production planning, support requirements planning, and project management and scheduling. Her current research in the healthcare area focusses on issues related to confidentiality and privacy of healthcare data, productivity and impact of information systems use by health management organizations, and data mining for decision support. She has published in journals such as *Management Science*, *Operations Research*, *Naval Research Logistics*, *Communications of the ACM*, *Journal of the American Statistical Association*, *INFORMS Journal on Computing* and *European Journal of Operational Research*.

W. David Kelton is a Professor in the Department of Quantitative Analysis and Operations Management at the University of Cincinnati. He received a B.A. in Mathematics from the University of Wisconsin-Madison, an M.S. in Mathematics from Ohio University, and M.S. and Ph.D. degrees in Industrial Engineering from Wisconsin. He was formerly on the faculty at the University of Minnesota, the University of Michigan, and Kent State. Visiting posts have included Wisconsin, the Institute for Advanced Studies in Vienna, and the Warsaw School of Economics. His research interests and publications are in the probabilistic and statistical aspects of simulation, applications of simulation, statistical quality control, and stochastic models. He is coauthor, with Averill M. Law, of *Simulation Modeling and Analysis*, in its third edition with McGraw-Hill. He is also co-author of *Simulation with Arena*, with Randall P. Sadowski and Deborah A. Sadowski, published by McGraw-Hill. David serves as Editor-in-Chief for the *INFORMS Journal on Computing* and has been on the editorial boards of, among others, *IIE Transactions*, *Operations Research*, the *Journal of Manufacturing Systems*, and *Simulation*. In 1994 he received the IIE Operations Research Division Award. During 1990–1992, he was President of the TIMS College on Simulation. In 1987 he was Program Chair for the Winter Simulation Conference, and in 1991 was General Chair.