ORSA Journal on Computing
Vol. 5, No. 1, Winter 1993

0899-1499/93/0501-0054 $01.25

Measuring Congestion for Dynamic Task Allocation in Distributed Simulation

MURALI S. SHANKER / *Department of Administrative Sciences, Kent State University, Kent, OH 44242, (216) 672-2750; Email: mshanker@scorpio.kent.edu*

W. DAVID KELTON / *Department of Operations and Management Science, Carlson School of Management, University of Minnesota, Minneapolis, MN 55455, (612) 624-8503; Email: dkelton@ux.acs.umn.edu*

REMA PADMAN / *School of Urban and Public Affairs, Carnegie Mellon University, Pittsburgh, PA 15213, (412) 268-2180; Email: rp25@andrew.cmu.edu*

(Received: June 1991; revised: February 1992; accepted: April 1992)

An important factor affecting the performance of distributed simulations running on parallel-processing computers is the allocation of logical processes to the available physical processors. An inefficient allocation can result in excessive communication times and unfavorable load conditions. This leads to long run times, possibly giving performance worse than that with a uniprocessor sequential event-list implementation. But the efficiency of any allocation strategy is dependent on the metric, or measure, it uses to characterize the load in the distributed system. This paper presents a simple and intuitive way of measuring and reallocating the load when the objective is to minimize simulation run time. The metric, based on estimating measures of message utilization at each processor, has been used in an adaptive scheme for load allocation, and experiments on an iPSC / 2 Hypercube indicate that it successfully characterizes the load for purposes of reducing simulation run time.

With advances in parallel machines, Distributed Simulation (DS) has become a viable way of dealing with time-consuming simulations. An important factor affecting the performance of DS models, like the Chandy and Misra model for DS,[6, 7, 22] is the allocation of logical processes (LPs) to available (physical) processors. The objective in making an assignment is to reduce the realized run time of the simulation. An inefficient assignment can result in excessive communication times, bottleneck processors, and unfavorable load conditions leading to long run times, possibly giving performance worse than that with a uniprocessor, sequential event-list implementation.

The problem of assigning LPs to processors is one instance of the task-allocation problem found in distributed systems, and it is NP-complete.[13] Similar problems exist in other fields. For example, in flexible manufacturing systems (FMS) we might seek to allocate a limited set of expensive tools to machine groups to maximize the throughput of jobs, where there is limited storage space between machines. A simple analogy to the current research is to view the tools as LPs, the machines as processors, the jobs as messages, and the limited space between machines as I/O buffer space. Another area is in managing traffic flow into and out of major roadways by controlling the entry points. In most instances, what is needed is a measure to define the congestion at an entry point so that access is controlled to maximize the flow. While it is harder to formulate this problem in terms of the current research, the congestion metric developed here could provide insight helpful in analysis and management of such systems. Various solution methods have been proposed for task-allocation problems, and they can be broadly classified into two groups: static schemes,[10, 14, 17, 19, 21] and dynamic schemes.[3, 11, 14, 20] Static schemes perform the allocation of tasks to processors once, typically at the beginning of the simulation, while dynamic schemes try to react to changes in the load on the system, and then make a reallocation. The LPs may be reallocated many times during the simulation using dynamic schemes. Static schemes are usually easier and less expensive (in terms of overhead) than dynamic schemes. But dynamic strategies are more appropriate for situations such as in DS where the load distribution is likely to change unpredictably during the simulation.

An essential element in all schemes is the *metric*, or measure, used to meet an objective. For example, Lu and Carey[20] use the sample variance of load distribution across the processors to balance the load on the system and minimize communication costs. In [10], Chu and Lan use intertask communication time and processor execution time as a measure to minimize the bottleneck processor utilization. Different metrics can be used to satisfy the same objective; the above two metrics could be used to balance the load in the system. However, the metrics and allocation strategies discussed in the literature are not particularly suited for *Distributed-Simulation Task-Allocation* (DSTA) problems. This paper develops a congestion metric, centered around message utilization at the processor, suitable for problems where the objective is to minimize the *run*

Subject classifications. Simulation. Efficiency.
Other key words: Distributed simulation.

time of the simulation. By run time we mean a *realization* that is a function of the underlying simulation and the strategy adopted for implementation. Thus for simulations using the same random numbers for identical purposes, different strategies could lead to different realizations of run time. Any strategy lowering this run time would then lead to a lower expected run time of the simulation.

The metric has been used in an adaptive scheme for task allocation,[26, 27] and experiments on an iPSC/2 Hypercube indicate that it successfully characterizes the load on the system for reducing simulation run time. While the DS model used in the experiments was the Chandy/Misra model,[6, 7, 22] it is felt that the metric can also be adapted to task allocation in "optimistic approaches" such as the time-warp mechanism.[15]

The next section describes the DSTA problem. Section 2 summarizes the results of load allocation strategies from the current literature, while Section 3 presents the congestion metric. Sections 4 and 5 discuss its applicability for dynamic task-allocation schemes. A brief overview of the implementation of the congestion metric in an adaptive scheme is given in Section 6. Results and conclusions are presented in Sections 7 and 8, respectively.

1. The Distributed Simulation Task-Allocation Problem

A particular assignment of LPs to processors can be represented as a mapping M (M: $LP_i \rightarrow P_j$, $i = 1, \ldots, c$, $j = 1, \ldots, d$), where c is the number of LPs in the model, and d is the number of processors available (the mapping is read as assigning LP i to processor j). The objective is to find a mapping to minimize the run time of the simulation.

To execute the simulation on a parallel-processing machine we develop a mapping such that processor P_j will execute all messages corresponding to the LPs assigned to it in the mapping. The arrival of a message to LP_i results in additional work to be executed at that LP, or on processor P_j to which LP_i is assigned. The execution time for a message on all processors is assumed to be positive but finite.

Messages originate at LPs called *source LPs*. They are then sent to the next LP in the logical structure where they are processed, and then sent to the next LP, etc., until finally they leave the system. The last LP visited before a message leaves the system is called the *sink LP*.

As an example, consider the logical system in Figure 1. Figure 2 shows a mapping to four identical processors. The communication time between processors is assumed to be negligible (see Section 5.1 for a relaxation of this assumption). Here, messages originate from LP_1 (the source), and leave the system from LP_{10} (the sink). LP_3 is a branch point, with messages going to LP_4 and LP_8 with equal (and independent) probabilities.

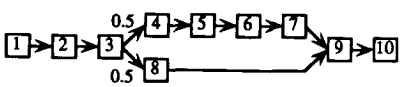

Figure 1. A logical system with 10 LPs.

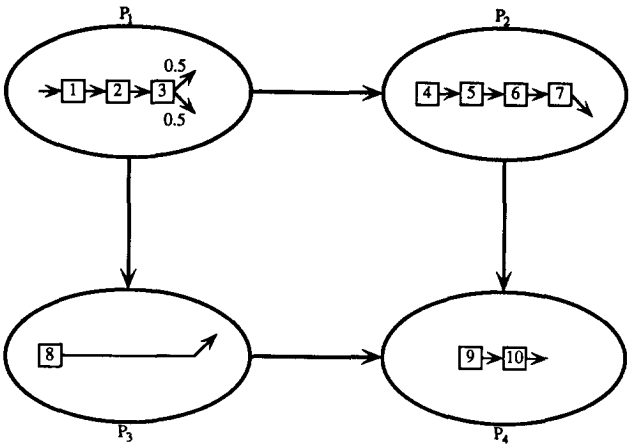

Figure 2. A possible assignment of LPs to processors. Beginning LPs: 1, 4, 8, 9. Ending LPs: 3, 7, 8, 10.

For such systems, the time spent by a message in the system depends not only on the processor execution time and communication times, but also on the congestion at the processors and the precedence relationship to be satisfied for each message. Congestion and precedence relationships influence the waiting time. The precedence relationship for a message is determined by factors like the location of the message in the network, the message type, the time stamp of the message, the channel in which the message arrived, the queue discipline for messages, and the characteristics of the physical system being simulated.

A simulation is usually stochastic in nature, which results in a varying number of events for each run. For a distributed simulation, it results in an unknown number of messages (and hence tasks) at each LP for each run of the simulation. Therefore, in a distributed scheme the load (total amount of work) on the system cannot be determined before the simulation is started, and the load on each LP may change substantially throughout the simulation.

Thus, the total time at LP_i (i.e., on P_j) for a message m depends on the mapping as well as the amount of work needed for that message to be processed, and any mapping seeking to minimize the run time must look beyond execution times.

The performance of the simulation is also affected by the characteristics of the host computer. While factors such as the speed of the processors and communication channels directly affect run time, the effect of a factor like the buffer space available for pending messages is more subtle. A full buffer at a processor may cause the computer (or the operating system) to resort to one of two actions: Processors may be blocked from further processing until the buffer space is cleared, or the simulation may abort. In either case, time is wasted, leading to a deterioration in the run-time performance of the simulation. While a full buffer at a processor usually indicates that the arrival rate λ of messages is greater than the service rate μ at that processor, in DS a full buffer could occur even when $\lambda < \mu$. This is because precedence relationships have to be satisfied at

each LP. It is therefore important to recognize and correct such unstable situations if distributed simulations are to run efficiently on parallel-processing computers.

A mapping of LPs to processors with the objective of minimizing the run time of a simulation should thus consider the following:

- The exact number of messages, and hence work, to be processed by the system is not known beforehand.
- The load on each LP can vary from run to run.
- Precedence relationships to be satisfied for a particular message at LP_i depend on the location and clock values of the messages, and on the structure of the physical system.
- The characteristics of the computer being used.

The *Distributed-Simulation Task-Allocation* problem (to distinguish it from earlier task-allocation problems) can then be defined as: Given a computer system with a fixed number of processors P_j, $j = 1, \ldots, d$ (not necessarily homogeneous), a logical system with c logical processes, LP_i, $i = 1, \ldots, c$, find a mapping (M: $LP_i \rightarrow P_j$, $i = 1, \ldots, c$, $j = 1, \ldots, d$), dynamic or static, such that the run time of the simulation is minimized.

2. Strategies for Task Allocation

As mentioned earlier, solutions to task-allocation problems (TAPs), sometimes referred to as *load-sharing* or *load-balancing* strategies, can be broadly classified into two categories: static or dynamic. A *static* decision is made independent of the current state, while a *dynamic* decision may depend on the state of the system. While many static load-sharing strategies have been proposed for distributed systems,[10, 14, 25, 30–32] their potential is limited for DSTA since they do not adapt to the unpredictable and time-varying changes in system load. On the other hand, dynamic schemes are based on the system state, and nodes (processors) can transfer task(s) to other nodes to make efficient use of available resources.[4, 5, 11, 14, 18, 20, 23, 29] Since dynamic schemes need more information, they are inherently more complex and involve additional computational overhead. Other research on load sharing includes theoretical studies,[9, 24, 25] developing metrics for task allocation,[12, 20] and proposing efficient strategies.[8, 11, 16, 28, 29, 33]

The results of previous studies on finding solutions for task-allocation problems 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, and unlike previous studies where strategies are set up for externally generated tasks (messages), in DS the rate of message generation depends on the strategy in effect. Other complicating factors in DS include unknown precedence relationships, reallocation of LPs rather than individual messages, and the fact 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.

The next section presents a congestion metric that can be adapted to different computer systems and captures the essential characteristics of the logical and computer system used.

3. A Congestion Measure for Task Allocation

We define the *message utilization* π_j of processor j as

$$\pi_j = \lambda_j/\mu_j \quad j = 1, \ldots, d$$

where

λ_j = arrival rate of *new messages* (see below) to processor j

$= 1/\text{E(interarrival time)}$

μ_j = service rate of messages on processor j

$= 1/\text{E(service time)}$.

A message m_{ik} from LP_i to LP_k ($i, k \in \{1, \ldots, c\}$), i.e., from P_{LP_i} to P_{LP_k} (where P_{LP_i} indicates that LP_i is assigned to P_{LP_i}), is a *new message* to P_{LP_k} if either of the following conditions is satisfied:

1. $P_{LP_i} \neq P_{LP_k}$ or
2. m_{ik} was generated at LP_k, i.e., LP_k is a source LP. In this case, $LP_i = LP_k$.

In Figure 2, a new message to P_2 is any message from LP_3 to LP_4 (condition 1). For P_1, a new message is a message generated at the source LP, LP_1 (condition 2).

A new message to P_j may visit many LPs on that processor before leaving it. At each LP the message experiences a delay. Delay occurs because of waiting in queue until a message's precedence relationships are satisfied and the processor is ready to execute it, and because of the execution time incurred at that LP. The total execution time incurred by a message m_{ik} on P_j is the sum of the execution times incurred at the various LPs visited by that message on P_j. Then $\psi_j = 1/\mu_j$ is the expected execution time incurred by a new message to P_j. In Figure 2, the expected execution time ψ_2 for a new message to P_2 is the sum of expected execution times incurred at LP_4, LP_5, LP_6, and LP_7.

The metric leads to the observation that, given λ_j and μ_j, the maximum departure rate δ_j of messages from P_j can be predicted. It is intuitive that 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, we can write

$$\delta_j \leq \min(\lambda_j, \mu_j). \tag{1}$$

As discussed in the following sections this observation has ramifications for allocation schemes seeking to minimize the run time of a simulation.

3.1. Characteristics

Two parameters must be evaluated each time dynamic reallocation is done: the cost of making an allocation, and the benefit that can be achieved due to the reallocation.

Cost: To carry out a dynamic scheme, time is spent in collecting statistics, calculating the new allocation, and in reallocating the load in the system. As dynamic reallocation is done during run time, each instance of load reallocation increases the simulation's run time. Thus the cost in using the dynamic scheme is the time spent in implementing it.

Benefit: The benefit in using a dynamic scheme is the potential improvement in run time that can be achieved by reallocating the load (compared to what would happen if the load were not reallocated).

To determine the cost effectiveness of a scheme we must know the benefit and cost of implementing it each time load is reallocated. While the cost can be estimated accurately, it is much harder to determine the benefit of using a dynamic scheme. Run time is an intangible objective, as it is known only at the end of the simulation. An equivalent but more tangible measure is δ_s, the departure rate of messages from the system. As run time is related inversely to δ_s, an assignment that increases this departure rate will produce a reduction in run time, if the cost of making the assignment 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.

Some characteristics of the metric and Equation 1 that are useful in determining the cost effectiveness of a dynamic allocation scheme are listed below.

- π_j reflects the load on processor j. If $\pi_j > 1$ it implies that P_j is receiving messages faster than it can process them. While this is an unstable condition, it does arise in distributed-simulation environments and it is important to recognize and correct such situations if simulations are to run effectively. One strategy may be to reduce the load by moving some LPs away from P_j.
- The metric can be used to determine the benefit of an allocation. As messages leave the system from processors containing sink LPs, δ_s can be expressed as the sum of departure rates of messages from processors containing those sink LPs. For example, in Figure 2, $\delta_s = \delta_4$. But δ_j can be predicted given λ_j and μ_j. Thus if λ_j and μ_j are known for two assignments, the better one can be chosen by determining the relative benefits (i.e., by observing δ_s) of the assignments. For example, Figures 2 and 3 show two possible assignments. By determining δ_4 (P_4 contains LP_{10}, the sink LP) for both assignments, the one that leads to a lower run time (higher δ_4) can be found. Note that to determine the cost effectiveness of a proposed assignment during the simulation, we have to predict the as-yet-unobserved values of λ_j and μ_j under this new assignment.

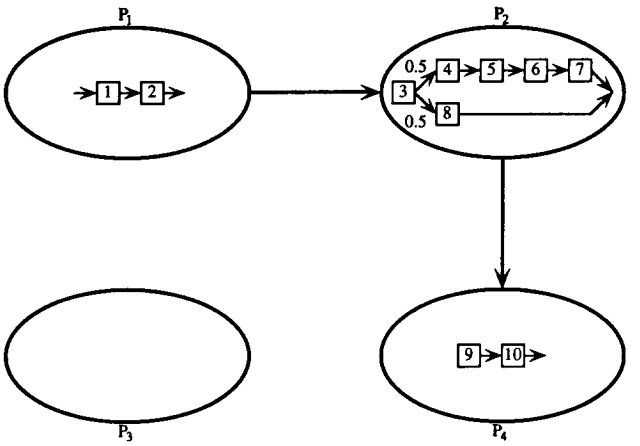

Figure 3. An alternative assignment of LPs to processors. Beginning LPs: 1, 3, 9. Ending LPs: 2, 7, 8, 10.

- In many task-allocation problems load is allocated so as to maximize the utilization of the processors. In such cases $\pi_j \approx 1, \forall j$. While such conditions are considered ideal, the particular assignment may not be efficient. An *efficient* assignment, when the objective is to minimize run time, will be conducive to processing messages at the earliest possible times. That is, the assignment will seek to reduce the waiting time in queue for messages. There are two main reasons why messages in DS wait in queue: to satisfy precedence relationships, and to wait for service time at the processor. Of the two, the former is usually more difficult to correct, as the latter can be adjusted for by decreasing the load on that processor, thereby increasing the effective service rate. The departure rate δ_j can be used to detect conditions where precedence relationships are not being satisfied in a timely manner. As the departure rate of messages from a processor is limited by the arrival and services rates at that processor, under ideal conditions, $\delta_j = \min(\lambda_j, \mu_j)$. But because of the overhead involved in processing a message and in collecting statistics to measure π_j and δ_j, $\delta_j \approx \min(\lambda_j, \mu_j)$ (usually). We are differentiating between theory and practice. In theory it is possible that $\delta_j = \min(\lambda_j, \mu_j)$. But when δ_j is calculated from observed values it is unlikely that the equality will hold. The *goodness* of an assignment can then be found by observing δ_j. If $\delta_j \ll \min(\lambda_j, \mu_j)$ it could suggest that messages are spending excessive time waiting to be processed because precedence relationships are not being satisfied (see Figure 4), a direct result of an inefficient assignment. Thus, even when $\pi_j = 1$ (arrival rate = service rate), the assignment may not be an efficient one. Reallocation schemes need to account for such situations if simulations are to run efficiently.

3.2. An Alternative Representation for λ_j and μ_j

An important use of the metric is in adaptive schemes for load reallocation. To facilitate its use in such schemes, we

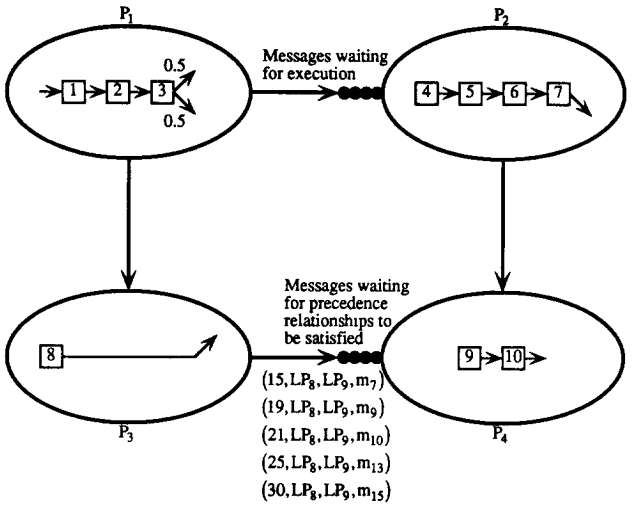

Figure 4. Messages waiting for precedence relationships to be satisfied at LP_9.

present an alternative, but equivalent, form for calculating λ_j and μ_j. In this new representation, λ_j and μ_j are expressed as a function of "load" on LPs instead of load on processors.

For a fixed assignment, let LP_i on P_j be a *beginning* LP if

1. $LP_i \rightarrow P_j$ (meaning that LP_i is assigned to P_j)
2. LP_i receives messages from LP_k, and $LP_k \rightarrow P_l$, $l \neq j$, or
3. LP_i is a source LP.

In Figure 2, LP_4 is a beginning LP on P_2, while LP_8 is a beginning LP on P_3. Note that a beginning LP is a recipient of new messages.

Also, let LP_i be an *ending LP* on P_j if

1. $LP_i \rightarrow P_j$
2. LP_i sends messages to LP_k, and $LP_k \rightarrow P_l$, $l \neq j$, or
3. LP_i is a sink LP.

Thus, LP_7 is an ending LP on P_2, and LP_8 is an ending LP on P_3. LP_8 is both a beginning and an ending LP. By definition, beginning and ending LPs depend on the assignment in effect.

Then λ_j can be expressed as the sum of arrival rates of new messages to beginning LPs on P_j. In Figure 2, λ_2 is the arrival rate of messages to LP_4 from LP_3, and λ_4 is the sum of arrival rates of messages to LP_9 from LP_7 and from LP_8. In general, let κ_{mn} be the arrival rate of messages to LP_n from LP_m. Also, if LP_n is a source LP let κ_{nn} be the rate at which messages are generated from it. Then,

$$\lambda_j = \sum_{LP_m \nrightarrow P_j} \sum_{LP_n \rightarrow P_j} \kappa_{mn} + \sum_{LP_n \rightarrow P_j} \kappa_{nn} \qquad (2)$$

(κ_{nn} will be 0 if LP_n is not a source LP). Thus, the arrival rate to a processor can be characterized as a function of arrival rates to beginning LPs.

As ψ_j is the expected execution time for a message on P_j, it can be calculated by knowing the expected execution time for a message on each LP in P_j, and the probability of choosing one of many paths (if more than one exists) in P_j (because the path determines the number of LPs a message will visit, and hence the execution time for that message). Once a path has been chosen the execution time for a message is the sum of the execution times incurred at the LPs on that path.

In Figure 3, two paths exist within P_2. Path 1 consists of LP_3, LP_4, LP_5, LP_6, and LP_7, and path 2 has LP_3 and LP_8. The expected execution time for a message on path 1 is $\sum_{k=3}^{7} \nu_k$, and on path 2 is $\nu_3 + \nu_8$, where ν_k is the expected execution time for a message on LP_k. The expected execution time for a new message to P_2 can then be written as [prob(path 1) × expected execution time on path 1 + prob(path 2) × expected execution time on path 2]. For our example, $\psi_2 = 0.5 \times \sum_{k=3}^{7} \nu_k + 0.5(\nu_3 + \nu_8)$. In general, if there are l_j paths in P_j, and if α_i is the probability of a message taking path i in P_j ($\sum_{i=1}^{l_j} \alpha_i = 1, \forall j$), and

$$\beta_{ki} = \begin{cases} 1 & \text{if } LP_k \in \text{path } i \\ 0 & \text{otherwise} \end{cases}$$

then

$$\psi_j = \sum_{i=1}^{l_j} \alpha_i \sum_k \beta_{ki} \nu_k. \qquad (3)$$

The processor and path to which α_i and β_{ki} refer should be clear from context.

As an example, consider evaluating δ_j for the assignment in Figure 2. Let the expected execution time for a message on the various LPs be $\nu_k = 1$ for $k = 1, 2, 3, 7, 8, 9, 10$, and $\nu_k = 2$ for $k = 4, 5, 6$. As the processors are identical the execution times are independent of the assignment. From Equation 3

$$1/\mu_j = \sum_i \alpha_i \sum_k \beta_{ki} \nu_k.$$

In Figure 2 each processor has only one path within it that a message can take. Therefore, $\alpha_1 = 1, \forall j$. Thus, in each processor $\beta_{ki} = 1$, i.e., a new message to P_j will be executed on all LPs in P_j. The values of $\mu_j, \forall j$, can therefore be calculated. For example, in P_1 the path is LP_1, LP_2, and LP_3. Also, from the values of the ν_k's and Figure 2, $1/\mu_1 = 1 + 1 + 1 = 3$. Similarly, $1/\mu_2 = 2 + 2 + 2 + 1 = 7$, and $1/\mu_3 = 1$. P_4 receives messages from P_3 and P_2, but since each message to P_4 takes the same path (i.e., LP_9 and LP_{10}), $1/\mu_4 = 1 + 1 = 2$.

To calculate λ_j, we note from Figure 2 that LP_1, LP_4, LP_8, and LP_9 are beginning LPs on P_1, P_2, P_3, and P_4, respectively. From Equation 2 and Figure 2, λ_1 is the rate of generation of messages at LP_1. If P_1 uses the algorithm in Table I to generate new messages, $\lambda_1 = \mu_1$. Therefore, $\lambda_1 = 1/3$. Assuming for simplicity that the assignment in Figure 2 is "good" (refer to Section 3.1), i.e., $\delta_1 = \min(\lambda_1, \mu_1)$, $\delta_1 = \lambda_1 = 1/3$. Therefore, $\lambda_2 = \lambda_3 = 1/2 \times 1/3 = 1/6$, assuming negligible communication times. Since $\mu_2 = 1/7$, $\pi_2 = 7/6$, and thus $\delta_2 = \min(\lambda_2, \mu_2) = 1/7$. Similarly λ_j, μ_j, and $\delta_j, \forall j$, can be calculated. Table II and Figure 5 show the calculated values. For this assignment LP_{10} is the sink LP, so $\delta_s = \delta_4 = 13/42$.

4. Predicting δ_j

As mentioned earlier, each time load is dynamically reallocated it is important to determine the cost effectiveness of the reallocation. While cost can be measured accurately, it is much harder to gauge the benefit. This section describes how the departure rate δ_s of messages for a current assignment M and a proposed assignment M^1 can be predicted during the simulation. The change in δ_s (between the current and the proposed assignment) can then be used to determine the benefit.

Let λ^1 and μ^1 denote the arrival and service rates of messages, respectively, if M^1 is implemented. We consider two cases: When the actual future values of λ^1 and μ^1 can be determined at a point in time during the simulation, i.e., when future values of parameters are known, and when λ^1 and μ^1 have to be estimated from observations collected under the current allocation M, i.e., future values have to be estimated from past data.

Table I. Algorithm to Generate New Messages

```
if messages are waiting then
  process message
else
  generate another message
endif
```

Table II. Departure Rate of Messages under M

j	k	ν_k	ψ_j	μ_j	λ_j	δ_j	π_j
1	1	1	1 + 1 + 1 = 3	1/3	1/3	1/3	1
	2	1					
	3	1					
2	4	2	2 + 2 + 2 + 1 = 7	1/7	1/6	1/7	7/6
	5	2					
	6	2					
	7	1					
3	8	1	1	1	1/6	1/6	1/6
4	9	1	1 + 1 = 2	1/2	13/42	13/42	13/21
	10	1					

$\delta_s = \delta_4 = 13/42$.

4.1. Known Parameters

Consider simulation of the logical system shown earlier (Figure 1). Let the current assignment be (M: $LP_1, LP_2, LP_3 \rightarrow P_1$; $LP_4, LP_5, LP_6, LP_7 \rightarrow P_2$; $LP_8 \rightarrow P_3$; $LP_9, LP_{10} \rightarrow P_4$) (Figure 2). For illustration purposes let the possibly new assignment be (M^1: $LP_1, LP_2, LP_3 \rightarrow P_1$; $LP_4, LP_5, LP_6 \rightarrow P_2$; $LP_8 \rightarrow P_3$; $LP_7, LP_9, LP_{10} \rightarrow P_4$) (Figure 6). Then a probable set of values for λ^1 and μ^1 is shown in Figure 7. While these values were calculated using the approach shown in Section 3.2 and assuming that the same execution times were valid under M^1, it is assumed that these are indeed the actual values of λ^1 and μ^1 under M^1. As LP_{10} is the sink LP, $\delta_4^1 = \delta_s^1 = 1/3$, where δ_s^1 is the departure rate of messages from the system under M^1. It has again been implicitly assumed that $\delta_j^1 = \min(\lambda_j^1, \mu_j^1)$, which may not be true. In any case, δ_s^1 has increased by 1/42 over the current assignment M (compare with δ_s in Figure 5). The cost effectiveness of the new assignment can then be determined by knowing the cost of implementation.

In most cases it would be impossible to determine λ^1 and μ^1 beforehand. This is because of the changing conditions of a distributed simulation and the difficulty in determining a priori the effect of an assignment on factors like precedence relationships and the path of a message. But if the future values of parameters are known the cost effectiveness of a scheme can be readily established.

4.2. Estimated Parameters

In presenting an estimate of the parameters, we will use the following convention: a superscript "1" will denote a value under M^1 (Figure 6). Thus, μ_j^1 would be the service rate of

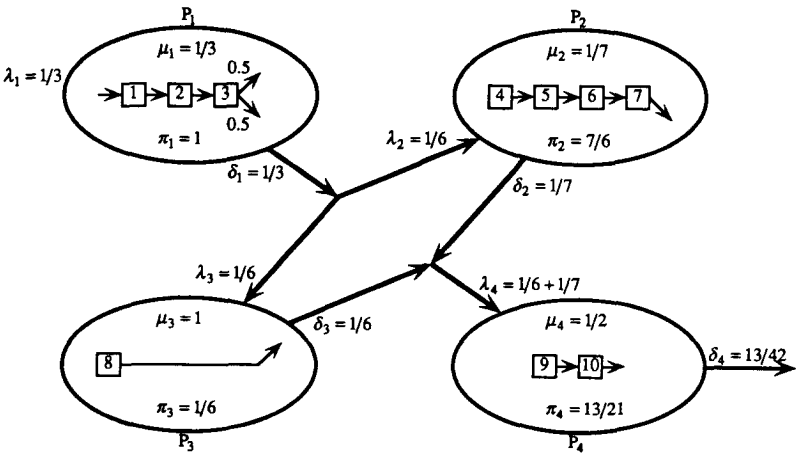

Figure 5. Departure rate of messages under M.

P_j under M^1. When values are estimated from past data we will represent the estimator with a $\hat{}$, so $\widehat{\mu_j^1}$ would be the estimated value of μ_j^1.

We assume in determining $\widehat{\lambda^1}$ and $\widehat{\mu^1}$ that the simulation at any point will behave like its recent past. This means that observations collected under the current allocation represent the future, i.e., arrival rate of messages between LPs and the service rate of messages at LPs will remain unaffected by a different assignment. While this assumption could be violated in many instances, we feel that λ^1 and μ^1 can still be estimated satisfactorily for purposes of choosing an alternative assignment. Also, as shown later, proportionality constants can be included to account for factors such as heterogeneous processors and varying communication rates.

As an example, consider determining $\widehat{\lambda_j^1}, \forall j$, for the assignment M^1 of the previous section (Figure 6). The current assignment M is shown in Figure 2. Assume that the data collected under M until time t is as given in Tables III and IV. Table III contains the cumulative time between arrival (CTBA) of messages to LPs under M. Thus, if TBA_{ijm} is the time between arrival of the mth and $(m-1)$st message to LP_j from LP_i, then $CTBA_{ij} = \sum_{m=2}^{N} TBA_{ijm}$, where N is the number of messages that arrived at LP_j from LP_i (empty sums are taken to be zero). Also, the cumulative execution time for a message at an LP (CET) and the number of messages executed at each LP are given in Table IV. From these data, consider calculating $\widehat{\lambda_4^1}$. In M^1 (Figure 6) LP_7 is a beginning LP on P_4, and now messages to LP_9 from LP_7 no longer constitute new messages. To get $\widehat{\lambda_4^1}$ we need the arrival rate of messages to LP_7 (beginning LP) and to LP_9 from LP_8. As the recently observed values are representative of the future, we can use the arrival rate of messages to LP_7 and to LP_9 from LP_8 from the current allocation (Table III) to estimate $\widehat{\lambda_4^1}$. For example, $\widehat{\lambda_4^1} = 1/6.82 + 1/6 = 1/3.19$. Similarly, λ_3^1 is the arrival rate of messages to LP_8 and can be estimated as $\widehat{\lambda_3^1} = 1/6$. Table V lists the calculated values of $\widehat{\lambda_j^1}, \forall j$. The values of λ_j^1 are also listed for comparison (from the previous section). In this case, $\widehat{\lambda_4^1}$ estimates λ_4^1 to within 6% $[100(1/3 - 1/3.19)/(1/3)]$, if the values of λ_j^1 are indeed correct.

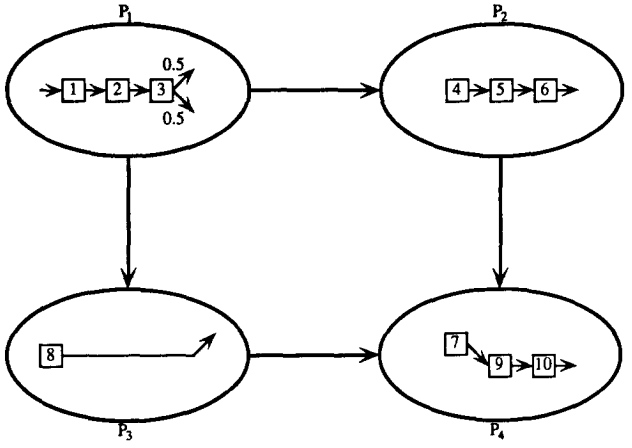

Figure 6. Assignment M^1.

To determine $\widehat{\mu_j^1}$ we need ν_k^1, the expected execution time at $LP_k \in P_j$, and the probability of choosing each path in P_j under M^1. For example, compare Figures 2 and 3. For the same logical system the path of a message in P_2 is different under the two assignments. In most simulations not only are the different paths difficult to determine, but also the probability of choosing a particular path is seldom known. This is because the path of a message is constantly updated as it proceeds from one LP to another, and this path changes when the assignment changes. Thus, in most cases it is not possible to determine the different paths, let alone the probability of choosing a particular path, for an

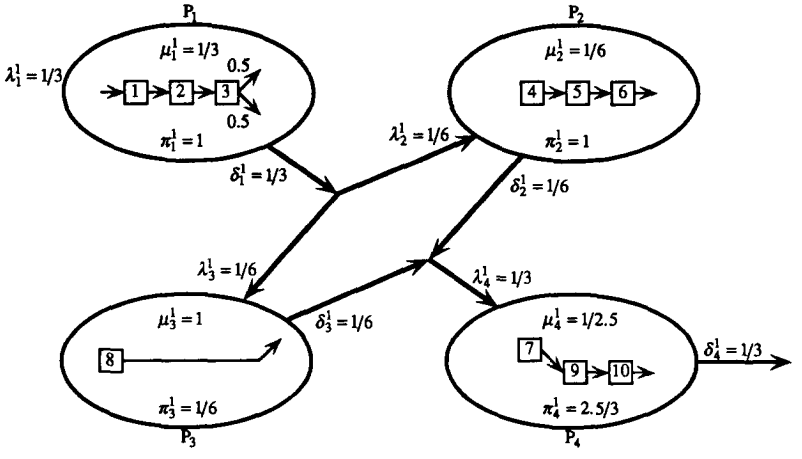

Figure 7. Departure rate of messages under M^1.

Table III. Average Arrival Rate of Messages to LPs

Origin LP	Destination LP	CTBA	# Arrivals $(N-1)$	Arrival rate = # Arrivals/CTBA
—	1	750	250	1/3
1	2	750	250	1/3
2	3	750	250	1/3
3	4	750	125	1/6
3	8	750	125	1/6
4	5	750	110	1/6.82
5	6	750	110	1/6.82
6	7	750	110	1/6.82
7	9	750	110	1/6.82
8	9	750	125	1/6
9	10	750	235	1/3.19
10	—	750	235	1/3.19

CTBA = Cumulative time between arrival of messages.

Table IV. Execution Time Observed at Each LP Under M

Processor	LP_k	CET_k	n_k	$\nu_k = \widehat{\nu_k^1} = n_k/CET_k$
1	1	250	250	1
	2	250	250	1
	3	250	250	1
2	4	220	110	2
	5	220	110	2
	6	220	110	2
	7	110	110	1
3	8	125	125	1
4	9	235	235	1
	10	235	235	1

CET_k = Cumulative observed execution time incurred at LP_k. n_k = Number of messages executed at LP_k. ν_k = Average execution time for a message at LP_k. $\widehat{\nu_k^1}$ = Predicted average execution time at LP_k under M^1.

assignment that has not yet been implemented. As μ_j^1 is to be estimated from past data the following approach is taken:

Using Equation 3 we can write

$$1/\mu_j^1 = \sum_i \alpha_i^1 \sum_k \beta_{ki}^1 \nu_k^1 = \sum_k \nu_k^1 \sum_i \alpha_i^1 \beta_{ki}^1 = \sum_k \nu_k^1 \Phi_{kj}^1$$

where

Φ_{kj}^1 = The probability that a message is executed at $LP_k \in P_j$ under M^1

Table V. Estimated Arrival Rate of Messages under M^1

Processor	Beginning LP	Arrival Rate to Beginning LP	$\widehat{\lambda_j^1}$	λ_j^1
1	LP_1	1/3	1/3	1/3
2	LP_4	1/6	1/6	1/6
3	LP_8	1/6	1/6	1/6
4	LP_7	1/6.82		
	LP_9	1/6	1/3.19	1/3

$$\beta_{ki}^1 = \begin{cases} 1 & \text{if } LP_k \in \text{path } i \\ 0 & \text{otherwise} \end{cases}$$

α_i^1 = Probability of a message choosing path i

ν_k^1 = Expected execution time for a message on LP_k.

Note that β_{ki}^1 and α_i^1 are defined relative to M^1. Therefore μ_j^1 can be determined by knowing ν_k^1, the expected execution time for a message at LP_k under M^1, and Φ_{kj}^1, the probability of a message being executed at LP_k. Since ν_k^1 and Φ_{kj}^1 are unknown we proceed as follows:

Under the assumption that recent data are representative of the future, ν_k^1 is estimated by $\nu_k = \widehat{\nu_k^1}$, the average of the observed times for a message at LP_k under M. For our example $\widehat{\nu_k^1}$ can be calculated as shown in Table IV. To estimate Φ_{kj}^1 we note that

Φ_{kj}^1 = Probability of a message being executed at LP_k

$$\widehat{\Phi_{kj}^1} = \frac{\text{Number of messages } \textit{that will} \text{ be executed at } LP_k \text{ under } M^1}{\text{Number of new messages to } P_j \text{ under } M^1}$$

$$= \frac{\text{Number that will be executed at } LP_k \text{ under } M^1}{\text{Number of messages to beginning LPs in } P_j \text{ under } M^1}$$

$$= \frac{\text{Number } \textit{that were} \text{ executed at } LP_k \textit{ under } M}{(\text{Number of messages to LPs under M that are now beginning LPs in } P_j \text{ under } M^1)}$$

$$= n_k/\widehat{NM_j^1}$$

where

n_k = Number of messages that were executed at LP_k under M

$\widehat{NM_j^1}$ = Predicted number of new messages to P_j under M^1 (calculated from data observed under M).

Thus,

$$1/\widehat{\mu_j^1} = \sum_k \widehat{\nu_k^1}\widehat{\Phi_{kj}^1}$$

where

$\widehat{\nu_k^1}$ = Average execution time observed for a message at LP_k under M

$\widehat{\Phi_{kj}^1}$ = The estimated probability of a message being executed at LP_k.

To calculate $\widehat{\Phi_{kj}^1}$ we need $\widehat{NM_j^1}$ and n_k. Since n_k is the number of messages executed at LP_k under M, it is known. NM_j^1 is the number of new messages to P_j under the proposed assignment M^1. Since new messages to P_j are messages to beginning LPs in P_j, NM_j^1 can be determined by observing the beginning LPs in P_j under M^1 and the number of messages that arrive to such LPs. Because future values are not known, NM_j^1 is estimated by knowing the number of messages that arrived to the above-mentioned LPs (beginning LPs) in the current allocation M (Table III). For example, consider determining $\widehat{NM_1^1}$. The beginning LP in P_1 under M^1 is LP_1 (Figure 6). An estimate of the number of new messages to P_1 is therefore the number of messages generated at LP_1 under M. From Table III this is equal to 250. Similarly, $\widehat{NM_2^1}$ is the number of messages that arrived to LP_4 (a beginning LP in P_2 under M^1) from LP_3 under M. From Table III, $\widehat{NM_2^1} = 125$. Once $\widehat{NM_j^1}$ is known, $\widehat{\Phi_{kj}^1}$ can be calculated. Table VI shows the results.

Tables VII and VIII show algorithms that can be used to estimate λ^1 and μ^1 from data collected under the present assignment.

4.3. Limitations

λ^1 and μ^1 are estimated based on the assumption that past data represent the future even when a reallocation is done. The purpose in estimating is to determine the benefit of a proposed assignment *during* the simulation. While this benefit may not be accurately determined when the assumption (that past data represent the future) is violated, the estimated values may still be used to determine a better assignment, i.e., to choose between the current allocation and a proposed assignment. It is obvious that the greater the departure from this assumption, the less accurate our estimate is likely to be. Since observed values are usually dependent on the assignment, it is possible that the assumption could be violated. In such cases an error is introduced in the estimation, and usually a greater degree of error is introduced in estimating λ_j^1 than in μ_j^1. For our example, $\widehat{\lambda_4^1}$ underestimates λ_4^1 by 6% [100(1/3 − 1/3.19)/ (1/3)], while $\widehat{\mu_4^1}$ overestimates μ_4^1 by 1.3% [100(1/2.5 − 1/2.468)/(1/2.5)]. Generally, the service time

Table VI. Estimated Service Rate of Messages under M^1

P_j	LP_k	$\widehat{NM_j^1}$	n_k	$\widehat{\Phi_{kj}^1} = n_k/\widehat{NM_j^1}$	$\widehat{\nu_k^1}$	$\widehat{\mu_j^1}$	μ_j^1
1	*1	250	250	1	1	1/3	1/3
	2		250	1	1		
	3		250	1	1		
2	*4	125	110	0.88	2	1/5.28	1/6
	5		110	0.88	2		
	6		110	0.88	2		
3	*8	125	125	1	1	1	1
4	*7	235	110	0.468	1	1/2.468	1/2.5
	*9		235	1	1		
	10		235	1	1		

*Indicates a beginning LP.

Table VII. Algorithm for Estimating λ^1 from Observed Data

```
subroutine calculate_arrival_rate (p)
/*Determines the arrival rate of new messages to P_p
under M^1*/
/*arrival_rate(LP_k, LP_i) = observed arrival rate of
messages from LP_k to LP_i under M*/

For each beginning LP LP_i ∈ P_p DO
    For each predecessor LP_k of LP_i DO
        If [(P_LPk ≠ P_p) or (LP_k = LP_i)] then
            λ^1_p = λ^1_p + arrival_rate(LP_k, LP_i)
        endif
    end_DO
end_DO
end_routine
```

Table VIII. Algorithm for Estimating μ^1 from Observed Data

```
subroutine update_messages_to_processor (p)
/*Determines the number of new messages coming to
P_p under the proposed assignment M^1 based on the
data observed under M*/
/*jobs(LP_k, LP_i) = Number of messages that
arrived to LP_i from LP_k under M*/

NM^1_p = 0
For each LP_i ∈ P_p DO
    For each predecessor LP_k of LP_i DO
        If [(P_LPk ≠ P_p) or (LP_k = LP_i)] then
            NM^1_p = NM^1_p + jobs(LP_k, LP_i)
        endif
    end_DO
end_DO
end_routine

subroutine calculate_execution_time (p)
/*Determines μ^1_p*/
/*n_i = Number of messages executed at
LP_i under M*/

ψ^1_p = 0
For each LP_i ∈ P_p DO
    Φ^1_ip = n_i/NM^1_p
    ψ^1_p = ψ^1_p + Φ^1_ip × ν_i
end_DO
end_routine
```

for a message on an LP depends only on the processor to which the LP has been assigned, and not on the precedence relationship among LPs. When considering a new allocation the observed execution time can be adjusted to account for the difference in processor performance, and therefore it is possible to estimate μ^1 accurately.

On the other hand, estimation of λ^1 is dependent on both the assignment and the precedence relationships among LPs. As the strength of the relationship, i.e., the frequency and length of the messages, is also dependent on the assignment, an error is introduced in estimating λ^1. Also, in most cases, a correction factor cannot be determined since it is difficult to determine the strength of precedence relationships a priori.

Since the point in doing dynamic task allocation is to improve the run time of the simulation, each time a reallocation is considered the potential improvement in run time because of the new, but as-yet-unimplemented, assignment should be known to determine the cost effectiveness of the proposed allocation. The potential improvement is measured by the change in δ_s. While δ_s can be predicted by knowing $\widehat{\lambda^1}$ and $\widehat{\mu^1}$, an assumption is made that past data represent the future. Under this assumption, the effect of precedence relationships is captured in past data and hence it is possible to predict δ_s. It is much harder to predict δ_s when this assumption is violated. In the Chandy/Misra model, precedence relationships at LPs with multiple arrival streams (precedence relationships for messages are always satisfied at LPs with a single input stream) can be modelled as a superposition process with a queue discipline that is dictated by the timestamp of the messages in the various input queues. This is because an LP will not process a message until it is sure that the timestamp of the message is the *least* among all messages that the LP will receive. The problem is in predicting the throughput for such systems. While approximations exist for systems with general arrival streams with a general service distribution (i.e., $\Sigma GI/G/1$ systems), and a prespecified queue discipline like FIFO,[1, 2, 34–36] to our knowledge no results yet exist for systems with queue discipline as defined by precedence relationships like those in DS. Thus while it is easy to recognize when precedence relationships are not being satisfied (by observing δ_j), it may be difficult to correct them, especially when the assumption on past data is violated.

5. Extensions

Thus far we have assumed that the communication time between processors is negligible and that the processors are identical. This section describes ways to relax these assumptions.

5.1. Positive Communication Times

Consider a computer system with 2 processors P_i and P_j and a single communication channel Ω_{ij} from P_i to P_j similar to a single-server system with one input queue, with the channel itself being the server. The service (communication) time for a message depends on the type of message and the service rate of Ω_{ij}. An arriving message finding Ω_{ij} busy waits in queue until it can be serviced. Clearly, if $\lambda_{\Omega_{ij}}$ is the arrival rate of messages to Ω_{ij}, and $\mu_{\Omega_{ij}}$ is the communication rate, then the departure rate $\delta_{\Omega_{ij}}$ of messages from Ω_{ij} cannot exceed $\min(\lambda_{\Omega_{ij}}, \mu_{\Omega_{ij}})$. Thus, the congestion metric π can be applied to each channel between two processors in the same manner as it was applied to processors themselves. One difference is that messages coming into the communication channel have satisfied their precedence relationships (as far as the channel is concerned), so it is quite likely that $\delta_{\Omega_{ij}} \approx \min(\lambda_{\Omega_{ij}}, \mu_{\Omega_{ij}})$.

As an example, consider the logical system shown earlier in Figure 2, with the communication rate between all processors being 0.1. The effect of this on δ_s is shown in Figure 8. Thus, while determining a new assignment the effect of communication can be taken into account.

In many instances the communication rate will not be known, so it could be estimated assuming that we can collect observations between any two processors for each message type. Then the observed communication rates for

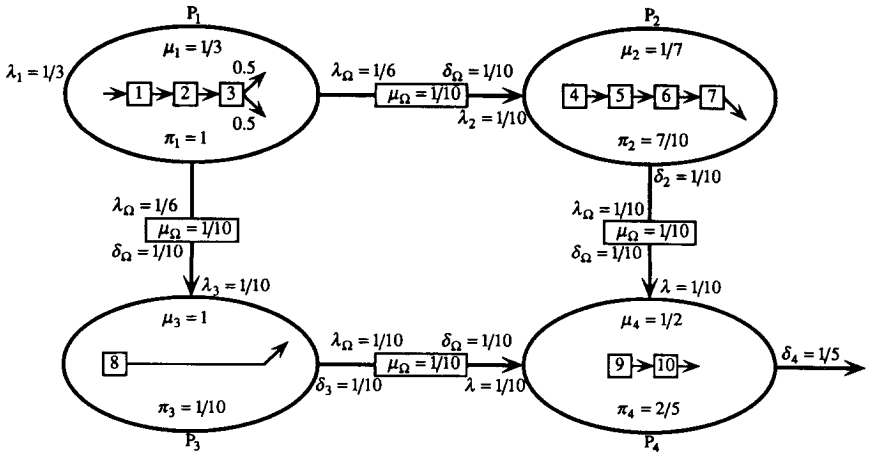

Figure 8. Effect of non-negligible communication time on δ_S.

each message type can be used in the calculation for a proposed allocation to determine the effect on δ_s.

5.2. Heterogeneous Processors

When the metric is used for estimating the effect of a proposed allocation an assumption is made that recently observed values represent the future. When heterogeneous processors are used this assumption is violated. Now the observed average service time for a message on an LP is valid only for its *current* processor assignment. To a certain extent, this problem can be alleviated if we knew the relevant performance characteristic of the processors in the system. Let τ_j represent the characteristic of interest for P_j. For example, let τ_j be the "speed" (in millions of instruction per second) of P_j. Then, when estimating μ^1 for a proposed allocation the observed execution times (under M) can be multiplied by the appropriate τ_j's to account for the difference in processor performance. For example, if LP_7 in Figure 2 is moved to P_4, the expected execution time for a message on that LP (assuming that everything else remained unchanged) would change to $\nu_7\tau_2/\tau_4$. In many instances it is easy to determine the relative speeds of the processors, and this can be taken into account while making a reallocation.

6. Implementation

An experimental study was conducted to evaluate the effectiveness of using the congestion metric in a dynamic scheme on tasks such as those found in distributed simulation. The measure of performance was the observed run time of the simulation. Three logical systems (Figures 9, 10, and 11) based on systems used in previous studies and suitable for simulating on the iPSC/2 Hypercube were considered.[26, 27] The following factors were included in the experimental design:

A The scheme used. The performance of the dynamic scheme[26, 27] was compared to a static strategy that was chosen to achieve the best possible run time for the simulation under the assumption that the initial load remained unchanged for the duration of the simulation.

B The 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 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.

Figure 9. Logical system 1. Source LP: 1.

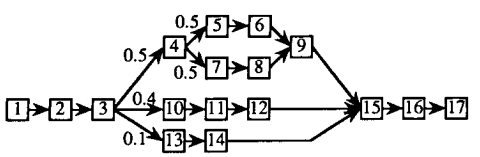

Figure 10. Logical system 2. Source LP: 1.

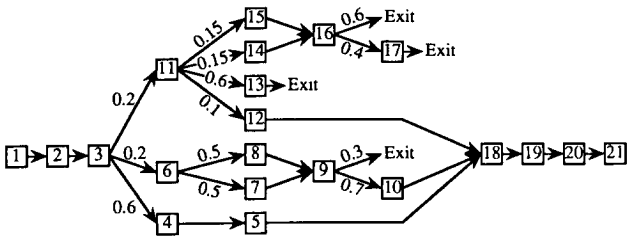

Figure 11. Logical system 3. Source LP: 1.

In addition, it was felt that the run length (i.e., the number of messages simulated) of the simulation, and the frequency of dynamic reallocations would affect the run time of the simulation. Technological constraints on the buffer space of the hypercube limited the maximum simulation run length to between 900 to 5000 messages depending on the logical system being simulated. A greater run length resulted in an error while running the simulations, especially when dynamic reallocation was *not* done.

Two different sets of experiments were conducted. The experiments differed 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 experiments there is still the overhead of actually implementing the reallocation, i.e., of moving LPs from one processor to another. In all cases a 2^4 full factorial design was used with the four factors mentioned previously, and all data generated were made independent by using nonoverlapping random-number streams. Each experiment was replicated until statistically significant results were obtained at the 90% confidence level. Further details of the implementation can be found in [26, 27].

7. Results

This section discusses the results of the experiments described in the previous section. The results of the experi-

ments, shown as a comparison between the dynamic and static scheme, are in Figures 12 through 14 and are described below. Note that each point in the above figures is the result of a pair of design points, and can be viewed as comparing the levels of A (scheme; see Section 6) with all other factor levels fixed. For example, the design point pair (1, 2) would represent a comparison between the dynamic and static scheme when factors B, C, and D are fixed at their low levels. Only the results from the first experiment are described since the results of the second experiment are qualitatively similar.

7.1. Experiment 1: Reallocation Using Only the Nodes

The run time achieved by using the adaptive scheme was in most cases substantially better than when running the simulation without the scheme. Figure 12 shows the percent change ($100 \times [d - s]/s$, where d and s are the average run time observed when using the adaptive and static schemes, respectively) in run time by using the dynamic scheme. For logical system 3 (LS_3), the run time improved by 30% to 50% by using the adaptive scheme, while for logical system 2 (LS_2) the improvement was between 20% to 44%. While the improvement for logical system 1 (LS_1) was not as great (Figure 12), it was nevertheless significant.

But in a few cases, for LS_1, the adaptive scheme in fact increased the run time (Figure 12). There were two main reasons for this increase: the topology of the logical system simulated, and the strategy used for task allocation. The adaptive scheme[27] attempts to improve the run time by taking a *localized* view of the problem. That is, tasks from processor P_j are reallocated to P_i 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 fails when there is a high degree of dependency among LPs (even if precedence relationships are satisfied immediately), as it is in LS_1 where the logical system is a pipeline model. 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 new assignment based on $\widehat{\delta_j^1}$ may lead to an inefficient reallocation.

Figure 13 shows the average threshold values for the three logical systems. If π_j^{ik} is the threshold value for P_j at the end of run i for design point k, then the average threshold value $\overline{\pi}^k = \sum_{j=0}^{d-1} \overline{\pi}_j^k / d$, where $\overline{\pi}_j^k = \sum_{i=1}^{N_k} \pi_j^{ik} / N_k$, 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 (Figure 13) the average threshold tends to be higher for the dynamic scheme, but for LS_2, not only did the adaptive scheme have a lower average threshold, but the balance of load among processors, i.e., the variance in threshold values among processors, is lower while using the dynamic scheme (Figure 14). 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 adaptive scheme achieved a better balance of load among the processors, leading to a lower run time when compared to the static strategy (Figure 14).

7.2. Effect of System Characteristics on Run Time

The above experiments show that the adaptive scheme was effective in reducing the run time of the simulation compared to using only the static scheme, and generally the scheme with a lower deviation in load among processors lead to the lower run time. This supports earlier studies where the variance of the load distribution among processors has been used as a measure to minimize the run time of the simulation.[20] At face value, this implies that irrespective of the problem and the computer system it is

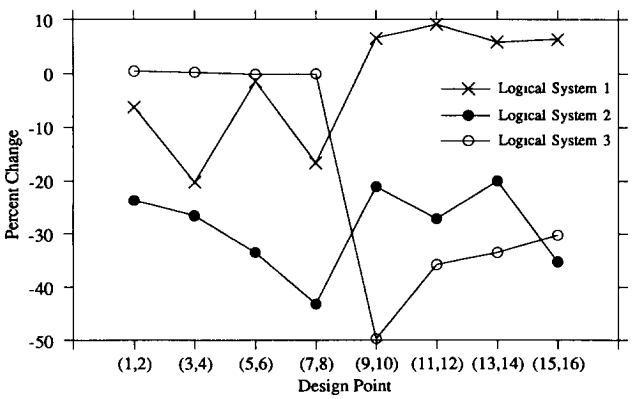

Figure 12. Percent change in run time.

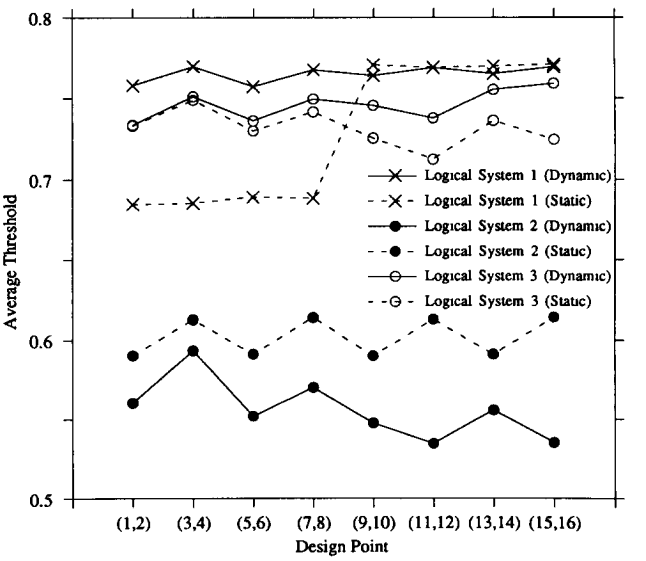

Figure 13. Average thresholds.

better to distribute the load uniformly across the available processors. Intuitively this would not be correct. The number of processors used should depend on the logical as well as the computer system. The following experiment uses Equation 1 to provide us with an intuitive way of assigning load to minimize the run time of the simulation.

LS_1 was considered for simulation. The average execution time for a message on all LPs was the same and $\approx$ 0.325ms. The communication time between processors was estimated to be $\approx$ 3ms. Five different static assignments were considered for simulation (Table IX). In each assignment, except one, 2-UB, the LPs were uniformly distributed over the available processors. In 2-UB (2 processors, unbalanced load), 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), and 4 and 8 too, the bottleneck is in the communication channels, a result of uniformly distributing the load. Specifically, in 2-UB, LPs are allocated to P_0 until $\delta_0 \approx 1/\zeta$, where ζ is the expected communication time between two processors. The remaining LPs are then assigned to P_1. The primary interest was to determine the relative performance of the above allocations.

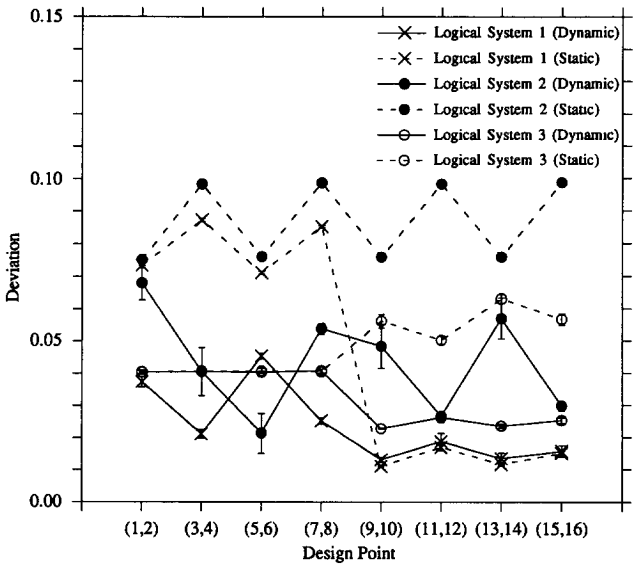

Figure 14. Average deviations in load.

Table IX. Approximate Values for δ_0

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

$1/\zeta \approx 0.3333$.

In the first set of experiments the simulation was run for 800 messages. The results are given in Figure 15. 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 after a period of time the I/O buffer gets 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 graphically by increasing the simulation's 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 (see Figure 15). Therefore, when communication time is important 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 7.1, a balanced load generally leads to a lower run time because the simulations were coarse grained, and hence communication was not the bottleneck.

8. Conclusions

While task-allocation problems in distributed systems have been studied extensively, as mentioned earlier, most problems make assumptions that limit their utility to distributed simulation. Thus, solutions developed in other fields are usually unsuitable (in terms of reducing the run time) for task allocation in distributed simulation. Also,

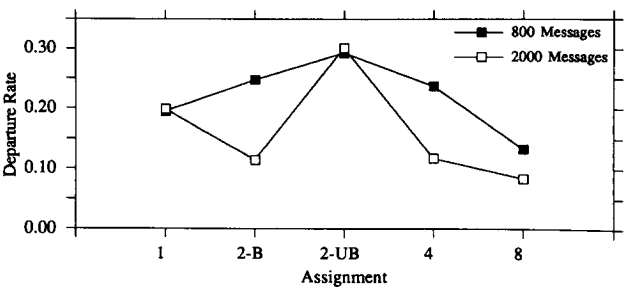

Figure 15. Average departure rate of messages from the system.

little empirical or theoretical work has been done in the area of dynamic task allocation in distributed simulation, modelers' generally relying on "guess work" to determine a good allocation, and that too a static one.

This paper has provided an intuitive and simple way of measuring the load during the simulation to minimize its run time. By studying the values of the congestion metric, it is possible to determine the effect of an allocation on the run time, thus giving a direct means of determining the benefit of a hypothetical assignment. This assures that, at least theoretically, it is possible to reallocate LPs during the simulation "optimally" (to reduce run time). From a practical standpoint, to use the property of determining the benefit, we need to assume that observations collected in the recent past represent the future. While this appears to be overly restrictive, the predictions (for determining the benefit) 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 congestion metric can be successfully used in providing a good estimate of the run time for a hypothetical assignment.

Acknowledgments

We would like to thank Tom Hoffmann, Chris Nachtsheim, and Sartaj Sahni for their valuable contributions to this work. Computational support was also received from several sources: Intel Scientific Computers (in particular, Randy Hufford and David Billstrom), University of Minnesota Academic Computing Services (especially Michael Frisch), the Minnesota Supercomputer Institute, and Argonne National Laboratory. We would also like to thank two anonymous referees for their comments that lead to important improvements of an earlier version of the paper.

References

1. S.L. Albin, 1982. On Poisson Approximations for Superposition Arrival Processes in Queues, *Management Science 28:2*, 126–137.
2. S.L. Albin, 1984. Approximating a Point Process by a Renewal Process, II: Superposition Arrival Processes to Queues, *Operations Research 32:5*, 1133–1162.
3. E. Andert, 1987. A Simulation of Dynamic Task Allocation in a Distributed Computer System in *Proceedings of the 1987 Winter Simulation Conference*, pp. 768–776.
4. K.M. Baumgartner and B.W. Wah, 1989. GAMMON: A Load Balancing Strategy for Local Computer Systems with Multiaccess Networks, *IEEE Transactions on Computers 38:8*, 1098–1109.
5. S.H. Bokhari, 1979. Dual Processor Scheduling with Dynamic Reassignment, *IEEE Transactions on Software Engineering SE-5:4*, 341–349.
6. K.M. Chandy and J. Misra, 1979. Distributed Simulation: A Case Study in Design and Verification of Distributed Programs, *IEEE Transactions on Software Engineering SE-5:5*, 440–452.
7. K.M. Chandy and J. Misra, 1981. Asynchronous Distributed Simulation via a Sequence of Parallel Computations, *Communications of the ACM 24*, 198–206.
8. T.C.K. Chou and J.A. Abraham, 1982. Load Balancing in Distributed Systems, *IEEE Transactions on Software Engineering SE-8:4*, 401–412.
9. Y.-C. Chow and W.H. Kohler, 1979. Models for Dynamic Load Balancing in a Heterogeneous Multiple Processor System, *IEEE Transactions on Computers C-28:5*, 354–361.
10. W.W. Chu and M.-T. Lan, 1987. Task Allocation and Precedence Relations for Distributed Real-Time Systems, *IEEE Transactions on Computers C-36:6*, 667–679.
11. D.L. Eager, E.D. Lazowska and J. Zahorjan, 1986. Adaptive Load Sharing in Homogeneous Distributed Systems, *IEEE Transactions on Software Engineering SE-12:5*, 662–675.
12. D. Ferrari and S. Zhou, 1986. A Load Index for Dynamic Load Balancing in *Proceedings of the Fall Joint Computer Conference*, pp. 684–690.
13. M.R. Garey and D.S. Johnson, 1979. *Computers and Intractability*, W.H. Freeman, San Francisco.
14. A.M. Iqbal, J.H. Saltz and S.H. Bokhari, 1986. A Comparative Analysis of Static and Dynamic Load Balancing Strategies in *Proceedings of the International Conference on Parallel Processing*, pp. 1040–1047.
15. D. Jefferson and H. Sowizral, 1985. Fast Concurrent Simulation Using the Time Warp Mechanism in *Proceedings of the Conference on Distributed Simulation*, pp. 63–69.
16. A. Kratzer and D. Hammerstrom, 1980. A Study of Load Levelling in *Proceedings Distributed Computing, COMPCON*, pp. 647–654.
17. S. Lee and J.K. Aggarwal, 1987. A Mapping Strategy for Parallel Processing, *IEEE Transactions on Computers C-36:4*, 433–442.
18. M. Livny and M. Melman, 1982. Load Balancing in Homogeneous Broadcast Distributed Systems in *Proceedings ACM Computer Network Performance Symposium*, pp. 47–55.
19. V.M. Lo, 1988. Heuristic Algorithms for Task Allocation in Distributed Systems, *IEEE Transactions on Computers 37:11*, 1384–1397.
20. H. Lu and M.J. Carey, 1986. Load-Balanced Task Allocation in Locally Distributed Computer Systems in *Proceedings of the International Conference on Parallel Processing*, pp. 1037–1039.
21. P. Markenscoff and W. Liaw, 1986. Task Allocation Problems in Distributed Computer Systems in *Proceedings of the International Conference on Parallel Processing*, pp. 953–960.
22. J. Misra, 1986. Distributed Discrete-Event Simulation, *Computing Surveys 18:1*, 39–65.
23. L.M. Ni and K. Abani, 1981. Nonpreemptive Load Balancing in a Class of Local Area Networks in *Proceedings Computer Networking Symposium*, pp. 113–118.
24. L.M. Ni and K. Hwang, 1981. Optimal Load Balancing Strategies for a Multiple Processor System in *Proceedings of the International Conference on Parallel Processing*, pp. 352–357.
25. L.M. Ni and K. Hwang, 1985. Optimal Load Balancing in a Multiple Processor System with Many Job Classes, *IEEE Transactions on Software Engineering SE-11:5*, 491–496.
26. M.S. Shanker, 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, MN.
27. M.S. Shanker, W.D. Kelton and R. Padman, 1989. Adaptive Distribution of Model Components via Congestion Measures in *Proceedings of the 1989 Winter Simulation Conference*, pp. 640–647.
28. W.H. Shaw, Jr., and T.S. Moore, 1987. A Simulation Study of a Parallel Processor with Unbalanced Loads in *Proceedings of the 1987 Winter Simulation Conference*, pp. 759–767.
29. K.G. Shin and Y.-C. Chang, 1989. Load Sharing in Distributed Real-Time Systems with State-Change Broadcasts, *IEEE Transactions on Computers 38:8*, 1124–1142.
30. H.S. Stone, 1977. Multiprocessor Scheduling with the Aid of

Network Flow Algorithms, *IEEE Transactions on Software Engineering SE-3:1*, 85–93.

31. H.S. STONE, 1978. Critical Load Factors in Two-Processor Distributed Systems, *IEEE Transactions on Software Engineering SE-4:3*, 254–258.
32. A.N. TANTAWI AND D. TOWSLEY, 1985. Optimal Static Load Balancing in Distributed Computer Systems, *Journal of the ACM 32:2*, 445–465.
33. Y.-T. WANG AND R.J.T. MORRIS, 1985. Load Sharing in Distributed Systems, *IEEE Transactions on Computers C-34:3*, 204–217.
34. W. WHITT, 1982. Approximating a Point Process by a Renewal Process, I: Two Basic Methods, *Operations Research 30:1*, 125–147.
35. W. WHITT, 1983. The Queueing Network Analyzer, *The Bell System Technical Journal 62:9*, 2779–2815.
36. W. WHITT, 1983. Performance of the Queueing Network Analyzer, *The Bell System Technical Journal 62:9*, 2817–2843.