Effects of Response and Stability on Scheduling in Distributed Computing Systems

THOMAS L. CASAVANT, MEMBER, IEEE, AND JON G. KUHL, MEMBER, IEEE

Abstract—**This paper quantitatively examines the effects of response and stability on scheduling algorithms for general-purpose distributed computing systems. Response characterizes the time required, following a perturbation in the system state, to reach a new equilibrium state. Stability is a measure of the ability of a mechanism to detect when the effects of further actions (which potentially consume the resource being scheduled) will not improve the system state as defined by a user-defined objective. These results have implications for distributed computations in general. Analysis is performed through the exercise of formal communicating finite automata models of two distinct approaches to the scheduling problem, each employing the objective of global optimal load balancing. The results indicate that absolute stability, as defined here, is not always necessary in dynamic systems for the same reasons that relatively small amounts of instability are tolerated in the design of analog control systems. It is also shown that response is a very important first-order metric of dynamic scheduling behavior, and that response and stability are related. As a paradigm for distributed computations in general, the schedulers examined reveal much about algorithm behavior.**

Index Terms—**Communicating finite automata, distributed algorithm modeling and analysis, distributed systems, load balancing, response, scheduling, stability.**

I. Introduction

SCHEDULING algorithms for computer systems are the operating system components which function continuously to manage the processing resource (among others) in the system. Proper design of such mechanisms has a great impact on overall system behavior. This design problem becomes two-dimensional in the domain of distributed computing systems since not only the question of *when* to execute, but also *where* to execute a particular task must be addressed. Towards this goal, many approaches to the problem have been tried, with variously reported results [3].

The design and analysis of distributed scheduling algorithms (DSA's) for large dynamic systems is complicated because there are a number of related factors which influence the aggregate behavior of the system and its scheduler. Two such factors are *response* and *stability*. While these are not the only factors influencing aggregate behavior, they are of particular interest since they are commonly discussed in the literature as either absolute requirements (stability) [1] or as strongly influential (response) in terms of their effect upon overall system behavior. The modeling technique given in [2] provides precise definitions of these terms as they apply to decision-making in distributed systems. Intuitively, response is a measure of how rapidly a scheduler reacts to restore the system to an equilibrium state following a perturbation from equilibrium. Stability relates to the amount of schedulable resource being consumed while the system state is changing, but not moving toward a more optimal state.

This paper addresses the relationship between the particular characteristics of response and stability and the aggregate characteristics of performance and efficiency. We define performance to be the degree to which an algorithm achieves its global objective, and efficiency to be a measure of the costs (e.g., resources consumed) associated with this level of goal achievement.

This paper describes experiments which measure the performance and efficiency of a class of distributed algorithms. Each of the chosen algorithms is in the category of global, dynamic, physically distributed, cooperative, optimal or suboptimal, approximate or heuristic techniques. Furthermore, the algorithms have the (globally defined) objective of load balancing, may be either adaptive or nonadaptive, and may involve one-time assignment or dynamic reassignment of tasks. The two strategies are based on the following philosophies.

1) Bidirectional Multilateral (BDML): This is a simple heuristic approach based on limited assumptions regarding the amount and type of information available to each node[1] of the scheduling algorithm. This information consists of a scalar value describing the desirability of the neighboring node accepting load from the node receiving the (scalar) information. Furthermore, each node may either choose to transfer a fraction of its own load to one or more of its own neighbors or request a fraction of the load from one or more of its neighbors.

2) Bidding: This is an application of the three-step contract net protocol [12]. In the first step, a request for bids is broadcast. In the second step, bids are received by the node which requested the bids, and in the third step, contracts are awarded and broadcast to the bidders.

Each algorithm variation has been completely and for-

Manuscript received June 23, 1986; revised June 6, 1988.

T. L. Casavant was with the Department of Electrical and Computer Engineering, University of Iowa, Iowa City, IA 52242. He is now with the School of Electrical Engineering, Purdue University, West Lafayette, IN 47907.

J. G. Kuhl is with the Department of Electrical and Computer Engineering, University of Iowa, Iowa City, IA 52242.

IEEE Log Number 8823660.

[1]Node refers to the computation on a local processor which executes to carry out the global scheduling policy.

mally specified using a communicating finite automata (CFA) modeling technique, as briefly described in the next section. These models were then automatically analyzed using a simulation-like method of model *exercising* to derive quantitative results.

Section II is an overview of the essential components of the model and provides definitions of the metrics of behavior used. Section III presents the analysis methods, specifies the algorithms, and discusses results of the analyses. Section IV makes some concluding remarks.

II. Quantitative Model Overview and Definitions

This section provides a brief overview of the CFA modeling technique [2] for the analysis of distributed decision-making (DDM) algorithms. This technique allows different mechanisms to be described uniformly under a set of standard terms and notation and permits characterization of the algorithm characteristics of information flow, the degree of global information sharing, the costs associated with a decision-making policy in terms of message passing events, and the performance of the policy in terms of satisfying a desired objective. Finally, these models provide a framework for quantitative evaluation and comparison among different approaches to a given problem.

Distributed algorithms have two types of characteristics: *structure* and *semantics*. Structure includes the topology of interconnection of decision-making entities, the static components of state, and some representation of static global knowledge at each node. Semantic characteristics are those which describe the actions of the algorithm at each node during execution.

The structural description of DSA's[2] employs the notation of directed graphs where the vertices represent the nodes of the scheduling algorithm and the edges represent the *neighbor relation*. Multiple message exchanges are required for communication between a pair of scheduling components not defined in the neighbor relation. The semantics of a DSA are specified by defining finite automata (FA) for each node of the graph. Communication of state information between nodes is implied by the input and output components of the FA. The portion of the model which conveys the semantics of the policy at a node is the FA transition function.

We define a component of node state (*phase*) which changes only when messages are passed. Using the value of this component to define equivalence classes of states, we greatly reduce the number of transitions which must be defined. This is done by collapsing large sets of states based only on the current information exchange phase of the node of the algorithm. The phase component also models the characterization of the degree of information sharing required by the algorithm and the passage of time in the system (i.e., we use message passing events as the basic quanta of *virtual time*).

The following definitions will be used in Section III to discuss the analysis of the two examples chosen. An important aspect affecting the performance of any dynamic scheduling function is the stability of the system resulting from a sequence of scheduling decisions. Formally, we define stability as follows.

Definitions: A *stable* scheduling algorithm, following a perturbation of the system state from equilibrium, will return the system to a state of equilibrium and *additionally* will cease continuing to take actions which cause changes in system state in finite time.

Here, *equilibrium* is defined as the condition in which the sequence of states visited by the system forms a cycle in the absence of further input changes. "Absence of further input" refers to the condition in which there are no externally-induced perturbations to system state. This is represented in a model by permitting the definition of functions which may change the internal state of a node of the algorithm without utilization of one of the algorithm's transition functions. From the algorithm's "point of view," these functions appear as stochastic processes.

Stability, as defined here, is not concerned with the rate of convergence to an equilibrium state. This rate of convergence is an efficiency attribute and is discussed next.

Definitions: Response is defined by the number of phase changes required, following a perturbation of the system state from equilibrium and in the absence of input, to return the system to an equilibrium condition.

Here, *equilibrium* is defined as the condition in which the sequence of global states visited by the system forms a cycle in the absence of further input changes. This is represented in the model by permitting the definition of functions which may change the internal state of a node of the algorithm without utilization of one of the algorithm's transition functions. From the algorithm's "point of view," these functions appear as stochastic processes.

Although optimality is not one of the metrics with which we are explicitly concerned here, it is the desired equilibrium state. Therefore, we include this definition as well.

Definition: An *optimal* DSA is one which, following a perturbation of the system state from an optimal state, will return the system to an optimal state in a finite period of time in the absence of further input.

III. Analysis Methods, Algorithms, and Experimental Results

DSA's which utilize widely varying approaches and structural frameworks present a difficulty to the task of objective quantitative comparison. Whereas the existence of static global knowledge assumptions may always produce a problem of subjectivity, the use of the model described earlier allows quantification of the cost of maintaining dynamic global knowledge in the form of measuring the amount of state information maintained internally and externally, and hence represents the density of communication load on the interconnection medium and ultimately provides the standard mechanism for objective evaluation of efficiency. In addition, fixed costs such as node complexity and cost per communication link

[2]DSA's are the subset of DDM's in which we are interested.

may be calculated using this model since the formal framework provides a way to quantify each. The neighbor relation provides a way to count numbers of communication links, and sequential algorithm analysis techniques may be used to evaluate nodal complexity.

A. Analysis Methods

The CFA modeling technique provides for *consistent* measures, useful in making comparisons between different approaches to a given problem and, ultimately, in making design choices. This is accomplished by performing two types of analysis: *static* and *dynamic*.

Static analysis is the direct measurement of the performance and efficiency attributes as defined in Section II. In our work, this was done through the application of a single perturbation to the system state, followed by a finite period of observation of the behavior of the modeled system. This is analogous to step analysis of control systems.[3] This type of measurement has two advantages: ease of measurement and consistency of measurement.

Dynamic analysis is used to measure the behavior of the system for a realistic execution environment for the scheduling algorithm. This type of analysis consists of dynamic application of multiple stochastic inputs for a (relatively) long period of time. In this research, dynamic analyses were performed to verify and correlate measurements made under static analysis to the behavior of the algorithm under simulated-dynamic operating conditions.

B. The Algorithms

Examination of the effects of response and stability will be accomplished by applying the model to two fundamentally different distributed scheduling mechanisms and presenting a number of general conclusions regarding performance and efficiency attributes. To simplify the discussion of performance, the global performance objective of load balancing will be used. This objective was chosen primarily because it is simple to describe and quantify and is also a very common factor contributing to overall system throughput in actual and proposed systems [6]–[11].

1) BDML Algorithms: The first type of algorithms to be considered are the BDML algorithms. BDML algorithms may be intuitively described as having one or more phases of exchange of scalar values describing the current load of neighbors or the current desire of neighbors to change their load. After some fixed number of iterations of this information exchange, simple "give-and-take" decisions are made with respect to neighbors. What follows is a brief description of a representative set of BDML algorithms from those analyzed.

UDI1 (Unidirectional Incremental Transfers): Each node examines the load of itself and that of its neighbors. One unit of load will be given to the neighbor with the least load if that load is less than its own load. Otherwise, no load is given. Ties are arbitrarily broken in one direction.

UDI2: This is the same as UDI1, except that the observed imbalance must be at least two units prior to making a decision to transfer any load.

UDI3: Each node examines the load of itself and that of its neighbors, beginning with the least heavily loaded neighbor and continuing through each neighbor in order of increasing load values. One unit of load will be given to each neighbor with load less than its own load, where the value of its own load used for purposes of comparison is repeatedly modified to reflect the total number of units committed to other neighbors up to this point.

UDI4: This is the same as UDI3, except that the observed imbalance must be at least two units prior to making a decision to transfer any load.

UDF1 (Unidirectional Fractional Transfers): Each node examines the load of itself and that of its neighbors. For each node which is less heavily loaded, the difference between the perceived mean value of load among all neighbors and the load of the receiving node will be given to each of these neighbors.

UDF2: This algorithm is the same as UDF1, except if after applying this heuristic no change was affected, UDI4 is applied.

BDF1 (Bidirectional Fractional Transfers): This algorithm is a true multilateral and bidirectional version of UDF1. Each node examines the load of itself and that of its neighbors. For each neighboring node, the difference between the load of the neighbor and the perceived mean value of load among all neighbors and itself will be the decision made regarding that neighbor. Note that these decisions may be negative, implying that load is intended to be taken from the neighbor, while UDI1 through UDF2 only allow decisions to take on positive values. In all further information exchanges (i.e., phases), the decision made in the previous phase is used to infer the load of neighbors at a distance 2 from the scheduling node. As more phases are added, the quality of the information upon which these inferences are made increases, but the volume of information describing the global state is not enlarged. A tunable parameter K is also added to this algorithm to control the rate at which decisions grow as a result of seeing a large imbalance towards one neighbor.

Regardless of the number of phases in this type of mechanism, the information passed is a scalar value, and the information stored by each node is a vector of scalars. The dimensionality of this vector is related only to the size of the neighbor set of a node, not the number of phases in the algorithm or the number of nodes in the system.

2) Bidding: The second mechanism evaluated is the bidding approach [12]. Here, each node has a unique identifier known to the node itself. The bidding approach collects predicted load information from only that subset of nodes which responds to bid requests. While this *may* equate to an estimated view of the entire system load dis-

[3]The mathematical methods used in quantitatively analyzing control systems, however, are not applicable to distributed computing systems in general due to inherent nonlinear behavior and semantics not easily characterized by mathematical relations such as transfer functions.

tribution, the mechanism itself does not require such an estimate in order to function. The bidding schemes examined are characterized as follows.

BID1: During the first phase, a "request-for-bids" message is broadcast to the system by the bid requester. The requester then "waits" for $N/3$ phases in order to allow all nodes within a distance $N/3$ to receive the bid request. The requester then "waits" for $N/3$ more phases in order to allow all nodes within a distance $N/3$ to respond to the bid request. Bids consist of a single positive-valued integer indicating the amount of load that the bidder is willing to receive. After $2N/3$ phases have occurred, the requester processes all bids and transmits the awards along with the load to be transferred. All decisions to transfer load are positive valued; hence, load is never requested from a bidder. A final $N/3$ phases are allowed to pass before the requester begins a new bidding cycle.

BID2: In this algorithm, the same fundamental mechanism as in BID1 is employed; however, the problem of overcommitment of resources is more intelligently dealt with. In BID1, when a bid is sent, all the load committed by the bid is assumed to be unavailable for commitment to other bid requests. BID2 incorporates a parameter which allows some load to be committed more than once. This is done by using a multiplicative factor to describe the fraction of a bid which is to be considered committed to another bid. Note that this parameter has the value 1.0 by default in BID1.

A formal specification of each algorithm was done in order to conduct extensive simulated[4] examinations of algorithm behavior. The volume of space required, however, for formal specification in this paper is prohibitive. The data supplied as illustrations in this section were obtained using the distributed scheduling simulation and analysis package) (DSSAP) [4], a tool developed to conduct simulation experiments. These experiments varied along many dimensions, including topology, algorithm type, algorithm parameter values, and system loading characteristics. The descriptions presented here are more informal and intuitive in an attempt to relate general properties of algorithms to their behavior.

C. Response

A critical aspect of analysis from control theory is that of measuring the response of a system to a reference input signal. A common reference input is the step. This is also our basis for evaluating distributed scheduling response. With respect to the objective of load balancing, we are primarily concerned with the ability of a system to respond reasonably to externally-induced changes in system load in order to arrive at a favorable load balance. This favorable response has two aspects. The first deals with the degree of optimality with which load is ultimately distributed in the system when equilibrium is reached. The second aspect is the rate of response following a perturbation from equilibrium.

Both aspects of response are measured by applying a reference signal and observing the resulting sequence of states. In our experiments, this is a step input of some number of units of load (positive, which represents arrival, or negative, which represents removal) at one processor of the system and observation of the response of the system over a period of time following the input. The observations made consist of 1) the statistical variance at equilibrium and 2) the number of phase transitions, or subcycles,[5] until equilibrium is reached.

While the actual means of detecting equilibrium varies between algorithms, common measures include the number of subcycles until the variance or range of load is within some tolerance of the equilibrium value or until incremental load movement is within some tolerance of the equilibrium value of movement per subcycle. The former mechanism is used in algorithms which are stable, and the final value of load variance is attained very slowly as equilibrium is approached. The latter mechanism is used in cases of unstable algorithms, but in which the amount of load moved per cycle is a (possibly large) constant.

As a first example, we consider two instances of a BDML load-balancing approach. The potential for instability is apparent in both UDI1 and UDI3 since a node could give load in such a way as to leave itself less heavily loaded than some of its neighbors at the end of the cycle. Then, on the following cycle, that load *could* all be returned. The element of the algorithm which makes this possible is the fact that the imbalance (bias) between nodes which may cause movement is of unit size. Therefore, load may be moved to a node when there is the possibility that the same unit of load may be returned on the next cycle. Analysis of the algorithm under a step input reveals that both UDI1 and UDI3 are unstable at equilibrium (i.e., load continues to move from node to node, even though the system is not tending towards a more optimal load distribution), as is shown in Fig. 1 and discussed in Section III-D. However, the difference between the two approaches is that UDI3 may be able to respond more quickly by being able to give load to *any* neighbor with less load.

As a specific example, we consider 20 nodes arranged under a number of topologies. The first topology is a loop (labeled l), the second and third are chordal rings of chord lengths 3 and 7 (labeled c3 and c7, respectively), the fourth is an irregular interconnection (labeled i) shown in Fig. 2, and the final topology is a full interconnection (labeled f).

The step input is an arrival of 100 units of load to node 0 at time 1 (time being measured in terms of phase transitions or subcycles). The dynamic load arrivals are characterized by two independent stochastic processes. The first describes interarrival times, and the second describes the size of arrivals in units of load. The removal of load models the completion of work and is represented in a

[4]The term simulation is used here to refer to the process of exercising an algorithm model and gathering performance and efficiency information.

[5]A cycle is the sequence of subcycles, or phases, over which information is gathered and a decision is carried out by a local scheduling entity.

cycle number	Node number 0	1	2	3	4
0	0	0	0	0	0
3	10	0	0	0	0
6	9	0	0	0	1
9	8	1	0	1	0
12	7	0	2	0	1
15	6	2	1	1	0
18	5	1	2	0	2
21	4	2	1	2	1
24	3	1	3	1	2
27	2	3	2	2	1
30	2	2	2	1	3
33	2	2	1	3	2
36	2	1	3	2	2

Fig. 1. Movement of load in a five-node loop under algorithm UDI1.

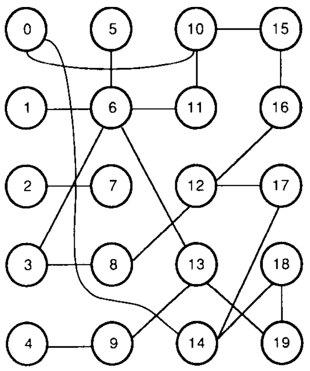


Fig. 2. Irregular topology used in simulations.

manner similar to load arrival. Three load conditions are simulated for each algorithm. Light loading (labeled dl) is characterized by geometrically distributed interarrival periods with a mean of 30 subcycles, and the size is described by a Poisson distribution with a mean of 2 units. Moderate dynamic load (labeled dm) is characterized by geometric interarrivals of mean 20 subcycles and Poisson sizes of mean 4 units. Finally, a heavy load (labeled dh) is characterized by the same distribution types, but with respective means of 15 and 8. This is summarized below.

Loading	Period Distribution	Period Mean	Size Distribution	Size Mean
dl	Geometric	30	Poisson	2
dm	Geometric	20	Poisson	4
dh	Geometric	15	Poisson	8

Table I shows the results of the step-input analysis of UDI1 and UDI3. In Table II, the results of the dynamic simulations are shown. Although UDI3 demonstrates generally poorer equilibrium states as indicated by the variance of load, the number of subcycles until equilibrium is smaller for all topologies considered. Note that in the case of a fully interconnected system, UDI3 performs better with respect to variance, but this is not a general characteristic; rather, it is a peculiar characteristic of this extreme case. A discussion of this phenomenon appears in [5] and is related to assumptions regarding static global knowledge. However, the values of range and variance averaged over time, as shown in Table II, show correlation not with improved final variance, but with increased response, as shown in Table I. Hence, the final variance in static step analyses must not be used exclusively to compare algorithms.

TABLE I
Step-Input Analysis Comparison of Two Unstable BDML Schemes

	UDI1		UDI3	
topology	subcycles to equilibrium	final variance	subcycles to equilibrium	final variance
l	320	0.1	203	0.9
c3	287	0.5	149	1.0
c7	287	0.6	107	1.1
i	332	0.8	206	1.6
f	263	22.0	17	0.0

TABLE II
Dynamic Analysis Comparison of Two Unstable BDML Schemes

		UDI1		UDI3	
topology	system load	mean range	mean variance	mean range	mean variance
l	dl	7.0	3.22	6.6	3.0
	dm	17.8	22.6	16.5	20.1
	dh	66.4	339.9	58.1	286.0
c3	dl	6.6	2.56	5.9	2.14
	dm	15.6	15.7	13.0	10.3
	dh	61.1	245.9	40.9	123.6
c7	dl	6.4	2.31	5.7	1.94
	dm	16.1	15.7	12.7	9.2
	dh	59.6	221.3	37.0	91.6
i	dl	6.7	2.58	6.3	2.27
	dm	16.9	18.02	14.0	11.79
	dh	72.9	396.87	51.0	191.0
f	dl	17.4	30.8	16.7	14.65
	dm	23.8	46.97	19.3	21.51
	dh	61.0	226.8	25.8	40.7

As a second example, we consider UDI2 and UDI4. The main difference between these and UDI1 or UDI3 is that both of these algorithms are stable. The same set of simulations was performed for these two algorithms, and the results are given in Tables III and IV. Note that even in the step analysis, a slight improvement or no change in the final variance is observed from UDI2 to UDI4. While this differs from the previous examples, it does not change the previously stated correlation because the magnitude of increase in response is easily identifiable as the change which may account for the improvement in the dynamic analysis.

We now consider the response of UDF1 and UDF2. Here, nodes may make decisions which not only include information regarding the state of their neighbors, but also the state of their neighbors' neighbors. UDF2 is designed to have the ability both to respond quickly and to achieve better equilibrium balances during times of lighter loading. The result, as justified in Tables V and VI, is that the basic mechanism of using averaging as a determining factor for the amount of load transferred works well with increasing response, and that this increase in response corresponds to improved performance over UDI2 and UDI4 in a dynamic environment. Also shown is that the

TABLE III
STEP-INPUT ANALYSIS COMPARISON OF TWO STABLE BDML SCHEMES

	UDI2		UDI4	
topology	subcycles to equilibrium	final variance	subcycles to equilibrium	final variance
l	278	8.5	161	8.5
c3	278	2.5	110	1.3
c7	284	0.7	104	0.6
i	284	0.7	110	0.7
f	287	0.0	17	0.0

TABLE IV
DYNAMIC ANALYSIS COMPARISON OF TWO STABLE BDML SCHEMES

		UDI2		UDI4	
topology	system load	mean range	mean variance	mean range	mean variance
l	dl	7.7	4.49	7.4	4.31
·	dm	18.2	24.11	17.0	21.6
	dh	66.5	343.0	58.2	287.2
c3	dl	6.6	2.64	6.0	2.3
	dm	16.0	16.1	13.3	11.1
	dh	61.1	245.7	41.4	126.2
c7	dl	6.3	2.3	5.7	1.9
	dm	16.0	15.7	13.0	10.0
	dh	59.4	220.4	37.0	92.2
i	dl	6.7	2.67	6.3	2.32
	dm	17.0	18.58	14.4	12.75
	dh	73.1	398.7	51.5	196.0
f	dl	15.5	24.1	12.3	8.3
	dm	23.6	44.0	18.0	17.3
	dh	59.5	217.8	25.3	38.2

TABLE V
RESPONSE AND OPTIMALITY OF UDF1 AND UDF2

	UDF1		UDF2	
topology	subcycles to equilibrium	final variance	subcycles to equilibrium	final variance
l	77	5.8	71	6.9
c3	41	0.8	41	0.8
c7	23	0.4	23	0.4
i	38	0.3	26	0.7
f	0	0.0	0	0.0

TABLE VI
DYNAMIC PERFORMANCE OF UDF1 AND UDF2

		UDF1		UDF2	
topology	system load	mean range	mean variance	mean range	mean variance
l	dl	4.2	1.7	4.1	1.6
	dm	10.9	11.1	10.8	10.7
	dh	20.9	41.0	20.8	41.0
c3	dl	3.9	1.4	3.6	1.2
	dm	8.9	6.2	8.9	6.1
	dh	19.5	29.5	19.9	30.7
c7	dl	4.0	1.2	3.8	1.1
	dm	8.9	5.8	8.6	5.5
	dh	19.3	29.0	18.8	27.1
i	dl	4.4	1.6	4.3	1.5
	dm	9.2	6.4	9.2	6.4
	dh	20.6	33.2	20.9	33.7
f	dl	5.2	2.0	3.8	1.2
	dm	14.9	30.2	15.6	36.2
	dh	-	-	53.4	512.9

increased algorithm complexity introduced in order to improve the steady-state optimality is capable of producing minor improvements in the lightly loaded cases, but that in a moderate to heavily loaded system there is no benefit, and in fact, it may be detrimental. This deficiency is explained by the similarity of the lightly loaded environment to the static environment under which UDI1 displayed a better equilibrium load than UDF1.

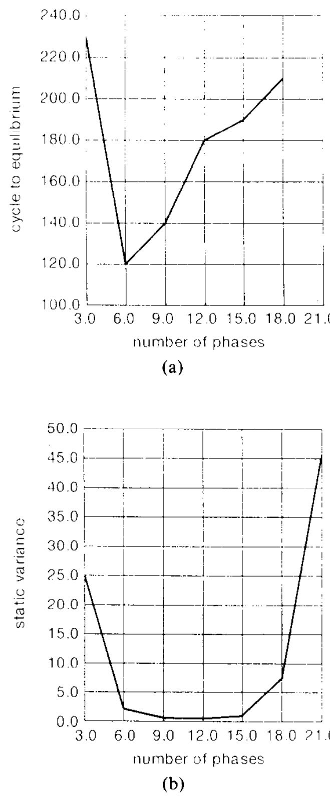


Fig. 3. Behavior of bidding. (a) Response. (b) Optimality.

We now consider BID1. Here, response is controlled through varying the number of information-exchange phases between decision-making phases. Fig. 3(a) shows a composite graph of the number of phases necessary to reach equilibrium versus the number of information exchanges per decision. Fig. 3(b) shows the relationship between the number of phases and the equilibrium values of variance—the measure of optimality. The curves are a composition of the data values sampled for each of the topologies and were constructed by averaging the values observed from each topology. Finally, Fig. 4 shows the dynamic performance of BID1 by plotting the number of phases for each topology versus the dynamic performance metric—the mean variance. Each part of Fig. 4 shows the dynamic performance under a different dynamic loading condition. A trend similar to those of the BDML algorithms is indicated. The general observation to be made again is that with respect to response in the bidding examples studied, dynamic performance correlates more strongly with response than with the ability to achieve an optimal steady state. If equilibrium optimality were used to select the number of phases to be used, a larger value would be chosen since with a higher ratio of message-passing phases to decision-making phases there is the potential for less movement of load. However, these results

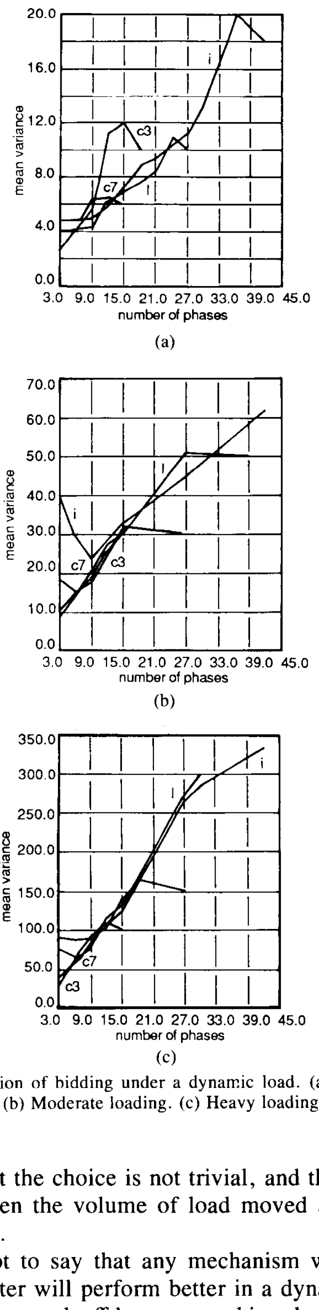


Fig. 4. Simulation of bidding under a dynamic load. (a) Light loading. (b) Moderate loading. (c) Heavy loading.

illustrate that the choice is not trivial, and that a tradeoff exists between the volume of load moved and dynamic performance.

This is not to say that any mechanism which simply responds faster will perform better in a dynamic setting. There is also a tradeoff between making decisions which make rapid changes in the gross load imbalance and making small variations in load in a region in order to find a completely optimal global load distribution. Response has been shown to have a positive effect on system behavior in this section, but it is obviously possible to create conditions of overresponse in which too high a level of response causes unacceptable levels of instability, and hence poor performance.

Another significant observation made in this portion of the study is that designing an algorithm to respond to local variations according to rules which allow the exact balancing of load under closely controlled conditions of a step workload arrival may be too simple an approach. The more relevant basis for algorithm design may be to choose algorithm parameters which allow the algorithm to adjust its rate of response to the rate of workload arrival instead of simply working from the assumption that the work which just arrived is the only event to deal with for the near future. In trying to develop distributed algorithms which satisfy an optimization criterion, while operating under a dynamic and noisy environment, it is tempting to focus on fine tuning the equilibrium decisions of the mechanism. However, the actual dynamic environment in which the algorithm is required to operate may differ from this equilibrium state to a great degree. While in some cases a mechanism may be developed which is provably optimal (in the sense of reaching an optimal state) or provably stable in equilibrium, this same mechanism may perform no better than, and possibly not as well as, an algorithm which focuses on achieving a more favorable rate of response. There are two implications of this point.

First, a common mechanism for thinking about the solution to a complex dynamic system problem is to look at localized effects and solve a local or regional subset of the original problem in a static environment. This is a logical approach since it is more difficult to try to develop mechanisms which respond intelligently to stochastic inputs than to do so for (somewhat) restricted static inputs. However, this approach may be invalid. Second, the analysis techniques used here are capable of predicting the relative merits of different factors which may influence a design. However, this is not to say that response is the most critical component in developing successful scheduling algorithms. It is generally difficult to determine which components influencing design are more critical to dynamic performance. Certainly, response is a factor which warrants examination. Finally, abstractions and simplifications made for the sake of design must be used, but the resulting mechanisms ultimately need to be evaluated in terms of dynamic performance.

D. Stability

Stability is an issue which arises in the analysis of any dynamic system, and while it is precisely defined in any particular discipline, the fundamental definition of stability is that a bounded input produces a bounded response. In the study of distributed scheduling, we consider an input to be any externally-induced change (perturbation) in the overall system state.

Without a precise or formal definition, earlier work which studied the notion of stability in distributed scheduling algorithms has implicitly assumed that unstable behavior is generally a detriment to performance [1], [13]–[16]. Using the CFA model described in Section II, we are able to examine carefully the effect of instability on performance.

1) Causes of Instability: As mentioned earlier, the BDML algorithm labeled UDI1 exhibited unstable behav-

characteristic; rather, it is a peculiar characteristic of this extreme case. A discussion of this phenomenon appears

corresponds to improved performance over UDI2 and UDI4 in a dynamic environment. Also shown is that the ior. Fig. 1 illustrates this. Here, the movement of load among the nodes of a five-node loop, initially with no load present at any node, when a step input of ten units is applied to node 0, is shown. The state of the system is shown only after it changes, and in this case, that is at the end of each cycle (every three subcycles). Since the overall state is not continuing to improve toward the ultimate goal of an optimal load distribution and the sequence of system states visited forms a cycle, this is identified as an unstable algorithm. The cause of the instability is the algorithm's rule for deciding whether to transfer load at a particular point in time. In this case, the rule is based on whether there is *any* perceived load imbalance. In a second example, UDI2, the rule is modified so that a transfer of load will not take place if the transfer itself will cause another imbalance to occur. Hence, if the bias is exactly two, then the result is to move a single unit, thus balancing the load, but if the bias is only one, then no load will be moved. Note that in order to avoid a condition of instability the definition of optimality for this scheduler is being loosened. In general, this is an example of *intolerance* instability since the inability to tolerate any condition other than complete optimality results in instability. The modified algorithm (UDI2) does not exhibit this phenomenon, as shown in Fig. 5.

cycle number	Node number 0	1	2	3	4
0	0	0	0	0	0
3	10	0	0	0	0
6	9	0	0	0	1
9	8	1	0	0	1
12	7	1	0	0	2
15	6	2	0	1	1
18	5	1	1	1	2
21	4	2	1	1	2
24	3	2	1	2	2
27	3	2	1	2	2
30	3	2	1	2	2

Fig. 5. Movement of load in a five-node loop under algorithm UDI2.

TABLE VII
OVER-RESPONSE INSTABILITY

	load moved per cycle			mean variance		
K	dl	dm	dh	dl	dm	dh
1.0	7.7	15.6	42.4	6.35	26.6	162.5
1.075	7.7	27.3	121.0	6.35	50.8	886.0
1.09	8.0	103.7	222.0	6.74	585.2	2117.0
1.1	141.6	19.0	232.0	761.4	32.0	2267.0
1.2	137.1	180.0	240.0	750.0	1242.0	2565.0
1.25	157.0	192.0	273.0	86.0	1383.0	3195.0
1.3	158.0	190.0	174.0	900.0	1373.0	3247.0
no alg.	0	0	0	95	300	1050

TABLE VIII
HIGH STATIC LOADING INSTABILITY

	load moved per cycle		
initial load	dl	dm	dh
0	2	9	20
10	24	33	40
20	50	65	75
30	75	90	100
40	110	120	135

A second cause of instability is the attempt to respond too quickly to inputs or to local conditions of suboptimality. As an illustration, consider BDF1. In this algorithm, K, a tunable parameter, is used to amplify requests for load when there is a perceived imbalance. In Table VII, observe that the effect of increasing K near and past the threshold value of approximately 1.1 causes the mean number of units moved per cycle to increase dramatically. This condition of increased scheduler activity coupled with a decreased level of dynamic performance represents severe instability. This is categorized as *overresponse* instability.

A third cause of instability is a situation in which the static level of loading in a system directly influences the amount of load moved in the system. This is termed *high static load* instability, and an illustration of this is seen in Table VIII. What is seen here is that as the initial loading of the system (data are drawn from algorithm UDF1 using the c3 topology) increases from 0 through 40 a condition of increasing instability arises. Since the arrival and departure processes are identical in the dynamic simulation of this algorithm, the mean load of the system throughout time is determined by the initial loading. This increase in scheduler activity is not simply linked to a more rapid arrival of larger amounts of load, but to the total amount of load present in the system at a given point in time. Notice that as the mean load increases, the number of units of load transferred per cycle increases roughly proportionally to the increase in mean load. The basic problem is that as the load continues to increase, the opportunity for the algorithm to overreact becomes greater. On a microscopic level, when the combination of aged information and noisy information allows the scheduler to make a first decision which causes a very large local imbalance, the algorithm tries to react quickly to rebalance the load. However, this condition is detected as an imbalance from the point of view of all those involved, and thus all parties try to respond to the imbalance in the same manner. Hence, the condition of instability propagates to a point where it is self-perpetuating. In the environment of lower static load, the opportunity to create large imbalances is not present, and therefore the situation is avoided. In the following section, the effect of this behavior pattern will be examined.

A fourth cause of instability is less easily stated in such quantifiable terms as the previous three. This cause is termed *invalid assumption* instability. It results when the operating environment of the scheduler violates certain assumptions made by the designer of the mechanism. Although this may seem quite obvious, the following example shows how the violation of an assumption which anticipates a poor operating environment in favor of a seemingly better one produces a negative impact on stability and performance.

In BDF1, a design assumption is made which presumes that the static cost of rich processor interconnection must be avoided. Hence, it is envisioned that there may be a high mean internode distance and, if a large load imbal-

TABLE IX
INVALID ASSUMPTION INSTABILITY

dynamic load	load moved per cycle			
	l	i	c3	c7
dl	7.7	111.0	142.8	142.1
dm	15.5	122.0	159.0	163.0
dh	46.2	158.0	181.0	200.0

TABLE X
A COMPARISON OF PERFORMANCE AND STABILITY SHOWING A NEGATIVE CORRELATION

topology/ algorithm	dl		dm		dh	
	mean variance	load moved	mean variance	load moved	mean variance	load moved
l/UDI1	3.22	3489	22.6	4151	340.0	4451
/UDI2	4.5	1638	24.1	3305	343.0	4219
c3/UDI1	2.56	3977	15.7	4605	245.9	4882
/UDI2	2.94	1943	16.1	3704	245.7	4633
i/UDI1	3.76	2983	28.9	3524	472.8	3899
/UDI2	4.4	1593	29.0	2995	471.0	3742
l/UDI3	3.0	3971	20.1	5170	286.0	6090
/UDI4	4.3	1739	21.6	3892	287.2	5633
c3/UDI3	2.14	5192	10.3	6646	123.6	8664
/UDI4	2.3	2123	11.1	4940	126.2	7933
i/UDI3	3.5	3680	23.1	4794	382.2	5795
/UDI4	4.1	1643	24.6	3724	388.5	5404

ance is detected locally, that this condition would be best alleviated (on a global basis) by actually predicting that the optimal value of system mean load will be less than the value computed among the scheduling node and its neighbors alone. This results in the transfer of more load than would be transferred under the control of an algorithm which made decisions attempting to create a local balance only. In other words, a new condition of local imbalance is purposely generated in the hope that this will lead to a faster convergence of overall system load balance. However, as is seen in Table IX, the poorly connected topology of the loop produces a more stable response, and as will be shown in the next section, the performance also improves as the topology becomes apparently worse.

2) The Effects of Instability on Dynamic Behavior: In this section, the effects of the various causes or forms of instability which were presented in the previous section will be discussed. We first consider intolerance instability.

This form is unique in that it is easily identifiable. It is simple to identify whether an optimality requirement is the cause of instability since this results in the unceasing movement of a small amount of load with little change in the value of the objective function during step analysis. Naturally, one might consider instability to be an undesirable attribute for a system in all cases. As described earlier, algorithms UDI1 and UDI3 both have the attribute of intolerance instability. Variants of these two algorithms called UDI2 and UDI4, respectively, were developed to be stable in this respect and to be otherwise equivalent to UDI1 and UDI3, respectively. However, interesting results, which did not support this expectation were obtained in the dynamic simulations of these algorithms. The following discussion refers to Table X. The results shown are based on a simulation length of 1000 subcycles, which was adequate for clearing initial conditions. In the presence of a light dynamic load, both UDI2 and UDI4 experience roughly half the load movement experienced by UDI1 and UDI3, respectively. This trend is continued to a somewhat lesser degree in the moderate loading environment, and in the heavily loaded simulation, the difference in load movement is barely noticeable. The explanation for this is that under light loading there are far more situations in which the load varies by a single unit than in the moderate or heavy load situations. More interesting is the fact that the unstable algorithms (UDI1, UDI3) each experienced consistently better dynamic performance than the stable algorithms. Although the reason for this lies in the fact that the instability causes a situation of more favorable response, this result demonstrates that the existence of instability may not *always* cause poorer performance. Although this example shows that instability does not always imply poor performance, the majority of the simulation results obtained did show a direct relationship between instability and poor performance.

In the example of overresponse instability in the previous section, it was observed that as the response parameter K increased, the system passed out of the domain of stable algorithms into one of extreme instability. As seen in Table VII, the number of units of load moved per cycle takes on about the same degree of increase as does the value of mean variance. In fact, the actual values of mean variance increase to levels near or above the figures for mean variance in the presence of no scheduling algorithm at all. These data show that an increase in load movement is *always* accompanied by an increase in mean variance. This illustrates the more intuitive relationship between instability and poor performance.

In the example of high static load instability presented in the previous section, it was observed that a mechanism exhibited more stable behavior for low values of system mean load than for higher values. As the amount of load increases, a corresponding decrease in system performance might be expected. However, the decrease in performance is not proportional to either the increase in static loading or the value of mean load movement. Here, as shown in Table XI, the value of mean variance under algorithm control eventually overtakes that when no algorithm is being used at all. This statistic supports the belief that under conditions of heavy system load the scheduler may do more harm than good with respect to dynamic performance. This certainly is true when considering only the cost of load movement, but here it is observed that the actual performance is degraded by the scheduler in times of heavy load independently of this cost.

Finally, in the case of invalid assumption instability, it was observed that the level of increase in load movement corresponding to an increase in static global knowledge is reflected in the degradation of performance as well. This

TABLE XI
THE EFFECT OF AN EXAMPLE OF HIGH STATIC LOAD INSTABILITY ON SYSTEM PERFORMANCE

	mean variance using BDF1/no algorithm		
initial load	dl	dm	dh
0	2/33	15/150	80/750
10	44/70	90/230	250/900
20	180/95	280/300	520/1050
30	430/100	575/380	900/1300
40	800/108	900/430	1408/1400

TABLE XII
POSITIVE RELATIONSHIP BETWEEN INVALID ASSUMPTION INSTABILITY AND DEGRADED DYNAMIC PERFORMANCE

	mean variance			
dynamic load	l	i	c3	c7
dl	6.4	568.0	877.0	767.0
dm	27.4	701.0	1045.0	991.0
dh	183.0	1103.0	1408.0	1614.0

	number moved per cycle			
dynamic load	l	i	c3	c7
dl	7.7	111.0	142.8	142.1
dm	15.5	122.0	159.0	163.0
dh	46.2	158.0	181.0	200.0

result is combined with that of Table IX in Table XII to illustrate this.

3) Summary of Stability Results: In this section, examples of both the causes and effects of instability in distributed scheduling algorithms are illustrated. An underlying goal has been to demonstrate the use of the CFA model in providing insight into the problem of locating instability and then determining whether there is a positive or negative impact on dynamic system performance. It was observed that it is possible for a limited amount of instability actually to have a positive effect on system performance. However, instability in itself is a source of lowered efficiency. This lowered level of efficiency is evident in the form of increased system overhead (message and/or load movement). As previously stated, without a precise or formal definition, earlier work which studied the notion of stability in distributed scheduling algorithms has implicitly assumed that stable behavior generally is an essential component of good performance. Utilizing the CFA model, we defined stability for the balancing of load and carefully examined the effect of instability on performance, thus answering the question of whether or not absolute stability is essential. In fact, we have shown that stability may not always be essential and, in addition, that some amount of instability may actually improve response in a way which improves overall dynamic performance. Finally, we observed that not only does instability carry a cost in terms of system overhead, but that it may also degrade the absolute performance independent of the efficiency penalty.

IV. SUMMARY AND DESIGN IMPLICATIONS

This paper addressed several issues. One of these is the use of a CFA model which allows specification of different mechanisms under the same framework. Another is the definition of metrics of quantitative behavior within the model. Finally, consistent analysis of two basic structures of scheduling algorithms for distributed systems has been reported.

The following two statements summarize the significant experimental results.

- Stability, as an absolute entity, is not essential to favorable operation of distributed scheduling algorithms. In fact, it was observed that in order to improve response in at least one class of algorithms a certain amount of instability is desirable.
- Designing algorithms for dynamic systems by assuming locally time-invariant conditions may not be a productive approach. While no other simple mechanism for design conceptualization is proposed here, it is acknowledged that the behavior of algorithms designed with static conditions in mind may operate as expected under static conditions, but may exhibit far from predictable behavior under dynamic conditions.

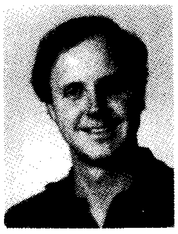
REFERENCES

[1] R. M. Bryant and R. A. Finkel, "A stable distributed scheduling algorithm," in *Proc. 2nd Int. Conf. Distrib. Comput.*, Apr. 1981, pp. 314-323.
[2] T. L. Casavant and J. G. Kuhl, "A formal model of distributed decision-making and its application to distributed load balancing," in *Proc. 6th Int. Conf. Distrib. Comput. Syst.*, May 1986, pp. 232-239.
[3] —, "A taxonomy of scheduling in general-purpose distributed computing systems," *IEEE Trans. Software Eng.*, vol. SE-14, pp. 141-154, Feb. 1988.
[4] T. L. Casavant, "DSSAP—An automated design aid for algorithms and software development in distributed computing systems," in *Proc. 2nd Int. Conf. Supercomput.*, Santa Clara, CA, May 1987, pp. 123-132.
[5] T. L. Casavant and J. G. Kuhl, "Analysis of three dynamic load-balancing strategies with varying global information requirements," in *Proc. 7th IEEE Int. Conf. Distrib. Comput. Syst.*, Sept. 1987, pp. 185-192.
[6] T. C. K. Chou and J. A. Abraham, "Load balancing in distributed systems," *IEEE Trans. Software Eng.*, vol. SE-8, pp. 401-412, July 1982.
[7] Y. C. Chow and W. H. Kohler, "Models for dynamic load balancing in a heterogeneous multiple processor system," *IEEE Trans. Comput.*, vol. C-28, pp. 354-361, May 1979.
[8] L. M. Ni and K. Hwang, "Optimal load balancing for a multiple processor system," in *Proc. Int. Conf. Parallel Processing*, 1981, pp. 352-357.
[9] L. M. Ni and K. Abani, "Nonpreemptive load balancing in a class of local area networks," in *Proc. Comput. Networking Symp.*, Dec. 1981, pp. 113-118.
[10] L. M. Ni and K. Hwang, "Optimal load balancing in a multiple processor system with many job classes," *IEEE Trans. Software Eng.*, vol. SE-11, pp. 491-496, May 1985.
[11] M. L. Powell and B. P. Miller, "Process migration in DEMOS/MP," in *Proc. 9th Symp. Oper. Syst. Principles, OS Rev.*, vol. 17, no. 5, pp. 110-119, Oct. 1983.
[12] R. G. Smith, "The contract net protocol: High-level communication and control in a distributed problem solver," *IEEE Trans. Comput.*, vol. C-29, pp. 1104-1113, Dec. 1980.
[13] J. A. Stankovic, "The analysis of a decentralized control algorithm for job scheduling utilizing Bayesian decision theory," in *Proc. Int. Conf. Parallel Processing*, 1981.
[14] —, "Simulations of three adaptive, decentralized controlled, job scheduling algorithms," *Comput. Networks*, vol. 8, no. 3, pp. 199-217, June 1984.
[15] —, "Stability and distributed scheduling algorithms," *IEEE Trans. Software Eng.*, vol. SE-11, pp. 1141-1152, Oct. 1985.
[16] G. M. Weinberg and D. Weinberg, *On the Design of Stable Systems*. New York: Wiley, 1979.

Thomas L. Casavant (S'85–M'86) received the B.S. degree in computer science and the M.S. and Ph.D. degrees in electrical and computer engineering from the University of Iowa in 1982, 1983, and 1986, respectively.

He is currently an Assistant Professor in the School of Electrical Engineering at Purdue University, West Lafayette, IN. His research interests include parallel processing, computer architecture, operating systems, distributed systems, and performance modeling and analysis.

Dr. Casavant is a member of the IEEE Computer Society and the Association for Computing Machinery.



Jon G. Kuhl (S'76–M'79) received the M.S. degree in electrical and computer engineering and the Ph.D. degree in computer science from the University of Iowa, Iowa City, in 1977 and 1980, respectively.

He is an Associate Professor in the Department of Electrical and Computer Engineering at the University of Iowa. His primary research interests are in distributed systems, parallel processing, and fault-tolerant computing. His other research interests include computer architecture, graph theory, and computer communications.