DYNAMIC TASK ALLOCATION MODELS FOR LARGE
DISTRIBUTED COMPUTING SYSTEMS

by

HUA WU DAVID CHANG, B.S., M.S.

A DISSERTATION

IN

COMPUTER SCIENCE

Submitted to the Graduate Faculty
of Texas Tech University in
Partial Fulfillment of
the Requirements for
the Degree of

DOCTOR OF PHILOSOPHY

Approved

Accepted

May, 1991

T3
1991

No. 10
Cap. 2

ACKNOWLEDGMENTS

I wish to express my appreciation to Dr. William J. B. Oldham for his guidance. His constant encouragement and valuable advice have made this dissertation possible. I would like to thank the other members of my committee, Dr. James E. Archer, Dr. Donald Gustafson, Dr. William M. Marcy, and Dr. Hua Li, for their helpful suggestions during the course of this dissertation.

I also want to thank my wife, Shumei, for her love and support which enabled me to undertake and complete my Ph.D. degree.

TABLE OF CONTENTS

ACKNOWLEDGMENTS	ii
ABSTRACT	v
LIST OF FIGURES	viii
LIST OF TABLES	x
CHAPTER	
I. INTRODUCTION	1
Background Overview	1
Taxonomy of Existing Task Allocation	
Approaches for DCS	3
Problem Definition and Proposed Approaches	5
Organization of the Dissertation	7
II. THE CLUSTERING SIMULATED ANNEALING	
MODEL (CSAM)	8
Definitions and Assumptions	8
Introduction of the CSAM	9
The Heuristic Clustering Algorithm (HCA)	10
The Annealing Process	13
III. THE MEAN FIELD ANNEALING MODEL (MFAM)	23
Theory Development of the MFAM	23
The Annealing Process	27
IV. MODEL SIMULATIONS	30
Description of the Simulation Environment	30
Simulation Results - Part I	40
Simulation Results - Part II	46
Simulation Results - Part III	53
Simulation Results - Part IV	65
V. CONCLUSIONS AND SUGGESTIONS FOR FUTURE	
WORK	74
Conclusions	74
Suggestions for Future Work	77
REFERENCES	78
APPENDICES	
A. COMPUTATION TIME OF THE HCA	83

B.	DERIVATION OF GIBB'S DISTRIBUTION	86
C.	A PARALLEL VERSION OF THE MFAM	90
D.	LIST OF INPUT DATA FOR THE SIMULATIONS	93

ABSTRACT

Dynamic task allocation for distributed computing systems (DCS) is an important goal to be achieved for engineering applications. The object is to maintain maximum utilization of computing resources while ensuring job performance at the highest productive level. The process of dynamic task allocation is to allocate or reallocate jobs (or tasks) to processors in a dynamic environment so that every processor in the system has a near equal execution load, and the total communication among processors is minimized. The dynamic environment is defined as conditions in which the execution load and communication overhead among the processors are time dependent. Yet, for a large DCS containing several hundred processors, a suitable approach that yields solutions to accommodate the dynamic environment and simultaneously preserve a near optimal solution is not available.

Most of the research relating to task allocation has produced algorithms providing sub-optimal solutions. These algorithms make task allocation decisions by utilizing fixed rules for transferring tasks from one processor to another. These approaches, referred to as the deterministic approaches, has the advantage that they perform fast task allocation decisions. In fact, the dynamics of these deterministic algorithms may be considered limiting cases of more general stochastic dynamics. Results generated by these algorithms are highly dependent on the structures of the input data and may lead to poor results that would correspond to a local minimum rather than a global minimum. An alternative approach is the powerful but computationally expensive statistical method of simulated annealing. Simulated annealing performs stochastic relaxation that decreases an energy function in searching for a global minimum to guide task allocation decisions. The result of

this approach depends on the annealing schedule instead of the structure of the input data. However, for a large DCS in a dynamic environment, the slow annealing schedule requires a long convergence time which is impractical for real-time, dynamic task allocation.

The goal of this research is to develop new task allocation methods for large DCSs in a dynamic environment. One of the basic requirements of any method is that it be capable of providing solutions faster but with quality nearly equal, for practical purposes, to that of the simulated annealing. Another requirement is that the method must yield satisfactory solutions while limited time is allowed for making the decisions. In this dissertation, we develop two dynamic task allocation models which meet these requirements. They are 1) the clustering simulated annealing model (CSAM) and 2) the mean field annealing model (MFAM). Both of these models combine characteristics of the statistical technique and differing deterministic approaches. These models are capable of performing rapid convergence of the deterministic approaches while preserving the solution quality afforded by simulated annealing. The research presented in this dissertation includes the development of theory and simulations of the CSAM and MFAM for large DCSs in a dynamic environment. Theory of the CSAM is developed by combining the simulated annealing technique with a well performed clustering algorithm. The MFAM is derived from the Gibb's distribution. Both of the CSAM and MFAM apply a new annealing schedule which provides fast convergence. In this research, the CSAM and MFAM are simulated in a large DCS with various load balance conditions. Tasks with various execution costs and inter-task communication costs are created and completed in each processor at different rates, presenting a dynamic environment. Performance of the CSAM and MFAM are evaluated and compared with both the simulated annealing method and a

deterministic method by a variety of statistics collected from the simulations. Simulation results show that the CSAM and MFAM perform much better in general than the simulated annealing method and the deterministic method in making dynamic task allocation decisions. In the two typical system conditions simulated, the CSAM and MFAM are capable of providing a stable and balanced system with $1/2$ and $1/10$ of the convergence time needed by the simulated annealing, respectively.

The results of this research is important in that it presents the feasibility of applying statistical based task allocation models on large DCSs in a dynamic environment. Solutions of these models depend on the annealing process instead of the structures of the input data, providing the possibility of obtaining better solutions by using more efficient computing hardware. These models can also be applied to other applications in which optimization is the objective.

LIST OF FIGURES

1.1	Existing task allocation approaches	3
2.1	Annealing process starts at the beginning of each time step	13
2.2	Cluster graph	15
2.3	Configuration matrix	15
2.4	Various annealing schedules	21
3.1	The change in spin as temperature decreases for a simulation with 9 tasks and 3 nodes	27
4.1	Network layout with 100 nodes contained in 8 LANs ..	32
4.2	Communication cost rate among the LANs in logical units	32
4.3	Time steps in the simulations	32
4.4	An example of task completion	34
4.5	Examples of system energy evolution	39
4.6	An example of ITC matrix which involves 10 clusters with one task overlapped	41
4.7	Results of annealing schedules (A), (B), and (C) ...	43
4.8	Results of annealing schedules (D), (E), and (F) ...	44
4.9	Energy evolution for SAA, CSAM, and MFAM	46
4.10	Task creation rate for simulation Part II	47
4.11	Layout of groups for part of the system	48
4.12	Interpretation of simulation result	49
4.13	Task allocation results of RAND at the first time step of Part II	50
4.14	Task allocation results of HCAM at the first time step of Part II	50
4.15	Task allocation results of SAA at the first time step of Part II with 200 iterations in the annealing process	51
4.16	Task allocation results of CSAM at the first time step of Part II with 200 iterations in the annealing process	51
4.17	Task allocation results of MFAM at the first time step of Part II with 40 iterations in the annealing process	52
4.18	Task allocation results of RAND at the 10th time step of Part III	54

4.19	Task allocation results of HCAM at the 10th time step of Part III	55
4.20	Task allocation results of SAA at the 10th time step of Part III with 200 iterations in the annealing process	56
4.21	Task allocation results of CSAM at the 10th time step of Part III with 200 iterations in the annealing process	57
4.22	Task allocation results of MFAM at the 10th time step of Part III with 40 iterations in the annealing process	58
4.23	Task allocation results of RAND at the 10th time step of Part III using sender initiate approach	61
4.24	Task allocation results of HCAM at the 10th time step of Part III using sender initiate approach	62
4.25	Task allocation results of SAA at the 10th time step of Part III using sender initiate approach with 200 iterations in the annealing process	63
4.26	Task allocation results of CSAM at the 10th time step of Part III using sender initiate approach with 200 iterations in the annealing process	64
4.27	Task creation rate for simulation Part IV	65
4.28	Simulation results without task allocation for Part IV	67
4.29	Task allocation results of HCAM at the 10th time step of Part IV	68
4.30	Task allocation results of CSAM at the 10th time step of Part IV with 200 iterations in the annealing process	69
4.31	Task allocation results of SAA at the 10th time step of Part IV with 200 iterations in the annealing process	70
4.32	Task allocation results of CSAM at the 10th time step of Part IV with 400 iterations in the annealing process	71
4.33	Task allocation results of MFAM at the 10th time step of Part IV with 40 iterations in the annealing process	72

LIST OF TABLES

4.1	Speedup and β for various applications	36
4.2	Input data for Part I	41

CHAPTER I

INTRODUCTION

Background Overview

Distributed computing systems (DCSs) are of current interest due to the advancement of microprocessor technology and computer networks [4, 10, 27, 32, 38, 40, 44, 45]. The DCS is customarily defined as a set of loosely coupled processors interconnected by a network. Tasks are created and completed in each processor at different rates. A task is defined as an execution entity which can be executed in any of the processors with messages that transfer back and forth to other tasks through the network. The time required to complete a task is assumed to be much longer than the communication delay for transferring the task to a remote processor. Examples of such tasks are application packages, executable programs, or procedures with a large number of iterations.

One of the major issues for DCS is task allocation [6, 12, 25, 46]. In a DCS with multiple tasks distributed among a set of processors, the problem of finding an optimal solution for task allocation to processors is found to be NP-hard [23]. The purpose of task allocation in DCSs is to increase the system throughput, which can be done by maximizing and balancing the utilization of computing resources and minimizing communication between processors. Minimizing only the inter-processor communication overhead (IPC) assigns the tasks which communicate with each other to a single processor. Load balancing alone would distribute the tasks evenly among the processors. The later process may cause a saturation effects [12, 14] due to heavy communication traffic induced by data transfers from one task to another residing in separate processors. Therefore, these two criteria are in conflict and a compromise must be made to find the optimal task assignment.

An optimal task assignment is defined as one which minimizes a cost function with consideration of load balancing and IPC costs incurred by that assignment. In a DCS with M tasks and N processors, there are N^M possible assignments of tasks to processors. A system search of solution space for the optimal solution would not be possible for even a moderately sized network. Therefore, it is necessary to apply task allocation techniques other than the method of exhaustive search to find the optimal or a good sub-optimal solution in real time.

Decisions on task allocation in DCSs are either static or dynamic. Static task allocation is the process of optimally allocating the tasks at the start, then allowing the tasks to run with no further attention. The IPC among the processors and the current state of the processors in the DCS is assumed unchanged during the time when tasks are being allocated and executed. Some applications in which this approach is adequate are geodynamics and stress analysis. In these cases, a fixed but irregular partitioning of space is made to reflect the structure under investigation [18]. However, static task allocation is not applicable to general-purpose DCSs because it does not adapt to a dynamic environment. The dynamic environment occurs because the IPC and the status of the processors are functions of time. The IPC can be changed due to the operation of the tasks involved. The status of a processor such as the execution load, free memory size, and the amount of resources available, can be changed because of task completion, task creation, task re-allocation, and processor failure. Dynamic task allocation is necessary for DCSs operating in a dynamic environment [5, 22, 36, 46, 47]. The process of dynamic task allocation is to allocate or reallocate tasks to processors in run time to meet changing conditions. In this dissertation, we address dynamic task allocation. Some applications of the dynamic task

allocation are circuit simulations, image processing, war games, and many particle dynamics [17, 18].

Taxonomy of Existing Task Allocation Approaches for DCS

Existing research on the task allocation issues can be grouped into three categories (see Fig 1.1). They are basically graph theoretical, mathematical programming, and heuristic. The graph theoretical approach [28, 42, 49] uses a graph to represent tasks, and minimizes the total IPC by performing a min-cut algorithm on the graph. The mathematical programming approach [11, 29] formulates task assignments as an optimization problem with respect to the overall system performance and solves it with mathematical programming techniques. Both of the graph theoretical and mathematical programming approaches provide optimal solutions but require large amounts of computing time and memory. These two approaches are suitable for static task allocation. The heuristic approaches provide fast but sub-optimal solutions for task allocation. Approaches in this category are clustering algorithms [7, 8, 9, 12, 14], bidding algorithm [44, 48], scatter decompositions [31, 34], orthogonal recursive bisections [20], simulated annealing [19, 26, 30], and neural networks [21, 40].

Fig. 1.1 Existing task allocation approaches

For dynamic task allocation, existing approaches are heuristic and fall into two categories -- local balancing and self-scheduling. The idea of the local balancing [22] is to apply a heuristic task allocation algorithm on a specific processor, called local balancer. The local balancer repeatedly activates the task allocation algorithm to perform dynamic task allocation among a subset of processors in the DCS. Different subsets are designed to overlap one another so that tasks can be distributed evenly over the entire system. Some mechanism must be used within the model to detect unbalanced conditions in order to invoke the local balancer. The self-scheduling approach [43, 48] allows each processor P_i to make decisions on task transfer between P_i and other processors. The task allocation model is contained in each processor in the DCS. Processors periodically exchange their load information and use the load information of the last period as a reference to make task allocation decisions for the current period.

Task allocation approaches may also be classified as deterministic or statistical (also called non-deterministic). Deterministic approaches make task allocation decisions by utilizing fixed rules for transferring tasks from one processor to another. Solutions via these approaches are highly dependent on the structures of the input data and may lead to poor results that would correspond to local minima rather than a global minimum. The advantage of the deterministic approach is that it makes task allocation decisions quickly in a dynamic environment. Approaches illustrated in Fig. 1.1 except the simulated annealing approach are all deterministic. In fact, the dynamics of these deterministic approaches may be considered limiting cases of a more general stochastic dynamics. In statistical approaches, a processor P_i will transfer some of its tasks to processor P_j with a probability P_{ij} which is dependent on the energy difference and a non-increasing

variable T corresponding to system temperature of a physical system. The statistical approaches permit stochastic relaxation that decreases an energy function in searching of a global minimum to guide the task allocation decision. The relaxation process is performed under a random basis, the effect of which is to avoid convergence to local minima. An example of the statistical approaches is simulated annealing. Task allocation under a static environment by using statistical approaches has been studied by numerous researchers [2, 3, 15, 16, 33]. These statistical approaches are not able to perform dynamic task allocation due to their requirement of long convergence time.

Problem Definition and Proposed Approaches

For large DCSs, the existing task allocation approaches are not capable of accommodating the dynamic environment and preserving optimal solutions at the same time. For the deterministic approaches which can make fast decisions under dynamic environments, the task allocation results are usually not very satisfactory for a more general system in which the execution load and the communication cost among processors are changed at random. On the other hand, for the statistical approaches which can provide better solutions, long convergence time is required. Therefore, new approaches are needed.

The goal of this research is to develop new task allocation methods for large DCSs in a dynamic environment. One of the basic requirements of any method is that it be capable of providing solutions faster but with quality equal to that of simulated annealing. Another requirement is that the method must perform satisfactory solutions while limited time is allowed for computing the task allocation decisions. To achieve the goal, we have developed two dynamic task allocation models which meet these requirements. They are (1) the clustering simulated annealing model (CSAM) and (2)

the mean field annealing model (MFAM). Both of these models are statistical models. The CSAM applies the annealing process on a set of task clusters generated by a heuristic clustering algorithm. Tasks involving high inter-task communication are formed into clusters. The annealing process searches for the global minimum by moving the clusters among the processors, reducing a large portion of the convergence time. The clusters are so constructed that high degree of relaxation is performed to avoid local minima. A new annealing schedule is developed to provide fast convergence when limited time is allowed for generating the solutions.

The MFAM applies the annealing process to update a set of spin-like variables which guides the optimal or sub-optimal solution after convergence. The approach of mean field annealing was originally developed in [2] for solving the graph partitioning problem. In the MFAM, we modify and adapt this approach to dynamic task allocation problem for large DCSs. Dynamics of the MFAM are derived from the Gibbs distribution. A new algorithm is developed to iterate the annealing process and to update the spin variables. The MFAM preserves the characteristic of simulated annealing in avoiding local minimum. Computation of the MFAM is particularly well suited for parallel and distributed implementations, largely reducing the convergence time. Detailed descriptions of the CSAM and MFAM are presented in Chapters II and III, respectively.

The research presented in this dissertation includes the development of theory and simulations of the CSAM and MFAM for large DCSs in a dynamic environment. In this research, the CSAM and MFAM are simulated in a large DCS with various loading conditions. Tasks with various execution costs and inter-task communication costs are created and completed in each processor at different rates, presenting a dynamic environment. During the simulation, we derive statistics on

system speed-up, task distribution, convergence time, and the evolution of the system energy. These statistics are required for analyzing the performance of each model and for investigating the feasibility of each model when being applied to large DCSs. Performance of the CSAM and MFAM are evaluated and compared with both the simulated annealing method and a deterministic method.

Simulation results show that the CSAM and MFAM perform much better than the simulated annealing method and the deterministic method in making dynamic task allocation decisions with limited decision time. Evolution of the system energy derived through the simulations shows that the CSAM and MFAM are capable of providing a stable and balanced system from extremely unbalanced load conditions. In more general and fluctuating load conditions, the CSAM and MFAM are capable of generating solutions near optimum with $1/2$ and $1/10$ of the convergence time needed by the simulated annealing method, respectively.

Organization of the Dissertation

The dissertation is divided into five chapters. Chapters II and III describe the complete theory development of the CSAM and MFAM, respectively. Chapter IV presents the simulation results and compares the CSAM and MFAM with the simulated annealing method and the deterministic method. Finally, conclusions and suggestions for further research are given in Chapter V.

CHAPTER II
THE CLUSTERING SIMULATED ANNEALING MODEL
(CSAM)

Before describing the model, the following definitions and assumptions are used throughout the rest of this dissertation.

Definitions and Assumptions

1. Let P denote the set of computing nodes, $P = \{P_1, P_2, \dots, P_n\}$, where n is the number of nodes in the DCS. Let T denote a set of tasks, $T = \{T_1, T_2, \dots, T_m\}$, where m is the number of tasks created by a node. Define C as a set of k task clusters, $C = \{C_1, C_2, \dots, C_k\}$, such that C is a partition of T . The task cluster is constituted by a number of tasks among which high inter-task communication is performed.

2. Define ITC (Inter-Task Communication) as an m by m matrix, where $ITC_{Tx, Ty}$ represents the average number of messages per unit of time sent from tasks T_x to T_y . It is not necessary for the $ITC_{Tx, Ty}$ to be the same as $ITC_{Ty, Tx}$. ITC is also used to represent the overhead of task synchronization. In this discussion, we assume that tasks are synchronized through message passing.

3. Define TET (Task Execution Time) as a vector of m elements, where TET_{Tx} represents the CPU time required for executing the task T_x . The cost of executing a task on any of the nodes is assumed identical. The sum of TET_{Tx} of all tasks on a node represents that processor's execution load (PEL). For maximum system throughput, it is desired to spread the tasks among the nodes as evenly as possible, i.e., the PELs are equal. We use the standard deviation of the PEL over all nodes as a measure of the load balance for different task assignments.

4. The ITC and the TET costs are assumed to be obtained from an evaluation of the characteristics of the tasks. They may be specified by the programmers, derived by the compiler, monitored by the operating system, or concluded from the previous execution results.

5. Define IPC (Inter-Processor Communication) as an n by n matrix, where IPC_{p_i, p_j} is the sum of ITC_{T_x, T_y} of all pairs of tasks (T_x, T_y) , where T_x is resided on P_i and T_y is on P_j . It is assumed that the communication cost between two tasks executed on the same node is negligible.

6. Define TL (Task Length) as a vector of m elements, where TL_{T_x} represents the code length of task T_x , i.e., the size of the code to be transferred to other processors. TL_{T_x} is independent of TET_{T_x} for any task T_x . The code of a task can be structured as a loop which requires a small code size but executes for a long time.

7. Define DIST (Distance) as an n by n matrix, where $DIST_{p_i, p_j}$ represents the communication delay between any two nodes P_i and P_j .

8. Assume that the time to transmit data between two nodes is proportional to the value given in the DIST table.

9. Define client as a node which creates tasks and server as a node which services tasks allocated from a client. Each node in the DCS can be both a client and a server.

10. Assume that tasks residing in a computing node are executed under a time-sliced, multi-tasking environment [1]. This environment allows tasks to be executed concurrently such that the ITCs among the tasks can be performed independently without concerning the execution sequence of the tasks.

Introduction of the CSAM

The CSAM developed is both suitable for the self-scheduling approach and for the local balancing approach.

In this chapter, we develop CSAM as a self-scheduling model. In Chapter IV, we will modify the CSAM as a local balancer and demonstrate the effectiveness of both approaches through simulations. The CSAM combines a heuristic clustering algorithm (HCA) and the simulated annealing technique to resolve task allocation decisions. The HCA was originally developed in our earlier research work [7] which is a deterministic type dynamic task allocation model. In [8], we extend the model and demonstrate by simulations that the HCA is capable of performing task allocation decisions more efficient and with quality better than conventional clustering algorithms [9, 12, 14]. Simulation results also show that the efficiency of the HCA is 90% of the optimal solution and 1.8 times better than solutions with tasks randomly assigned to clusters. We will illustrate the performance of the HCA in Chapter IV. Computation time of the HCA will be demonstrated to be $O(N^3)$, where N is the number of tasks. In the CSAM, we modify the HCA to form task clusters which constructs the system configuration for simulated annealing. Simulated annealing is a technique that has recently attracted significant attention as a suitable method for large scale optimization problems [39]. The CSAM combines the HCA and the simulated annealing to generate fast and global optimal solutions for dynamic task allocation problems.

The Heuristic Clustering Algorithm (HCA)

Given a set of N tasks $T = \{T_1, T_2, \dots, T_N\}$, the HCA generates $N-1$ sets of task clusters. Each cluster contains tasks which involve high ITC. Each of the cluster sets presents a different level of ITC reduction and work load distribution among the clusters. Let K be the variable indicating the number of clusters for each task assignment. Let Z_i denote the sum of TET for cluster C_i , i.e., $\sum TET_{Tx}$, where $Tx \in C_i$. Let ICC_{C_i, C_j} denote the inter-cluster

communication cost between C_i and C_j , i.e., $\sum \sum ITC_{Tx, Ty}$, where $Tx \in C_i$ and $Ty \in C_j$. Note that the ICC is different from IPC since a node may contain multiple clusters. The algorithm is as follows:

1. Initially assign each task $Tx \in T$ to a cluster C_i .
2. While ($K > 1$) do
3. {
 - Form the most independent clusters set $I \in C$ such that for each $C_i \in I$, $\sum \sum ITC_{Tx, Ty \in C_i}$ is minimum as compared to other $C_j \in C$.
4. Select the most appropriate cluster C_i from I to be split and merged with other clusters in C . The decision is made by selecting the C_i such that $\sum ICC_{C_i, C_j \in C}$ is maximum, where $i \neq j$.
5. Merge each task, Tx , in C_i into other cluster C_j (where C_i is selected from step 4 and $C_j \in C$) by following the best fit policy. The best fit policy chooses the C_j such that assigning Tx to C_j leads to a minimum value of $(Z_j - \sum ITC_{Tx, Ty})$, where $Ty \in C_j$.
- }

Step 1 of the HCA is to set the initial condition in which each cluster contains only one task. Steps 2 to 5 are iterated to establish various task assignments, from the task assignments with large number of clusters to that with small number of clusters. Step 3 is to form the most independent clusters. "Most independent" means that the sum of the ITC among the tasks involved in the cluster is minimum, i.e., the level of interaction among these tasks is the least. It is reasonable to split the most independent cluster in order to form a task assignment with smaller number of clusters. Since there can be multiple clusters contained in I , step 4 selects the most appropriate cluster to be split. The decision is made by selecting the cluster which contains the highest ICC to other clusters. Splitting such a cluster causes maximum ICC reduction as comparing to

other clusters. Finally, step 5 split the selected cluster and assigns each task in the cluster to other clusters according to a best fit policy. The best fit policy assigns the task to a cluster which yields a minimum increment on the task execution load and the ICC. The computation time of the clustering algorithm will be demonstrated as $O(N^3)$ in Appendix A, where N is the number of tasks in the assignment.

An example of the HCA is presented below.

[Example 1]

Given a task set $T = \{T1, T2, \dots, T8\}$, assume the ITC and the TET are defined as below.

ITC

	T1	T2	T3	T4	T5	T6	T7	T8
T1	0	15	0	0	0	15	10	0
T2	15	0	30	0	20	0	0	0
T3	0	30	0	20	0	0	0	0
T4	0	0	20	0	15	0	80	0
T5	0	20	0	15	0	0	0	0
T6	15	0	0	0	0	0	0	10
T7	10	0	0	80	0	0	0	20
T8	0	0	0	0	0	10	20	0

TET

T1	T2	T3	T4	T5	T6	T7	T8
10	10	10	10	10	10	10	10

The results of the HCA are presented in the following table.

# of Cluster	Tasks in Each Cluster
8	(T1) (T2) (T3) (T4) (T5) (T6) (T7) (T8)
7	(T1) (T2) (T3) (T5) (T6) (T4, T7) (T8)
6	(T1) (T2, T3) (T5) (T6) (T4, T7) (T8)
5	(T1, T6) (T2, T3) (T4, T7) (T5) (T8)
4	(T1, T6) (T2, T3) (T4, T7) (T5, T8)
3	(T1, T6) (T2, T3, T5) (T4, T7, T8)
2	(T1, T2, T3, T5) (T4, T6, T7, T8)
1	(T1, T2, T3, T4, T5, T6, T7, T8)

The Annealing Process

In the CSAM, we derive a dynamic equation to guide the process of searching for the optimal solutions. The dynamic equation gives no assumption on the order of tasks transferring between a processor and its neighboring nodes, thereby, allowing for non-deterministic dynamics in making the task allocation decisions. The non-deterministic dynamics of tasks transferring between processors correspond to movements between the states in an underlying Markov chain. A modified Metropolis procedure is developed to generate iterative improvements on the task assignment through the annealing process. The annealing process is applied repeatedly on each node at the beginning of each designated time step (see Fig. 2.1).

Figure 2.1 Annealing process starts at the beginning of each time step. Solutions must be resolved fast enough after the anneal process started.

The time step for each node can be different in length, allowing asynchronous operations on dynamic task allocation. It is required that the solution must be resolved fast enough so that tasks can be executed efficiently for the rest of the time step. Assume that new tasks are created at the beginning of each time step according to the task creation rate defined and that tasks are completed by the end of each time step according to the task completion rate. Let t denote the time step and L_t denote the number of tasks

in node i . The dynamics of the model in each processing node i is driven by the following equation.

$$L_i(t+1) = L_i(t) - \sum_{j \in A(i)} U_{ij}(t) + \sum_{j \in A(i)} U_{ji}(t) + f^+(t, \mu_i) - f^-(t, \mu_i). \quad (2.1)$$

where

$U_{ij}(t)$ = Number of tasks to be transferred from node i to node j at t .

$f^+(t, \mu_i)$ = Number of tasks created at t according to a distribution function with mean μ_i , e.g., the Poisson distribution function in the simulation cases.

$f^-(t, \mu_i)$ = Number of tasks completed at t according a distribution function with mean μ_i , e.g., the random distribution function in the simulation cases.

$A(i)$ = The set of neighboring nodes around node i .

The non-deterministic part of this model is in U_{ij} . In this model, we assume the task allocation decisions are "sender initiated", therefore, the value of U_{ji} in (2.1) is given by node j through message passing. To make use of the modified Metropolis procedure of simulated annealing for determining the U_{ij} , the following elements are developed.

Description of the System Configuration

The system configuration is constructed by the task clusters formed by the HCA. These clusters compose a cluster graph which guides the annealing process in constructing the configuration matrix. The cluster graph which corresponding to Example 1 is given in Fig. 2.2. Fig. 2.3 illustrates an example of the configuration matrix. In Fig. 4a, each row contains a set of clusters which form a partition set of T . Tasks contained in the same cluster involve high ITC. In Fig. 2.3, S is a m by n matrix, where

Figure 2.2 Cluster Graph, $T = \{T_1, T_2, \dots, T_8\}$

	P1	P2	.	.	.	Pn
T1	0	1	.	.	.	0
T2	1	0	.	.	.	0
.			.			
.			.			
.			.			
T _m	0	1	.	.	.	0

Figure 2.3 Configuration Matrix, S , $S_{ij} \in \{0,1\}$.

m is the number of tasks to be allocated from a node, say node j , and n is the number of neighboring nodes of node j . $S_{ij} = 1$ indicates that task i is to be allocated to node j . During the annealing process, configurations are updated by reassigning a cluster of tasks from one node to another.

Generation of Random Changes in the Configuration

The update of configuration matrix starts from the highest row entry in the cluster graph, e.g., row 8 in Fig. 2.2. We first select in random a cluster, C_p , in this row entry and a node, $P_j \in A(i)$, where i is the node which launches this model. The configuration matrix is then updated by reassigning the tasks contained in C_p to P_j . A decision is made to either accept or reject this new configuration. Descriptions of the decision making will be presented later. This process is repeated by selecting another cluster in the same row and another node in $A(i)$. Let T denote the temperature in the annealing process. When the energy difference becomes stabilized within some threshold, we reduce the row entry by one and decrease the temperature by $T = \alpha * T_0$, where T_0 is the starting temperature and α is some function with value less than 1. The basic idea is that tasks involving high ITC are usually allocated to the same node, therefore, updating the configuration by clusters will reduce the convergence time significantly. However, since the HCA is a deterministic type algorithm, clusters generated by the HCA may not be the best cluster set for the configuration update. In order to find better solutions, relaxation on configuration update is performed by reassigning smaller clusters among the nodes. This relaxation process allows a portion of the tasks in the clusters to be reassigned so that better solutions can be obtained. The temperature decrease in the annealing process reduces the probability of hill-climbing in searching for

the global minimum and guarantees convergence as $T \rightarrow 0$. It is desired that when the annealing process has been completed in a limited time, the task allocation decision, which is represented by the configuration matrix, demonstrate a solution with quality better than that of traditional simulated annealing method.

Ideally, updates of the configuration matrix starts from the highest row entry in the cluster graph and repeatedly reduces the row entry by one when the energy difference becomes stabilized. However, for a large configuration, e.g., 100 tasks and 10 nodes, computation time of this annealing process may not be satisfied for the problem of dynamic task allocation. Therefore, to reduce the computation time, the next row entry selected can be several rows above the current row. In this chapter, a strategy for choosing suitable row entries will be described.

When performing the random updates, the model also checks the following constraints.

- (a) $0 \leq U_{ij}(t) \leq \frac{1}{2}(L_i(t) - L_j(t)) + \text{Threshold}$,
if $L_i(t) \geq L_j(t)$.
- (b) $0 \leq U_{ij}(t) \leq L_i(t)$, if $L_i(t) < L_j(t)$.

If these constraints are satisfied, the change is accepted and the energy function computed; otherwise, the next update is started. These constraints are valid since (a) prohibits node i from transferring a very large amount of load to node j and creating a load imbalance in the opposite direction and (b) allows tasks transfer from node i to node j which may possibly decrease the energy by reducing the interprocessor communication (IPC) cost.

The Cost Function E Whose Minimization Is the Goal of the Procedure

The energy E_i to be minimized by the task allocation model in node i is constructed as follows:

$$E_i = \alpha \sum_{j \in A(i)} IPC_{ij} + \beta LB + \alpha \sum_{j \in A(i)} \sum_{T_x} TL_{Tx} DIST_{i,j} \quad (2.2)$$

$$IPC_{ij} = \sum_{T_x, T_y} ITC_{T_x, T_y} * DIST_{ij} .$$

$$LB = \frac{TET_{ave}^2}{N_{A(i)}} \sum_{j \in A(i)} L_j^2 .$$

where

$A(i)$ = A set of neighboring nodes around node i .

TET_{ave} = Average task execution time.

$N_{A(i)}$ = Number of nodes in $A(i)$.

The first term of the cost function represents the overall IPC overhead among the nodes. The second term of the cost function quantifies the load balance. The load balance term, LB , is derived from the dispersion of the execution load per node, which is

$$LB = \langle TET^2 \rangle - \langle TET \rangle^2 .$$

$$= \frac{1}{N_{A(i)}} \sum_{j \in A(i)} TET_j^2 - \left[\frac{1}{N_{A(i)}} \sum_{j \in A(i)} TET_j \right]^2 \quad (2.3)$$

Dropping the irrelevant constant term $\langle TET \rangle^2$ in Eq. (2.3), LB becomes

$$LB = \frac{1}{N_{A(i)}} \sum_{j \in A(i)} TET_j^2 .$$

$$= \frac{TET_{ave}^2}{N_{A(i)}} \sum_{j \in A(i)} L_j^2 .$$

The third term of the cost function is the total task transfer time through the network. Note that rating the cost function in term of the execution load or in term of communication cost for ITC and task transfer are not mutually exclusive. These two measures can be combined by invoking two coefficients, α and β . The value of α and β depend on the type of DCS used. In a network with large

communication delays, for example, α will be much greater than β . In a network with shared memory or high speed interconnection, this may no longer be true [35].

Let E^1 and E^2 be the energy values before and after the configuration update. If $E^1 > E^2$, this configuration update is valid, otherwise, it depends on the Boltzmann probability, $P = \exp[-(E^2 - E^1)/KT]$, where K is the Boltzmann's constant [37] which relates temperature T to energy. If P is greater than some random number between 0 and 1, the configuration update is still valid, otherwise, invalid. This allows the optimization process to take an uphill step to avoid local minima.

Annealing Schedule

The annealing schedule which controls the temperature T is critical for the final state of the system configuration to remain at the global minimum without causing fluctuations in a finite time. A large drop in temperature at the wrong time may freeze the system in a local minimum. In [30, 39, 50], several annealing schedules have been developed, each of which may be suitable for certain problems. In the CSAM, we modify the annealing schedule proposed by Szu and Hartley [50], called the fast simulated annealing. In [50], the annealing schedule is written as $T_s(t) = T_0/(1+t)$, where $T_s(t)$ is the s 'th temperature at time t and T_0 is the starting temperature. However, for the problem of dynamic task allocation, convergence time is usually limited. The approach of fast simulated annealing may not guarantee convergence within a small time period. In the CSAM, we propose an annealing schedule which is

$$T_r = \frac{T_0}{r+1} \frac{R-r-1}{R} . \quad (2.4)$$

where

$$R = \text{maximum number of steps to reduce temperature,}$$

$$r = 1, 2, \dots, R.$$

In Eq. (2.4), we use steps as units for computing the temperature. T_r represents the temperature at step r . The value of R is proportional to the maximum convergence time allowed. We will describe R later in this section. In Eq. (2.4), the value of T_0 is chosen to be larger than the largest dE normally encountered in order to have a sufficiently large probability for reaching the global minimum. T_r is reduced to zero when r is approaching R , guaranteeing convergence on a fixed point of the configuration. To simplify the problem of measuring convergence time, we use the number of iterations to represent physical time. Each iteration involves (1) updating the configuration matrix, (2) calculating the energy, and (3) deciding whether the new configuration is accepted. During the annealing, T_r is held constant for a certain number of iterations in order to allow enough configuration updates for the energy difference to become stabilized. In CSAM, the strategy of deciding when to reduce the T_r is given below.

Given a cluster graph with m rows, the annealing process of the CSAM starts from the row entry which contains $N_{A(i)}$ clusters. Clusters in the selected row are randomly distributed among the nodes. This initial procedure provides a good starting configuration in which tasks are evenly distributed and a certain amount of IPC is reduced. Then, each cluster in the row is reassigned to each node as a configuration update, performing the annealing process. After $N_{C(p)} * N_{A(i)}$ iterations, where $N_{C(p)}$ is the number of clusters in row $(m-p+1)$ (i.e., the p 'th row from the bottom of the cluster graph), T_r is reduced according to Eq. (2.4). The next row entry selected will be the $\text{MAX}(\text{current row entry} - N_{A(i)}, 1)$. However, the number of rows reduced from

the current row entry is not necessary to be $N_{A(i)}$. We choose $N_{A(i)}$ by experimenting that clusters in the selected row can be more evenly distributed among the nodes. The above procedure is repeated until the maximum number of iterations have been performed. This strategy guarantees convergence and, at the same time, performs relaxation by reassigning suitable size of smaller clusters among the nodes. Figure 2.4 illustrates the temperature path for various annealing schedules.

Figure 2.4 Various annealing schedules.

By knowing the above strategy, the number of steps to reduce the temperature (i.e., R) can be pre-determined before the annealing process starts. Let M denote the maximum number of iterations allowed for annealing. Then,

$$\begin{aligned}
 M &= N_{A(i)} \sum_{p=1}^R p N_{A(i)} . \\
 &= N_{A(i)}^2 \frac{(R+1) R}{2} .
 \end{aligned} \tag{2.5}$$

where

p $N_{A(i)}$ = number of clusters in row $(m-p+1)$ of the cluster group.

Solving for R , we get

$$R = \frac{-1 + \sqrt{1 + \frac{8M}{N_{A(i)}^2}}}{2} . \quad (2.6)$$

CHAPTER III
THE MEAN FIELD ANNEALING MODEL
(MFAM)

Theory Development of the MFAM

The MFAM is developed as a local balancer to perform dynamic task allocation among a set of computing nodes. Dynamics of the MFAM are derived from the Gibbs distribution [24] which shows that in an isolated system at equilibrium, the probability of the states $P(X,t)$ is

$$P(X, t) = \frac{e^{-\beta E(X)}}{\sum_Y e^{-\beta E(Y)}} \quad (3.1)$$

The evolution of $P(X,t)$ may be described by the dynamic equation [41] :

$$P(X, t+\Delta t) = \sum_Y w(X, \Delta t | Y, 0) P(Y, t) \quad (3.2)$$

where $w(X|Y)$ is the transition probability from state Y to state X and

$$\begin{aligned} w(X|Y) &= \frac{e^{-\beta E(X)}}{e^{-\beta E(X)} + e^{-\beta E(Y)}} \\ &= \frac{1}{1 + e^{-\beta (E(Y) - E(X))}} \end{aligned} \quad (3.3)$$

Equation (3.3) represents that when $\beta \rightarrow \infty$, which is equivalent to the zero temperature limit ($\beta = 1/T$), $w(X|Y)$ vanishes if $E(X) > E(Y)$, whereas it takes a value of 1 if $E(X) < E(Y)$. The derivation of $w(X|Y)$ is given in Appendix B.

To map this approach to the task allocation problem, we first define the environment to be optimized by the local balancer. Assume that there are m tasks and n nodes contained in the environment. Let S be an m by n matrix where $S_{ij} \in (0,1)$ is a spin variable indicating the

probability of assigning task i to node j . Let $[i]$ denote the original node location of task i and let function $\delta(i,j) = 1$ if $i = j$, otherwise $\delta(i,j) = 0$. The energy function to evaluate the states of the environment is defined by (3.4).

$$\begin{aligned}
 E(S) = & \alpha \sum_{i=1}^m \sum_{j \neq i}^m \sum_{k=1}^n ITC_{ij} S_{ik} (1 - S_{jk}) + \\
 & \beta \frac{1}{n} \sum_k^n \left(\sum_i^m (TET_i S_{ik}) \right)^2 + \\
 & \alpha \sum_i^m \sum_k^n TL_i S_{ik} (1 - \delta([i], k)) D_{[i], k}.
 \end{aligned} \tag{3.4}$$

The first term of Eq. (3.4) represents the overall IPC overhead among the nodes. The second term of the energy function quantifies the balance of load among the nodes. The third term of the energy function is the total task transfer time through the network. Finally, α and β are the same coefficients as in Eq. (2.2). The goal of the local balancer in this environment is to make optimal task allocation decisions by minimizing the energy. We assume that the local balancer monitors the current state of the nodes in the environment periodically. Now, let all the spins be at equilibrium. From (3.2), $P(S_{ij}=1)$ can be determined by the following dynamic equation.

$$\begin{aligned}
 P(S_{ij}=1, t+\Delta t) = & W(S_{ij}=1, \Delta t | S_{ij}=1, 0) P(S_{ij}=1, t) \\
 & + W(S_{ij}=1, \Delta t | S_{ij}=0, 0) P(S_{ij}=0, t).
 \end{aligned} \tag{3.5}$$

where t denotes the iteration steps of the dynamics. We assume asynchronous update for the spin variables, i.e., one spin being flipped from 0 to 1 at a time. It is desired that when the equation converges to a fixed point, S_{ij} converges to 0 or 1 for all i, j and $\sum_j S_{ij} = 1$ for all i . The matrix S then represents a task allocation decision with the global minimum. To simplify the notation, we rewrite equation (3.5) to (3.6).

$$S_{ij} = w(1|1)S_{ij} + w(1|0)(1-S_{ij}). \quad (3.6)$$

Let $E_0 = E(S)|_{s_{ij}=0}$ and $E_1 = E(S)|_{s_{ij}=1}$. Since $w(1|1)+w(0|1) = 1$, then

$$\begin{aligned} S_{ij} &= (1-w(0|1))S_{ij} + w(1|0)(1-S_{ij}) \\ &= S_{ij}(1-w(0|1)+w(1|0)) + w(1|0) \\ &= w(1|0) \\ &= \frac{1}{1 + e^{-(E_0 - E_1)/T}}. \end{aligned} \quad (3.7)$$

The result of (3.7) also matches with [2] which uses mean field annealing to solve graph partitioning problems.

Since S_{ik} 's are not independent of each other, changing a spin will affect other spins which again will affect itself. To achieve convergence, we provide the following annealing algorithm.

```

1.   for( i=1; i ≤ m; i=i+1 )
2.       for( j=1; j ≤ n; j=j+1 )
3.            $S_{ij} \leq 1/n$ 
4.   T ≤ Starting temperature
5.   while( T > Tmin )
6.       { do until(  $S_{ik}$  stabilized for all spins )
7.           { Randomly select an entry i
8.               for( j=1; j ≤ n; j=j+1 )
9.                    $S_{ij}^{old} \leq S_{ij}$ ;
10.          for( j=1; j ≤ n; j=j+1 )
11.              { for( k=1; k ≤ n, k ≠ j; k=k+1)
12.                   $S_{ik} \leq S_{ik} + S_{ij}/(n-1)$ 
13.                   $S_{ij} \leq 0.0$ 
14.                   $E_0 \leq (3.4)$ 
15.                   $S_{ij} \leq 1.0$ 
16.                  for( k=1; k ≤ n, k ≠ j; k=k+1)
17.                       $S_{ik} \leq 0.0$ 
18.                       $E_1 \leq (3.4)$ 

```

```

19.          Sij    <= (3.7)
20.          for( k=1; k ≤ n, k != j; k=k+1)
21.              Sik <= Sikold + (Sij - Sijold)/(n-1)
22.          }
23.      }
24.      T <= αT, where α is some constant, 0 < α < 1;
25.  }
```

The algorithm begins by initializing each $S_{ij} = 1/n$, for all i and j (lines 1 - 3). A high temperature is set to allow more relaxation in reaching to a global minimum (line 4). The iteration process (lines 5 - 25) then follows a sequence of operations: randomly select a spin (lines 7 - 10), update the spin (lines 11 - 18), and place the new value back (line 19), whereupon the new spin will affect other spins. During the update, the constraint, $\sum_j S_{ij} = 1$, must be maintained in order to get reasonable energy value (lines 11 - 12, 16 - 17, and 20 - 21). When all the spin variables become stable, we reduce the temperature T (line 24) and the same iteration starts again. By stable, we mean that the value of S_{ij} is within the range, $S_{ij} \pm \epsilon$, where ϵ is some threshold. At a sufficiently low value of T , all the spins will converge to 0 or 1, which indicates a local or global minimum of the energy.

An example of the effect that the decrease in temperature had upon the individual spins is illustrated in Fig. 3.1. At high temperatures, the spin variables are about $1/n$. As the temperature is lowered, a critical temperature, T_c , is reached where each spin variable begins to converge into 1 or 0. At sufficiently low temperature, the spins stabilize at 1 or 0. From this example, we can see that the critical temperature is a major factor that affects the performance of the annealing process. Starting at too high a temperature is wasteful since no progress is made toward a solution until the critical temperature is reached. Starting at too low a temperature, however, can

quench the system and quickly force it into a poor solution. In the MFAM, T_c and the annealing schedule are developed by the following strategy.

Figure 3.1 The change in spin as temperature decreases for a simulation with 9 tasks and 3 nodes.

The Anneal Process

The spin perturbations near T_c are small enough so that all the spins remain near their high temperature average of $1/n$. With this assumption, Equation (3.7) becomes

$$\begin{aligned}
 S_{ij} &= \frac{1}{1 + e^{-(E_0 - E_1)/T}} \\
 &= \frac{1}{n} \pm \epsilon.
 \end{aligned} \tag{3.8}$$

Assume $\epsilon \approx 0$, then

$$\begin{aligned}
 e^{-(E_0 - E_1)/T_c} &= n - 1, \\
 \frac{-(E_0 - E_1)}{T_c} &= \ln(n - 1).
 \end{aligned}$$

$$\begin{aligned}
T_c &= \frac{E_1 - E_0}{\ln(n-1)} . \\
&= \frac{\Delta E}{\ln(n-1)} . \tag{3.9}
\end{aligned}$$

In Equation (3.9), ΔE is estimated by randomly flipping a number of spin variables from 0 to 1 and computing the average of positive energy differences, i.e.,

$$\Delta E = \frac{\sum^x (E_1 - E_0)}{x} .$$

where

$$\begin{aligned}
x &= \text{arbitrary integer value.} \\
(E_1 - E_0) &> 0 .
\end{aligned}$$

In CSAM, the annealing schedule starts with T_c . After each of the $m * n$ spin variables being updated, the temperature is reduced according to Equation (3.11).

$$T_r = \frac{T_c}{r + 1} \frac{R - r - 1}{R} .$$

The idea of Equation (3.11) is similar to that of Equation (2.4) in CSAM, except for the value of R . Let M denote the maximum number of iterations allowed before convergence. For MFAM, R is given to be $M/(m*n)$, allowing each of the $m*n$ spin variables to be updated for each temperature. From Eq. (3.11), when $r = R-1$, T_r becomes 0, forcing all the spin variables to converge.

The major advantage of the MFAM is that computation of the annealing algorithm can easily be implemented in parallel. In Chapter IV, we simulate the model on a SIMD (Single Instruction, Multiple Data) machine by using a parallel version of the MFAM algorithm. The parallel version of the algorithm is presented in Appendix C. The parallelism can be performed by (1) using a multi-processor

computer as the local balancer and (2) allowing each node in the environment to iterate a portion of the spin variables. For either case, changes of the spin variables on one node (or processor) are broadcast to other nodes through message passing. We assume asynchronous iterations on these spin variables such that the algorithm at each node does not have to wait at predetermined points for predetermined messages to become available.

CHAPTER IV

MODEL SIMULATIONS

This chapter presents the simulation results on the proposed models as compared to other task allocation approaches for DCSs. We first describe the simulation environment which includes the hardware tools, the software structures of the simulation programs, and the experiment design. Then, we demonstrate and discuss the performance of the proposed models based on various statistics collected from the simulations.

Descriptions of the Simulation Environment

Hardware Tools and Software Structures of the Simulation Programs

In this research, the CSAM and MFAM are compared with (1) the simulated annealing approach (SAA), (2) the heuristic clustering algorithm model (HCAM), and (3) the approach of randomly assigning tasks to nodes (RAND). The SAA is a powerful but slow technique for solving combinatorial optimization problems. We use the SAA as a reference to demonstrate the solution quality of the CSAM and MFAM. The approach of HCAM is the same as CSAM without annealing. We use the HCAM as a deterministic model to show how CSAM and MFAM can improve the performance with short annealing processes. As mentioned in Chapter II, the HCAM was developed in our earlier research work and we have demonstrated in [8] that its performance is better than several existing heuristic approaches. Finally, the RAND is utilized as an indicator to present the performance of the proposed models. The MFAM was simulated on a MasPar [51] mini-supercomputer. The MasPar is an SIMD machine containing 2048 process elements (PEs) running under the UNIX operating system. The MFAM is implemented by allowing each PE to update one spin variable. Therefore, all the

spin variables can be updated in parallel, reducing the convergence time substantially. Simulation programs for other models are all executed on SUN workstations. The simulation programs are written in C and MPL (MasPar Parallel Application Language).

The simulation environment involves 100 nodes located in a network as shown in Figure 4.1. This network is constituted by 8 local area networks (LANs) with mesh connection. Communication cost among the LANs is assumed larger than that within the LAN. Figure 4.2 presents the communication cost rate among the LANs in a logical unit. The communication cost rate within a LAN is assumed identical, corresponding with standard communication mechanisms for LANs (e.g., Ethernet, token bus, and token ring). In the simulation, we assume that the system is homogeneous, i.e., the hardware characteristic of each node in the system is identical. In the future study of this research, we will extend the simulation in non-homogeneous systems.

The simulation process contains several time steps (see Fig. 4.3). Various unbalanced load conditions are generated at the beginning of each time step by allowing each node in the system to create different number of tasks. The number of tasks created in a node are derived from a Poisson distribution with specific task creation rate equal to the mean for that node. The TET and ITC among the tasks are random values with specified mean and standard deviation. The task allocation model is applied in each time step after the task creation. Tasks are completed at the end of each time step according to the task completion rate for that node. In a real system, each node may create and complete tasks at any instance of time. In this simulation experiment, task creation and task completion are performed periodically. This assumption is valid since the task allocation model should be activated periodically or when

Figure 4.1 Network layout with 100 nodes contained in 8 LANs.

	LAN0	LAN1	LAN2	LAN3	LAN4	LAN5	LAN6	LAN7
LAN0	1	2	3	2	3	4	3	4
LAN1	2	1	2	3	2	3	4	3
LAN2	3	2	1	3	2	2	4	3
LAN3	2	3	3	1	2	3	2	3
LAN4	3	2	2	2	1	2	3	2
LAN5	4	3	2	3	2	1	3	2
LAN6	3	4	4	2	2	1	3	2
LAN7	4	3	3	3	2	2	2	1

Figure 4.2 Communication cost rate among the LANs in logical units.

Figure 4.3 Time steps in the simulations.

unbalanced conditions occur. Therefore, when the delay of the time step is short enough, the assumption of creating and completing tasks periodically does not affect the result of the task allocation models. In real applications, the time delay before activating the task allocation model and the definition of unbalanced conditions are parameters to be tuned. These parameters should be set properly so that the system can constantly maintain a balanced condition without causing too much overhead in applying the task allocation model.

One distinguishing feature of our simulation is that the completion of tasks is performed by considering the multitasking operation among the tasks in a node. In real systems, the multitasking operation allows all the tasks to be executed concurrently. The task execution and the inter-task communication are also performed concurrently. The simulation programs implement the multitasking operation by considering both the TET and ITC in task completion at the same time. The amount of $ITC_{i,j}$ from task i to task j is accomplished in proportional to the completeness of TET for task i , providing consistency in task completion. Figure 4.4 illustrates an example of task completion performed in the simulation. In this example, Task1 and Task2 are partially completed while Task3 and Task4 are completed. The amount of ITC completed is proportional to the amount of TET completed. The total completion time for these tasks must be smaller or equal to the task completion rate defined.

Another distinguishing feature of our simulation is keeping track of each task in the system throughout the time steps instead of merely considering the task information statistically. Each task in the simulation program allocates an individual data structure which contains information such as the TET, TL, task source location, task current location, and the ITC to other tasks. This feature

allows us to make precise evaluation of IPC and the distribution of tasks.

Figure 4.4 An example of task completion.

Experiment Design

To build a simulation environment which closely represents a real system, we collected a number of application tasks used as references for the tasks created in the simulation. These applications are specifically designed for distributed computation. Examples of these applications include phase demodulation [52], animation systems [54], motion estimation systems [53], two dimensional macrotasked multigrid schemes [55], and three-dimensional macrotasked embedded multigrid schemes [55]. It is not our interest how these applications can solve various problems. We are interested in the characteristics, such as the level of parallel computation, the variance of task execution time, and the inter-task communication, among these applications. In order to make consistent use of these applications, we developed a scheme to evaluate these applications and map the results to our simulation environment. The scheme is developed as the following:

Define p as the parallel ratio such that

$$p = 1 - \frac{\sum_i^n \sum_j^n ITC_{i,j}}{\sum_i^n TET_i} \quad (4.1)$$

where $\sum TET$ and $\sum \sum ITC$ are the overall task execution cost and the inter-task communication cost, respectively, among the tasks to be considered. Let S_n be the speedup which is defined as the ratio of the cost when executing all the tasks in a node to the cost when executing the tasks in n nodes [13]. From [52], [53], [54], and [55], we collected the speedup for various applications in which n tasks are involved, each executing on a node. Therefore,

$$\begin{aligned} S_n &= \frac{\sum_i^n TET_i}{\text{MAX}_i (TET_i + \sum_j^n ITC_{i,j})} \\ &= \frac{\sum_i^n TET_i}{(TET_{ave} + \sum_i^n ITC_{ave}) + \Delta (TET_{ave} + \sum_i^n ITC_{ave})} \\ &= \frac{\sum_i^n TET_i}{\beta \left(\frac{\sum_i^n TET_i}{n} + \frac{\sum_i^n \sum_j^n ITC_{i,j}}{n} \right)} \end{aligned} \quad (4.2)$$

where $\beta > 1$, indicating the distribution of TET. From (4.1) and (4.2), we get

$$S_n = \frac{n}{\beta + \beta(1 - p)} \quad (4.3)$$

$$\beta = \frac{n}{(2 - p) S_n} . \quad (4.4)$$

Since we know S_n , by assuming p , we can compute β . Thus, from (4.2), we get

$$\Delta(TET_{ave}) = (\beta - 1) TET_{ave} . \quad (4.5)$$

Eq. (4.5) indicates the standard deviation of the TETs among the applications. Therefore, by assuming a logical value of TET_{ave} , the range of TETs becomes $((2-\beta)TET_{ave}, \beta TET_{ave})$. The lower bound of the range must be further defined as $\text{MAX}(0, (2-\beta)TET_{ave})$ to provide reasonable data. This range can be used to define the TET values for the tasks created in our simulation. Table 4.1 presents a list of speedup, the corresponding β 's, and TET ranges computed from the applications. We assume $p = 0.9$, indicating that the ITC is small as compared to the TET.

Table 4.1 Speedup and β for various applications.

Applica- tion No.	n	S_n	β	TET range (TET_{ave})
1	16	7.07	2.05	(0.00 - 2.05)
2	16	15.25	1.00	(1.00 - 1.00)
3	16	2.70	5.38	(0.00 - 5.38)
4	16	2.30	6.32	(0.00 - 6.32)
5	10	4.00	2.27	(0.00 - 2.27)
6	5	3.30	1.37	(0.63 - 1.37)
7	4	3.26	1.11	(0.89 - 1.11)
8	4	1.10	1.10	(0.90 - 1.10)
9	4	2.67	1.36	(0.64 - 1.36)
10	3	1.70	1.60	(0.40 - 1.60)

The ITC values can be evaluated according to p and the TETs computed. In the energy functions presented in Eq. (2.1) and Eq. (3.4), TET is quadratic, representing the load balance among the nodes. We multiply the ITC by a constant in order to scale the IPC term in the energy function so that the energy value provides a reasonable representative of both the load balance term and the IPC term. For the simulations presented in this chapter, the ITC value is selected by experimenting that the IPC term is about 1/10 of the load balance term for a balanced system.

The proposed models are simulated in a dynamic environment in which the task execution time and the number of tasks are generated dynamically. The TET and ITC given for each task is referenced from the above table. The number of tasks created in each node is derived from a Poisson distribution with the corresponding task creation rate equal to the mean. In order to demonstrate the performance of the proposed models in various system conditions, we designed two typical environments. One environment generates extremely unbalanced load conditions, not only among nodes but also among LANs. This environment allows us to see how tasks can be distributed in a LAN as well as be transferred from heavy loaded LANs to light loaded LANs. The extremely unbalanced load conditions are generated by assigning a number of nodes with high task creation rates (e.g., 30 or 40) while others with low task creation rates (e.g., 0 to 4). The other environment designed generates system conditions in a more general sense but with large fluctuations occurring in the number of tasks created. This environment is designed by varying the task creation rate for each node from 2 to 10. Since the number of tasks created follows the Poisson distribution, there may be cases that at one time step large number of tasks are created while at the next time step only few tasks are created at the same node.

Description of Statistics Collected in the Simulation

To investigate the performance of the proposed models, the following four statistics items will be generated during the simulations. Items 1 and 2 are observed from the system point of view and items 3 and 4 are observed more microscopically.

Evolution of System Energy

The evolution of the system energy generally demonstrates how a model performs in balancing the load and in minimizing the IPC through out the whole DCS for a number of time steps. The system energy E_s is computed based on

$$E_s = \frac{1}{N} (\alpha \sum_{i,j \in N} IPC_{ij} + \beta \sum_{i \in N} PEC_i^2 + \alpha \sum_{Tx} TL_{Tx} DIST_{[Tx], \langle Tx \rangle}). \quad (4.6)$$

where

$[Tx]$ = location of task Tx before task allocation.

$\langle Tx \rangle$ = location of task Tx after task allocation.

N = number of nodes in the DCS.

s = time step.

Under an unsaturated condition, $\sum_i f_i^+ \leq \sum_i f_i^-$, where f_i^+ and f_i^- are task creation rate and task completion rate in node i , respectively, we expect the evolution of system energy to be similar to Fig. 4.5(a) instead of Fig. 4.5(b).

Task Distribution

This information describes how balanced the work load is within the DCS and whether tasks are distributed locally by illustrating the PEC and IPC values for each node. We expect that the proposed models are capable of distributing tasks evenly among the DCS and that tasks created from the same node are located near to each other, minimizing the IPC.

Figure 4.5 Examples of system energy evolution. (a) Satisfactory result. (b) Unsatisfactory result.

Energy Evolution through the Annealing Process

This item presents the energy with respect to the temperature through the annealing process which demonstrates the efficiency and the effectiveness of the task allocation model. Applying different annealing schedules in the model will generate different energy results. We will evaluate several annealing schedules based on this information.

Convergence Time

In order to interpret the convergence time of the models independently from the hardware utilized, the convergence time is measured by the number of iterations. Each iteration involves (1) assigning a task (or tasks) to a node, (2) calculating the energy, and (3) deciding whether the new configuration is accepted. In the proposed models, the convergence time will be measured by the number of iterations.

Organization of the Simulations

The simulations presented in this chapter are divided into four parts. In part I, we first demonstrate the performance of several annealing schedules which were proposed in the previous chapters. The best schedule is then applied to other parts of the simulations. Then, the CSAM and the MFAM are compared to the SAA for static task allocation by evaluating the energy evolution through the annealing process. Simulations of this part presents the performance of task allocation for each model in basic level. Part II of the simulations demonstrates how CSAM and MFAM performed in balancing execution cost and in minimizing the IPC as compared to SAA, HCAM, and RAND in a system with extremely unbalanced load conditions. Part III extends the simulation from Part II by simulating the models for 10 time steps, demonstrating the performance of dynamic task allocation. A comparison of the sender initiate and the local balancing approaches is also presented. Part IV demonstrates the performance of each model in a general but with fluctuating load condition. Simulations in part IV are also performed for 10 time steps, presenting the results of dynamic task allocation.

Simulation Results - Part I

Simulations of this part are all executed in the same environment which contains 53 tasks and 9 nodes located in a LAN. Table 4.2 lists the TET ranges, ITC ranges, and the number of tasks in each node. The task cluster contained in a node indicates high ITC among the tasks in that cluster. For example, in node 4, there are 40 tasks with ITC ranged from 10 to 30. The number of task clusters is 10 with 1 task overlapped between two clusters. Then, the ITC among the tasks becomes the matrix as shown in Fig. 4.6 in which high ITC involved in every 5 tasks. The overlap of tasks in

the ITC matrix provides ambiguous input for task allocation decisions.

Table 4.2 Input data for Part I.

Node No.	# of Task	TET Range	ITC Range	# of Task Cluster	# of Task Overlapped
0	2	4 - 6	10 - 30	1	0
1	1	4 - 6	10 - 30	1	0
2	3	4 - 6	10 - 30	1	0
3	3	4 - 6	10 - 30	1	0
4	40	2 - 32	10 - 30	10	1
5	2	4 - 6	10 - 30	1	0
6	1	4 - 6	10 - 30	1	0
7	0	4 - 6	10 - 30	1	0
8	1	4 - 6	10 - 30	1	0

Figure 4.6 An example of ITC matrix which involves 10 clusters with one task overlapped.

In this simulation, we have experimented with two cooling schedules and three time schedules of changing the temperature, composing a set of six different annealing schedules. The two cooling schedules are

$$T_r = \frac{T_0}{\log(r) + 1} \frac{R - r - 1}{R}. \quad (4.8)$$

$$T_r = \frac{T_0}{r + 1} \frac{R - r - 1}{R}. \quad (4.9)$$

To avoid confusion, we use iterations, r , to represent the time in the annealing process. Let m be the number of tasks and n be the number of nodes, the three time schedules for changing the temperature T are:

1. Reduce T for each iteration r .
2. Hold T until $\Delta E < 0.5T$ or $\Delta E > 0$ for $(m*n)/10$ iterations.
3. Hold T for every 100 iterations.

Time schedule (1) is proposed suitable for short annealing process while (2) and (3) are corresponding to the time schedules of conventional simulated annealing. The six annealing schedules, (A) to (F), are composed as the following:

- (A): Eq. (4.8) with time schedule (1).
- (B): Eq. (4.8) with time schedule (2).
- (C): Eq. (4.8) with time schedule (3).
- (D): Eq. (4.9) with time schedule (1).
- (E): Eq. (4.9) with time schedule (2).
- (F): Eq. (4.9) with time schedule (3).

Figures 4.7 and 4.8 illustrates the energy evolution and the corresponding temperatures of the SAA with various annealing schedules for 1900 iterations. Initially, all the tasks are randomly distributed among the nodes. Then, each task is moved in a random order through the annealing process. The node moved to is also selected randomly. Comparing the results from (A) to (F), annealing schedule (D) provides the best performance not only on the final solution but also during the early stage of the annealing process. In the problem of dynamic task allocation, the

Figure 4.7 Results of annealing schedules (A), (B), and (C): (a) energy evolution (b) temperature evolution.

Figure 4.8 Results of annealing schedules (D), (E), and (F): (a) energy evolution (b) temperature evolution.

number of iterations allowed for annealing are limited. It is required that the annealing process must reduce large amount of energy with small number of iterations. Annealing schedule (D) is especially suitable for the CSAM because at the beginning of the annealing process, the annealing schedule allows large fluctuations so that task clusters with large size can be switched from node to node. After a few iterations, the temperature is reduced dramatically allowing only small fluctuations while small task clusters are moved at this moment. In the simulations that followed, annealing schedule (D) is used.

Figure 4.9 illustrates the energy evolution for SAA, CSAM, and MFAM. In the simulation, all the models start from the same initial conditions. The SAA and MFAM then randomly distribute the tasks while the CSAM applies the HCA at the beginning of the annealing process. The overhead of HCA is m^3 which corresponds to m/n of the time complexity of the energy function, where m and n are the number of tasks and nodes respectively. Since each iteration needs to compute the energy function twice, the overhead of HCA can be ignored in a 2000 iteration simulation. The starting energy of the CSAM represents the result of the HCA. The difference between the starting energy for SAA and that in Fig. 4.7 and Fig. 4.8 is due to the difference of random number generators on these two programs. The data presented in Table 4.2 is designed such that the optimal solution is difficult to obtain by deterministic approaches. The annealing process is then required in order to obtain better solutions. From Fig. 4.9, we can see that the CSAM performs much better than the SAA both in the final result and that in the early annealing stage. This is because that the HCA provides a good starting position for annealing and that when the number of iterations becomes large, the task cluster moved only contains small number of tasks (e.g., 1 or 2 tasks in this simulation), providing good granularity

in load balancing. In this simulation, the MFAM updates the spin variables sequentially. Therefore, it requires a larger number of iterations as compared to the CSAM in order to obtain good result. However, the MFAM was developed specially for parallel computation which can reduce large amount of iterations. In the following simulations, we will demonstrate that the MFAM, when being paralleled, is capable of performing equivalent results with 1/5 of the iterations used by the CSAM.

Figure 4.9 Energy evolution for SAA, CSAM, and MFAM.

Simulation Results -- Part II

The purpose of this section is to demonstrate the task allocation result in load balancing and in minimizing IPC by RAND, HCAM, SAA, CSAM, and MFAM in extremely unbalanced load conditions. The simulation environment involves 100 nodes with task creation rate listed in Fig. 4.10. The TET_{ave} for each node is set to be 5. The TET ranges are referenced from Table 4.2. The mean and standard deviation of ITC are

20 and 10, respectively. The maximum number of tasks which forms a task clusters in the ITC matrix is 5 with 1 task overlapped between clusters. The task length, TL, is set to be 1. A complete list of the input data is given in Appendix D. From this input, extremely unbalanced load conditions occur within LAN0, LAN2, LAN3, LAN5, and LAN7. In addition, LAN0, LAN2, and LAN3 becomes heavy-loaded LANs; LAN5 and LAN7 medium-loaded; and LAN1, LAN4, and LAN6 light-loaded. The load within the LAN generates unbalanced conditions throughout the whole DCS.

2	1	3	4	3	2	1	3	2	4
3	40	2	1	1	3	2	4	30	2
1	0	1	2	3	4	3	1	3	1
20	2	3	30	1	0	40	0	2	0
1	0	1	1	3	1	0	3	2	0
3	30	4	3	2	4	1	3	1	0
2	1	30	2	1	2	4	0	30	1
3	3	2	2	1	3	0	2	3	2
0	1	3	1	0	30	1	2	1	2
2	0	2	2	3	2	1	4	0	3

Figure 4.10 Task creation rate for simulation Part II.

In this section, all the task allocation models are performed by using the approach of local balancing. The CSAM is modified as a local balancer simply by extending the cluster graph and the configuration matrix for one node to multiple neighboring nodes. Each local balancer makes task allocation decisions for a group of 4 to 9 nodes. Groups are overlapped with each other, allowing tasks to be transferred throughout the DCS. Figure 4.11 illustrates the

layout of the groups for a part of the system. The local balancer for each group is assumed activated asynchronously for each time step. This assumption is implemented by selecting each group in a random order and by letting all the local balancers be activated within each time step. In order to assure that heavy-loaded groups have opportunities to transfer tasks to light-loaded groups, we allow each local balancer to activate twice for each time step in different random sequences. That is, after all the local balancers have been activated, each of the local balancers will be activated again in a different random sequence.

Figure 4.11 Layout of groups for part of the system. Each group contains 9 nodes with nodes overlapped.

Figure 4.12 illustrates the system condition before task allocation is applied. The PEC and IPC values are classified into seven levels as illustrated in Fig. 4.12(b) and Fig. 4.12(c), respectively. The node represented by a triangle indicates a light-loaded node, while a circle indicates a medium-loaded node and a square indicates a heavy-loaded node. This classification of PEC and IPC will be used throughout the simulation results in Part II and Part III.

Figure 4.12 Interpretation of simulation result: (a) Initial system condition before task allocation is applied in simulation Part II (b) PEC levels (c) IPC levels.

Figures 4.13 to 4.17 illustrate the results of various task allocation models in the first time step. The number of iterations applied is 40 for MFAM and 200 for SAA and CSAM. Update of the spins in the MFAM are performed in parallel for each iteration. The 40 iterations applied in the MFAM allow the model to pass through the critical temperature slowly in order to obtain satisfactory results. Since the simulation starts from a system which contains no tasks, the result of each model is capable of providing a light-loaded system at the first time step. However, each of the models performs differently to some degree. The major differences are the capability of providing less IPC and more balanced execution load among the nodes. Basically, we expect a well performed model to provide a system in which the PEC and IPC of each node are as small and as balanced as possible. From the result of RAND in Fig. 4.13, many nodes are either medium-loaded or heavy-loaded and the overall IPC is high. This result is due to the random distribution of tasks such that the IPC becomes large and tasks with high TET may be allocated to the same node. In Fig. 4.14, the HCAM does not perform

Figure 4.13 Task allocation results of RAND at the first time step of Part II: (a) distribution of PEC (b) distribution of IPC.

Figure 4.14 Task allocation results of HCAM at the first time step of Part II: (a) distribution of PEC (b) distribution of IPC.

Figure 4.15 Task allocation results of SAA at the first time step of Part II with 200 iterations in the annealing process: (a) distribution of PEC (b) distribution of IPC.

Figure 4.16 Task allocation results of CSAM at the first time step of Part II with 200 iterations in the annealing process: (a) distribution of PEC (b) distribution of IPC.

satisfactorily in minimizing the IPC. In Fig. 4.15, the SAA improves the result of the RAND to some degree. In Fig. 4.16, the CSAM performs better than SAA in both load balancing and in IPC reduction. For 200 iterations, which is a very short annealing process for the problem in this simulation, the CSAM reduces 23% of the system energy from HCAM and 34% from RAND. Finally, in Fig. 4.17, the MFAM performs extremely well in IPC reduction and provides a system with balanced load. The MFAM reduces 31% of the system energy from HCAM and 41% from RAND. The performance difference among the above models may not seem apparent in the first time step. However, this difference will largely effect the task allocation results if the simulation lasts for a number of time steps. In the next section, we will demonstrate this effect.

Figure 4.17 Task allocation results of MFAM at the first time step of Part II with 40 iterations in the annealing process: (a) distribution of PEC (b) distribution of IPC.

Simulation Results -- Part III

In real systems, tasks are usually created by batch for a period of time. We simulate such conditions by running the simulation for a number of time steps. This section examines the simulation results that given extremely unbalanced load conditions for a number of time steps, how the proposed models accommodate such conditions dynamically. Input data of this simulation is the same as that in Part II. The simulation starts from a clean system in which no task exists. Tasks are then created and completed for each time step according to the task creation rate and task completion rate defined. Since we assume that tasks are executed in multitasking, a task may last for a number of time steps before completion. To get the best efficiency in task completion, tasks must be evenly distributed with the minimum IPC among the nodes. Since the system is simulated in an unsaturated environment, most of the tasks should be completed before new tasks are created by applying the optimum task allocation. Ideally, the result of the final task distribution after a number of time steps should be close to that in Part II. Therefore, for a well performed model, we expect that the value of PEC and IPC are both small and evenly distributed among the DCS.

Figures 4.18 to 4.22 illustrate the task distribution result and the system energy evolution for each model at the end of the 10th time steps. The system energy evolution diagram illustrates the system energy after task allocation has been completed at each time step. The system energy at time step 0 is the energy of the initial system condition. In Fig. 4.18, the RAND randomly distributes tasks among the network which results large IPC. The overhead of IPC degrades the efficiency of task completion. Therefore, extremely unbalanced conditions occur both on the distribution of PEC and on the IPC as well. The system energy of RAND increases dramatically, indicating that the

(a)

(b)

(c)

Figure 4.18 Task allocation results of RAND at the 10th time step of Part III: (a) distribution of PEC (b) distribution of IPC (c) system energy evolution.

Figure 4.19 Task allocation results of HCAM at the 10th time step of Part III: (a) distribution of PEC (b) distribution of IPC (c) system energy evolution.

Figure 4.20 Task allocation results of SAA at the 10th time step of Part III with 200 iterations in the annealing process: (a) distribution of PEC (b) distribution of IPC (c) system energy evolution.

(c)

Figure 4.21 Task allocation results of CSAM at the 10th time step of Part III with 200 iterations in the annealing process: (a) distribution of PEC (b) distribution of IPC (c) system energy evolution.

Figure 4.22 Task allocation results of MFAM at the 10th time step of Part III with 40 iterations in the annealing process: (a) distribution of PEC (b) distribution of IPC (c) system energy evolution.

system becomes more and more loaded as the time step increases. The results of HCAM in Fig. 4.19 performs better than that of RAND in PEC distribution and in IPC reduction, but still, presenting unbalanced load conditions in some parts of the system. From Fig. 4.19(c), the system energy increases slightly as the time step increases, meaning that the performance of the HCAM will be degraded if more time steps are executed. By applying the annealing process with 200 iterations in each time step, the SAA performs much better than the RAND. However, in Fig. 4.20, several nodes in LAN0 are heavy-loaded. This is because that SAA reduces IPC by clustering tasks which involve high ITC among each other and that tasks does not re-allocated fast enough after being clustered in order to maintain load balancing in the annealing process. The system energy evolution diagram of SAA also shows a slight increase in the system load. In Fig. 4.21, the CSAM performs much better than any of the above models. The PEC and IPC illustrated in Fig. 4.21 are both low and balanced throughout the system. From the system energy evolution diagram, the CSAM is stable, meaning that the performance of the CSAM can be maintained for more time steps. For the MFAM, the performance of task distribution is as good as the CSAM with only 40 iterations. The system energy of MFAM is also stable during the simulation.

The above models except MFAM were also simulated by using the sender initiate approach. In this approach, the task allocation model is activated for each node of the system in random orders. Since tasks can only be transferred to the neighboring nodes, the task transfer rate is slower than that by using the local balancing approach. However, only the tasks resided at the activated node will be transferred. The number of iterations in the annealing process can be largely reduced. Therefore, in order to compensate the slow task transfer rate, we allow each model

to activate three times, each for 50 iterations, in a time step. The total number of iterations for all the nodes in one time step is then $150 * 100$ which is larger than the $400 * 25$ iterations performed in the local balancing approach with 25 groups. However, in the sender initiate approach, all the nodes can be activated asynchronously while in the local balancing approach, only the local balancers are activated. Therefore, the question of which of these two approaches is better is actually a design issue that depends on the hardware used on the nodes and the local balancers. Figures 4.23 to 4.26 illustrate the final task distribution and the system energy evolution of all the models with the approach of sender initiate. In Fig. 4.23, the system provided by RAND is unbalanced on both PEC and IPC. The unbalanced conditions occur because the activated node randomly sends tasks to the neighboring nodes without considering the TET and ITC among the tasks. Tasks with large TET may be assigned to the same node. In addition, tasks sent from a heavy-loaded node may be sent back to that node when its neighboring nodes are activated, reducing the speed of distributing tasks from heavy-loaded nodes to light-loaded nodes. The system energy of RAND increases through the simulation, providing a heavy-loaded systems. In Fig. 4.24, the HCAM reduces the IPC to some degree by sending task clusters to neighboring nodes. Since in HCAM, tasks have been clustered before being sent according to the TET and ITC values of the tasks. Tasks with high TET are not likely to be allocated to the same node unless high ITC is involved among these tasks. Therefore, the HCAM performs better than RAND. However, the result of HCAM still needs improvement because the system energy increases in the simulation. In Fig. 4.25, SAA does not perform well in load balancing in some part of the system. For the sender initiate approach, a heavy-loaded node with its neighboring nodes heavy-loaded becomes difficult to send its tasks out

Figure 4.23 Task allocation results of RAND at the 10th time step of Part III using sender initiate approach: (a) distribution of PEC (b) distribution of IPC (c) system energy evolution.

Figure 4.24 Task allocation results of HCAM at the 10th time step of Part III using sender initiate approach: (a) distribution of PEC (b) distribution of IPC (c) system energy evolution.

Figure 4.25 Task allocation results of SAA at the 10th time step of Part III using sender initiate approach with 50 iterations in the annealing process: (a) distribution of PEC (b) distribution of IPC (c) system energy evolution.

(c)

Figure 4.26 Task allocation results of CSAM at the 10th time step of Part III using sender initiate approach with 50 iterations in the annealing process: (a) distribution of PEC (b) distribution of IPC (c) system energy evolution.

until its neighbors become light-loaded. The system energy of SAA also increases in the simulation. Finally, in Fig. 4.26, the CSAM performs much better than all the other models. The PEC and IPC are evenly distributed throughout the whole system. The system energy is stable with the energy value close to that of the CSAM with local balancing approach.

From the simulation results shown above, we can see that the CSAM and MFAM performs well in a dynamic system with extremely unbalanced load conditions in which deterministic models and conventional simulated annealing approaches have difficult providing satisfactory solutions. The approaches of sender initiate and local balancing provides equivalent results for the CSAM.

Simulation Results -- Part IV

The simulation environment for this part provides more balanced but fluctuating load conditions than that in Part II and Part III. The task creation rate for each node is ranged from 2 to 10 (see Figure 4.27). Since the number of

Figure 4.27 (a) Task creation rate for simulation Part IV (b) PEC levels (c) IPC levels.

tasks created in each node follows the Poisson distribution, the environment fluctuates on the number of tasks created in each node throughout the time space. In this part of the simulation, all the task allocation models are simulated for 10 time steps. Figures 4.28 to 4.33 present the final task distribution results and the system energy evolution for each of the models. Since the variance of task creation rate among the nodes is small, the approach of RAND actually performs worse than that without task allocation. This is because that the RAND cannot provide enough energy reduction by distributing the tasks but instead, generates heavy IPC among the nodes. In Fig. 4.28, we illustrate the simulation results without applying task allocation. Obviously, the system becomes unbalanced and the system energy increases dramatically after a few time steps. Since tasks are not distributed, IPC is zero in the system. Figure 4.29 presents the simulation results with the HCAM approach. For this environment, the HCAM performs well both on the PEC distribution and on the IPC reduction, demonstrating that the HCA is capable of generating satisfactory task allocation decisions in general conditions. The system is also stable by viewing the system energy evolution diagram. Figure 4.30 presents the simulation results for CSAM with 200 iterations in the annealing process. Since the HCAM has provided satisfactory solutions already, the CSAM does not improve the performance much by applying the annealing process, meaning that the system has been close to an optimum condition. Figure 4.31 presents the simulation result for SAA. The SAA does not perform satisfactorily with an annealing process of 200 iterations. A few nodes are still heavy-loaded. From the system energy diagram of the SAA, the system energy increases throughout the simulation. In Fig. 4.32, we simulate the SAA by increasing the number of iterations to 400. The performance of the SAA is then approximately

(a)

(b)

Figure 4.28 Simulation results without task allocation for Part IV: (a) distribution of PEC (b) system energy evolution.

Figure 4.29 Task allocation results of HCAM at the 10th time step of Part IV: (a) distribution of PEC (b) distribution of IPC (c) system energy evolution.

Figure 4.30 Task allocation results of CSAM at the 10th time step of Part IV with 200 iterations in the annealing process: (a) distribution of PEC (b) distribution of IPC (c) system energy evolution.

Figure 4.31 Task allocation results of SAA at the 10th time step of Part IV with 200 iterations in the annealing process: (a) distribution of PEC (b) distribution of IPC (c) system energy evolution.

Figure 4.32 Task allocation results of SAA at the 10th time step of Part IV with 400 iterations in the annealing process: (a) distribution of PEC (b) distribution of IPC (c) system energy evolution.

Figure 4.33 Task allocation results of MFAM at the 10th time step of Part IV with 40 iterations in the annealing process: (a) distribution of PEC (b) distribution of IPC (c) system energy evolution.

equivalent to the CSAM in Fig. 4.30. Finally, Fig. 4.33 presents the simulation results of MFAM. The MFAM, in this case, allows each local balancer to activate once per time step with 80 iterations for the annealing process. The MFAM requires 80 iterations in order to converge to an equilibrium state. The MFAM performs extremely well in load balancing and in minimizing IPC. Comparing Figures 4.32 and 4.33, the MFAM provides equivalent performance as the SAA with only 1/10 of the iterations. From the experiments above, we conclude that in a general environment, the CSAM and MFAM are capable of providing solutions close to optimum with 1/2 and 1/10 of the iterations, respectively, needed in SAA.

CHAPTER V

CONCLUSIONS AND SUGGESTIONS FOR FUTURE WORK

Conclusions

We have introduced the problem of dynamic task allocation and the existing approaches to solve this problem. The goal of this research is to develop new approaches which provide satisfactory solutions in real-time and which provide solutions faster and with quality nearly equal to that of the simulated annealing method. In this dissertation, we have achieved the goal by developing two dynamic task allocation models, CSAM and MFAM, for large DCSs. Theory of these models has been described in detailed. The CSAM and MFAM together with other approaches have been simulated for a large DCS in a dynamic environment. The MFAM was simulated in an SIMD typed mini-supercomputer with 2048 process elements, while other models were simulated on Sun workstations. The simulation environment was designed to represent many features of real systems. The experiments simulated in this research includes two typical system conditions, one with extreme unbalanced load conditions and the other with more general but fluctuating load conditions. Input data for these experiments are derived from real applications. The criteria used for performance evaluation include evolution of system energy, final task distribution, model convergence time, and energy evolution through the annealing process.

From the simulation results, we conclude the following:

1. The CSAM and MFAM are capable of providing satisfactory solutions in real-time. The CSAM and MFAM improve the performance of the deterministic approaches substantially by applying annealing processes within limited period of time. This improvement becomes substantial especially when the problems are difficult for deterministic

approaches to obtain satisfactory results. For a typical system condition with extreme unbalanced load as given in the simulation Part II, the CSAM reduces 23% of the system energy from HCAM and 34% from RAND. The MFAM, with the parallel computation capability, reduces 31% of the system energy from HCAM and 41% from RAND. In a dynamic environment with the same system conditions lasted for 10 time steps, the CSAM and MFAM both reduce 50% of the final system energy from HCAM and 80% from RAND. For all the cases in the simulations, the CSAM and MFAM provide systems with balanced load and small IPC. To obtain the above improvement, the CSAM and MFAM applied an annealing schedule which is especially suitable for short annealing process, satisfying the requirement of making task allocation decisions in real-time. In the simulation, the annealing schedules are given as 200 iterations and 40 iterations for CSAM and MFAM, respectively. The time complexity of each iteration is $c \cdot m^2 \cdot n$ where c is a constant indicating the number of instructions in the loop, m is the number of tasks, and n is the number of nodes. The overhead of the annealing process is considerably small. For example, given 50 tasks to be assigned to 9 nodes, a local balancer with computation speed as 16 MIPS requires $c \cdot 1.4$ milliseconds per iteration, where c is 1 in our simulation programs. Therefore, for a 200 iteration annealing process, it takes only 0.28 seconds which is considerably smaller as compared to time steps with tens of seconds to minutes.

2. The CSAM and MFAM are capable of performing better and faster convergence than conventional SAA. From the energy evolution results presented in simulation Part I, the CSAM performs much better than the SAA both in the final energy and in the early annealing stage. The advantage of CSAM is that the HCA provides a good starting position for annealing. The approach of switching task clusters among the nodes favors load balancing with minimum IPC, reducing

the number of iterations in the annealing process. The granularity of load balancing is preserved by switching small task clusters among the nodes. The advantage of MFAM is that the model can be easily implemented in parallel computation. The parallelism feature of the MFAM largely reduces the model convergence time. The solution quality of the MFAM is preserved by the approach of mean field approximation. From the simulation results in Part III, the CSAM and MFAM reduce 46% and 44% of the final system energy from SAA, respectively. From the simulation results in Part IV, the HCA performs extremely well. The result of HCA leads the CSAM to provide a system with optimal task allocation solutions within limited number of iterations. The number of iterations for CSAM and MFAM is $1/2$ and $1/10$, respectively, of that needed for the simulated annealing approach to get equivalent performance.

3. The CSAM and MFAM are capable of accommodating dynamic environment and maintaining stable system condition throughout the time space. In the system with extreme unbalanced load conditions, both the CSAM and MFAM are capable of providing results with load more balanced and with IPC lower than other methods. The result of load balance and low IPC from the proposed models allows the system to remain stable throughout the time space, while in other methods, the system becomes heavy-loaded after a few time steps due to the unbalanced load conditions and the large overhead of the IPC.

4. The approach of sender initiate against the local balancing approach has been simulated in the same environment. From the simulation results, there is no apparent evidence to show which of these two approaches is better. In the simulation of CSAM, the final result of the local balancing approach is better than the sender initiate approach, both in the system energy and in the number of iterations. However, the sender initiate approach allows

each of the node in the system to activate the task allocation model asynchronously, providing more relaxation and parallelism in task distribution. In the local balancing approach, all the nodes controlled by the local balancer must wait for the task allocation decisions generated by the local balancer. Therefore, the decision of selecting which of the above two approaches is actually a design issue that depends on the hardware used on the nodes and on the local balancer.

From the above conclusions, we can assure that the CSAM and MFAM perform well in a dynamic system with various load conditions in which deterministic methods and conventional simulated annealing approach are difficult to provide satisfactory solutions. The convergence time required for the annealing process in the CSAM and MFAM is limited, providing the possibility of obtaining better solutions by using efficient computing hardware.

Suggestions for Future Work

Several issues in the proposed models still deserve further experimentation. These issues are: (1) methods of limiting the communication overhead for message passing among the PEs in the MFAM, (2) decisions of defining the size of an environment controlled by a local balancer and the size of the overlap among adjacent environments, (3) methods of solving the contention problem that arises when more than one local balancer is trying to control the same node, and (4) simulations of the models in non-homogeneous systems. In the future work of this research, we will extend the models with the above issues considered.

REFERENCES

1. D. V. Anderson, "Parallel Computing and Multitasking," Computer Physics Communications 43, pp. 69-87, 1986.
2. David E. Van Den Bout and Thomas K. Miller, III, "Graph Partitioning Using Annealed Neural Network," IEEE Transactions on Neural Networks, Vol. 1, No. 2, June 1990.
3. David E. Van Den Bout and Thomas K. Miller, III, "Graph Partitioning Using Annealed Neural Network," Proc. IEEE Int. Conf. Neural Networks, 1989, pp. I:521-I:528.
4. Jonathan P. Bowen and Tim J. Gleeson, "Distributed Operating System," Distributed Computer Systems, Theory and Practice, Butterworths, 1990.
5. Thomas L. Casavant and Jon G. Kuhl, "Effects of Response and Stability on Scheduling in Distributed Computing Systems," IEEE Transactions on Software Engineering, Vol. 14, No. 11, November 1988.
6. Thomas L. Casavant and Jon G. Kuhl, "A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems," IEEE Transactions on Software Engineering, Vol. 14, No. 2, February 1988.
7. H. W. David Chang and William J.B. Oldham, "A Task Allocation Model for Distributed Computing Systems in a Catenet," ISMM, Computer Applications in Design, Simulation and Analysis, March 1990.
8. H. W. David Chang and William J.B. Oldham, "A Task Allocation Model for Large Distributed Computing Systems," International Journal of Mini & Microcomputers, (to be published).
9. Wesley W. Chu and Lance M-T. Lan, "Task Allocation and Precedence Relations for Distributed Real-Time Systems," IEEE Transactions on Computers, Vol. C-36, No.6, June 1982.
10. David R. Cheriton, "The V Distributed System," Communication of the ACM, Vol. 31, No. 3, March 1988.
11. Timothy C. K. Chou and Jacob A. Abraham, "Load Balancing in Distributed Systems," IEEE Transactions on Software Engineering, Vol. SE-8, No. 4, July 1982.

12. Wesley W. Chu and etc, "Task Allocation in Distributed Data Processing," IEEE Computer, November 1980.
13. D. L. Eager, John Zahorjan, and E. D. Lazowska, "Speedup Versus Efficiency in Parallel Systems," IEEE Transaction on Computers, Vol. 38, No. 3, March 1989.
14. Kemal Efe, "Heuristic Models of Task Assignment Scheduling in Distributed Systems," Computer, pp. 50-56, June 1982.
15. J. W. Flower and S.W. Otto, "A Preprocessor for Irregular Finite Element Problems," C³P292, Caltech Report, June 1986.
16. G. C. Fox and etc, Solving Problems on Concurrent Processors, p. 307, Vol I, Prentice Hall, 1989.
17. Geoffrey C. Fox and Steve W. Otto, "Concurrent Computation and the Theory of Complex Systems," C³P255, Caltech Report, March 3, 1986.
18. Geoffrey C. Fox, "A Review of Automatic Load Balancing and Decomposition Methods for the Hypercube," C³P385, Caltech Report, December 5, 1986.
19. G. C. Fox, S.W. Otto, and E.A. Umland, "Monte Carlo Physics on a Concurrent Processor," Hm214, Caltech Report, November 6, 1985.
20. G. C. Fox, "A Review of Automatic Load Balancing and Decomposition Methods for the Hypercube," C³P385, Caltech Report, December 5, 1986.
21. G. C. Fox and W. Furmanski, "Load Balancing by a Neural Network," C³P363, Caltech Report, September 1986.
22. G. C. Fox, A. Kolawa, and R. Williams, "The Implementation of a Dynamic Load Balancer," C³P328, Caltech Report, November 24, 1986.
23. Michael R. Garey and David S. Johnson, Computers and Intractability, A Guide to the Theory of NP-Completeness, W. H. Freeman and Company, San Francisco, 1979.
24. Stuart Geman and Donald Geman, "Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images," IEEE Transactions on Pattern Analysis and Machine Intelligence, 1984.

25. A. J. Harget and I. D. Johnson, "Load Balancing Algorithms in Loosely-coupled Distributed Systems: A Survey," Distributed Computer Systems, Theory and Practice, Butterworths, 1990.
26. S. Kirkpatrick, C.D. Gelatt, Jr., and M.P. Vecchi, "Optimization by Simulated Annealing," *Science* 220:671-680, 1983.
27. Edward D. Lazowska, "The Architecture of the Eden System," *SIGOPS, ACM*, Vol. 15, No. 5, December 1981.
28. Virginia Mary Lo, "Heuristic Algorithm for Task Assignment in Distributed Systems," *IEEE Transactions on Computers*, Vol. 37, No. 11, November 1988.
29. Perng-Yi Ma, Y. S. Lee, and Tsuchiya Masahiro, "A Task Allocation Model for Distributed Computing Systems," *IEEE Transactions on Computers*, Vol. C-31, No. 1, January 1982.
30. Michael P. McLaughlin, "Simulated Annealing," *Dr. Dobb's Journal*, September 1989.
31. R. Morison and S. Otto, "The Scattered Decomposition for Finite Elements," C³P286, Caltech Report, May 19, 1985.
32. R. M. Needham and A. J. Herbert, The Cambridge Distributed Computing System, Addison Wesley Publishing Company, 1982.
33. Lionel M. Ni and Kai Hwang, "Optimal Load Balancing in a Multiple Processor System with Many Job Classes," *IEEE Transactions on Software Engineering*, Vol. SE-11, No. 5, May 1985.
34. David M. Nicol and Paul F. Reynolds, Jr., "Optimal Dynamic Remapping of Data Parallel Computations," *IEEE Transactions on Computers*, Vol. 39, No. 2, February 1990.
35. Christos Nikolaou and Gautam Kar, "Allocation and Relocation of Process in Distributed Computer Systems," Current Advances in Distributed Computing and Communications, Computer Science Press.
36. Joseph Pasquale, "Using Expert Systems to Manage Distributed Computer Systems," *IEEE Network*, pp. 22-28, September 1988.

37. Carsten Peterson and Eric Hartman, "Explorations of the Mean Field theory Learning Algorithm," *Neural Networks*, Vol. 2, pp. 475-494, 1989.
38. G. Popek and et al., "LOCUS, A Network Transparent, High Reliability Distributed System," *SIGOPS, ACM*, Vol. 15, No. 5, December 1981.
39. William H. Press, "Combinatorial Minimization: Method of Simulated Annealing," Numerical Recipes in C, p. 343, Cambridge University Press.
40. David E. Rumelhart and James L. McClelland, Parallel Distributed Processing, Vol. 1, The MIT Press, 1987.
41. R. Serra and G. Zanarini, Complex Systems and Cognitive Processes, p.102, Springer-Verlag, 1990.
42. Chien-Chung Shen and Wen-Hsiang Tsai, "A Graph Matching Approach to Optimal Task Assignment in Distributed Computing Systems Using a Minimax Criterion," *IEEE Transactions on Computers*, Vol. C-34, No. 3, March 1985.
43. Kang G. Shin and Yi Chieh Chang, "Load Sharing in Distributed Real-Time Systems with State-Change Broadcasts," *IEEE Transactions on Computers*, Vol. 38, No. 8, August 1989.
44. Reid G. Smith, "The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver," *IEEE Transactions on Computers*, Vol. C-29, No. 12, December 1980.
45. William Stallings, Data and Computer Communications, Wesley Publishing Company, 1985.
46. John A. Stankovic, "A Perspective on Distributed Computer Systems," *IEEE Transactions on Computers*, Vol. C-33, No. 12, December 1984.
47. John A. Stankovic, "An Application of Bayesian Decision Theory to Decentralized Control of Job Scheduling," *IEEE Transactions on Computers*, Vol. C-34, No. 2, February 1985.
48. John A. Stankovic and Inderjit S. Sidhu, "An Adaptive Bidding Algorithm for Processes, Clusters and Distributed Groups," CH2021-4/84/000/0049, IEEE, 1984.

49. Harold S. Stone, "Multiprocessor Scheduling with the Aid of Network Flow Algorithms," IEEE Transactions on Software Engineering, Vol. SE-3, No. 1, January 1977.
50. Harold Szu and Ralph Hartley, "Fast Simulated Annealing," Physics Letters A, Vol. 122, No. 3,4, June 8, 1987.
51. MasPar System Overview, MasPar Computer Corporation, Sunnyvale, CA 94086, PN9300-0100, May 1990.
52. N. Balram, C. Belo, and J.M.F. Moura, "Parallel Processing on Supercomputers: A Set of Computational Experiments," CH2617-9/88/0000/0247, IEEE, 1988.
53. Alok N. Choudhary and Janak H. Patel, "Load Balancing and Task Decomposition Techniques for Parallel Implementation of Integrated Vision Systems Algorithms," ACM, pp. 266-275, 1989.
54. Ajit Singh, "A Template-Based Approach to the Generation of Distributed Applications Using a Network of Workstations," IEEE Transactions on Parallel and Distributed Systems, Vol. 2, No. 1, January 1991.
55. J. M. Swisshelm, "Multitasking a Three-Dimensional Navier-Stokes Algorithm on the Cray-2," Supercomputing 88, Volume II: Science and Applications, IEEE Computer Society Press, 1989.

APPENDIX A: COMPUTATION TIME
OF THE HCA

Proposition : The computation time of the heuristic clustering algorithm (HCA) applied in the CSAM is $O(N^3)$, where N is the number of tasks in the assignment.

Proof : Since the clustering algorithm iterates $(N-1)$ times from step 3 through step 5, the computation time will be $(N * \text{the largest computation time in one of the steps})$. Let $CT(i)$ denote the computation time in step i . Let K be the number of clusters, $2 \leq K \leq N$. Let X_i be the number of tasks in cluster C_i and X be the average value of X_i s, then, $N = \sum_i X_i = K * X$, where $1 \leq i \leq K$. The computation time in each step are:

$$\begin{aligned} CT(3) &= \sum_{i=1}^K (X_i^2 - X_i) \\ &\leq (N-1)^2 + (K-1) - KX \\ &\leq (N-1)^2 - 1 \\ &= O(N^2). \end{aligned}$$

$$\begin{aligned} CT(4) &= \sum_{i=1}^K \sum_{j=i+1}^K (X_i X_j) \\ &\leq \frac{1}{2} K (K-1) X^2 \\ &= O(N^2). \end{aligned}$$

$$\begin{aligned} CT(5) &= X_i \left[\sum_{j=1}^K (2 X_j) \right] \\ &= 2 X_i N \\ &= O(N^2). \end{aligned}$$

where

$$\begin{aligned} 1 &\leq X_i \leq (N-1) \\ 2 &\leq K \leq N. \end{aligned}$$

In step 3, $\sum_x \sum_y ITC_{Tx, Ty}$ is computed for each of the K clusters, i.e., $\sum_i (X_i^2 - X_i)$. The worst case occurs when most of the tasks are resident at one cluster while other clusters have one task only. Since $X_i \leq (N-1)$, the time complexity for computing $\sum_x \sum_y ITC_{Tx, Ty}$ will be $(N-1)^2$ for the

largest cluster and be 1 for other clusters. In step 4, $\sum_j$ ICC_{c_i, c_j} is computed for each C_i . The computation time for ICC_{c_i, c_j} is $X_i * X_j$. Since $ICC_{c_i, c_j} = ICC_{c_j, c_i}$ and $ICC_{c_i, c_i} = 0$, the computation time for this step is $\sum_i \sum_{i+1} (X_i X_j)$. The worst case is when $I = C$, i.e., all the clusters in C are selected into I from step 3. This situation causes a time Complexity of $1/2[K(K-1)]$ for computing the ICC's among the clusters. In addition, the worst case in computing the ICC's is when tasks are evenly distributed among the clusters. Therefore, the time complexity of this step is $\sum_i \sum_{i+1} (X_i X_j) \leq \sum_i \sum_{i+1} (X^2) = 1/2[K(K-1)(X^2)]$. In step 5, the computation of $(Z_j - \sum_y ITC_{T_x, T_y})$ is executed for each T_x in C_i and for each C_j in C , where $i \neq j$. The time complexity for $(Z_j - \sum_y ITC_{T_x, T_y})$ is $2X_j$. Therefore, merging each T_x in C_i to other clusters C_j 's, step 5 requires a time complexity of $X_i[\sum_j(2X_j)]$ which is equal to $2X_i N$. Since the time complexity for each step (CT(3) to CT(5)) is less than N^2 , the computation time for the algorithm is $O(N^3)$. ■

APPENDIX B: DERIVATION OF
GIBB'S DISTRIBUTION

In an isolated system at equilibrium, the probability distribution of the states $p(X,t)$ is given by the Gibbs distribution¹

$$p(X) = \frac{e^{-\beta E(X)}}{\sum_Y e^{-\beta E(Y)}}.$$

The evolution of $p(X,t)$, under fairly general conditions, may be described by the so called master equation:

$$\begin{aligned} p(X, t+\Delta t) &= \sum_Y w(X, \Delta t | Y, 0) p(Y, t) \\ &\quad - w(X, \Delta t | X, 0) p(X, t) + \sum_{Y \neq X} w(X, \Delta t | Y, 0) p(Y, t). \end{aligned}$$

From the normalization condition for the transition probability densities:

$$\sum_Z w(Z, \Delta t | X, 0) = 1,$$

we have

$$w(X, \Delta t | X, 0) = 1 - \sum_{Z \neq X} w(Z, \Delta t | X, 0).$$

By substituting in the master equation we obtain:

$$\begin{aligned} p(X, t+\Delta t) - p(X, t) &= \sum_{Y \neq X} w(Z, \Delta t | Y, 0) p(Y, t) \\ &\quad - \sum_{Z \neq X} w(Z, \Delta t | X, 0) p(X, t). \end{aligned}$$

We then examine the conditions under which the Gibbs distribution is a stationary solution of the master equation:

¹Derivation of the transition probability $W(Y|X)$ is given by [41] (Serra, 1987, pp. 102-103).

$$0 = \sum_{Y \neq X} (w(X, \Delta t | Y, 0) e^{-\beta E(Y)} - w(Y, \Delta t | X, 0) e^{-\beta E(X)}).$$

A sufficient condition for this is that every addendum of the sum in Y is zero, i.e.,

$$w(X, \Delta t | Y, 0) e^{-\beta E(Y)} - w(Y, \Delta t | X, 0) e^{-\beta E(X)} = 0.$$

This condition, known as detailed balance, is satisfied if

$$\begin{aligned} w(X, \Delta t | Y, 0) &= A(X | Y) e^{-\beta E(X)} \\ w(Y, \Delta t | X, 0) &= A(Y | X) e^{-\beta E(Y)}. \end{aligned}$$

provided that:

$$A(X | Y) = A(Y | X).$$

The normalization condition can then be rewritten as

$$\sum_X A(X | Y) e^{-\beta E(X)} = 1.$$

If all the coefficients $A(X, Y)$ are taken as being equal to a constant A , we have

$$A = \frac{1}{\sum_X e^{-\beta E(X)}} = \frac{1}{Z}.$$

where Z is the partition function.

As an alternative, it is possible to put some $A(X, Y)$ equal to zero without violating the condition of detailed balance.

According to the choices made, various dynamics are obtained. For example, one may impose that X and Y do not differ from one another for the states of more than a given number of cells. In the asynchronous dynamics the given number is 1. The idea is that, if the time increments are sufficiently small, it is plausible that only one spin variable per step is allowed to update. Thus, $A(X | Y)$ is not zero only if the states X and Y differ at the most by one

element. In this case the transition probability from the current state, say X , to the state Y is

$$w(Y|X) = \frac{e^{-\beta E(Y)}}{e^{-\beta E(Y)} + e^{-\beta E(X)}} .$$

$$= \frac{1}{1 + e^{-\beta (E(X) - E(Y))}} .$$

This formula is the starting point for computer simulations. If one takes the limit $\beta \rightarrow \infty$, which is equivalent to the zero temperature limit ($\beta = 1/T$) one finds that the transaction probability $w(Y|X)$ vanishes if the state Y has greater energy than the initial state X , whereas it takes a value of 1 if $E(Y) < E(X)$.

APPENDIX C: A PARALLEL VERSION
OF THE MFAM

In the parallel version of the MFAM, each of the spins is updated by a process element (PE) at the same time. Each PE has its local memory containing the values of all the spins. When a spin is updated, the updated value is broadcast to all the other PEs, providing synchronization on the spins. Since all the spins are updated in parallel, some of the spins may be conflict to each other, i.e., more than one spin in a row may have values close to 1, indicating that a task should be allocated to more than one nodes. To avoid such situation, we use the approach of [2] in which a small distance is moved from the old mean field to the new mean field for each update, where the mean fields (MF) is defined as $E_0 - E_1$. In addition, after each spin update, we normalize the spins in each row so that sum of the spins in a row equals to 1. The algorithm is given below.

```

1.    parallel( i=1 to m; j=1 to n )
2.        {  $S_{ij} \leq 1/n$ 
3.             $MF_{ij} \leq 0$ 
4.        }
5.    T <= Starting temperature
6.    while( T >  $T_{min}$  )
7.        { do until(  $S_{ik}$  stabilized for all spins )
8.            { parallel( i=1 to m; j=1 to n )
9.                { for( k=1; k  $\leq$  n, k  $\neq$  j; k=k+1 )
10.                     $S_{ik} \leq S_{ik} + S_{ij}/(n-1)$ 
11.                     $S_{ij} \leq 0.0$ 
12.                     $E_0 \leq (3.4)$ 
13.                     $S_{ij} \leq 1.0$ 
14.                    for( k=1; k  $\leq$  n, k  $\neq$  j; k=k+1)
15.                         $S_{ik} \leq 0.0$ 
16.                         $E_1 \leq (3.4)$ 
17.                         $MF_{ij} \leq MF_{ij} + \mu( E_0 - E_1 - MF_{ij} )$ 
18.                         $S_{ij} \leq [1 + \exp(-MF_{ij}/T)]^{-1}$ 
19.                    }
20.                parallel( i=1 to m )

```

```

21.           Normalize  $S_{ij}$  on row  $i$ 
22.           Synchronize  $S_{ij}$  on all PEs
23.       }
24.        $T \leq \alpha T$ , where  $\alpha$  is some constant,  $0 < \alpha < 1$ ;
25.   }
```

The algorithm begins by initializing each $S_{ij} = 1/n$ and $MF_{ij} = 0$, for all i and j (lines 1 - 4). A high temperature is set to allow more relaxation in reaching to a global minimum (line 5). The process from lines 6 to line 25 are then iterated until $T \leq T_{\min}$. From line 8 to line 19, all the spins are updated in parallel. The value μ (line 17) used in the simulation program is set to be 0.1. The new spin is calculated at line 18. The updated spins, each contained in a PE, is then normalized (lines 20 - 21). Finally, all the spins are synchronized on all the PEs and ready for the next iteration (line 22). When all the spin variables become stable, we reduce the temperature T (line 24) and the same iteration starts again. By stable, we mean that the value of S_{ij} is within the range, $S_{ij} \pm \epsilon$, where ϵ is some threshold. At a sufficiently low value of T , all the spins will converge to 0 or 1, which indicates a local or global minimum of the energy.

APPENDIX D: LIST OF INPUT DATA FOR
THE SIMULATIONS

Simulation Part II

N_Node = 100

TET Average = 5

ITC Average = 20

TL Average = 1

Node No.	Task Creation Rate	Task Completion Rate	TET Range			ITC Range			# of Cluster	# of Over-Lap
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----
00	2	7	4	-	6	10	-	30	1	0
01	1	7	4	-	6	10	-	30	1	0
02	3	7	4	-	6	10	-	30	1	0
03	4	7	4	-	6	10	-	30	1	0
04	3	7	4	-	6	10	-	30	1	0
05	2	7	4	-	6	10	-	30	1	0
06	1	7	4	-	6	10	-	30	1	0
07	3	7	4	-	6	10	-	30	1	0
08	2	7	4	-	6	10	-	30	1	0
09	4	7	4	-	6	10	-	30	1	0
10	3	7	4	-	6	10	-	30	1	0
11	40	7	2	-	32	10	-	30	10	1
12	2	7	4	-	6	10	-	30	1	0
13	1	7	4	-	6	10	-	30	1	0
14	1	7	4	-	6	10	-	30	1	0
15	3	7	4	-	6	10	-	30	1	0
16	2	7	4	-	6	10	-	30	1	0
17	4	7	4	-	6	10	-	30	1	0
18	30	7	2	-	26	10	-	30	6	1
19	2	7	4	-	6	10	-	30	1	0
20	1	7	4	-	6	10	-	30	1	0
21	0	7	4	-	6	10	-	30	1	0
22	1	7	4	-	6	10	-	30	1	0
23	2	7	4	-	6	10	-	30	1	0
24	3	7	4	-	6	10	-	30	1	0
25	4	7	4	-	6	10	-	30	1	0
026	3	7	4	-	6	10	-	30	1	0
27	1	7	4	-	6	10	-	30	1	0
28	3	7	4	-	6	10	-	30	1	0
29	1	7	4	-	6	10	-	30	1	0
30	20	7	2	-	12	10	-	30	5	1
31	2	7	4	-	6	10	-	30	1	0
32	3	7	4	-	6	10	-	30	1	0
33	30	7	3	-	7	10	-	30	6	1
34	1	7	4	-	6	10	-	30	1	0
35	0	7	4	-	6	10	-	30	1	0
36	40	7	2	-	10	10	-	30	7	1
37	0	7	4	-	6	10	-	30	1	0
38	2	7	4	-	6	10	-	30	1	0
39	0	7	4	-	6	10	-	30	1	0
40	1	7	4	-	6	10	-	30	1	0

Simulation Part II (continue)

Node No.	Task Creation Rate	Task Completion Rate	TET Range			ITC Range			# of Cluster	# of Over- Lap
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----
41	0	7	4	-	6	10	-	30	1	0
42	1	7	4	-	6	10	-	30	1	0
43	1	7	4	-	6	10	-	30	1	0
44	3	7	4	-	6	10	-	30	1	0
45	1	7	4	-	6	10	-	30	1	0
46	0	7	4	-	6	10	-	30	1	0
47	3	7	4	-	6	10	-	30	1	0
48	2	7	4	-	6	10	-	30	1	0
49	0	7	4	-	6	10	-	30	1	0
50	3	7	4	-	6	10	-	30	1	0
51	30	7	3	-	7	10	-	30	6	1
52	4	7	4	-	6	10	-	30	1	0
53	3	7	4	-	6	10	-	30	1	0
54	2	7	4	-	6	10	-	30	1	0
55	4	7	4	-	6	10	-	30	1	0
56	1	7	4	-	6	10	-	30	1	0
57	3	7	4	-	6	10	-	30	1	0
58	1	7	4	-	6	10	-	30	1	0
59	0	7	4	-	6	10	-	30	1	0
60	2	7	4	-	6	10	-	30	1	0
61	1	7	4	-	6	10	-	30	1	0
62	30	7	4	-	6	10	-	30	6	1
63	2	7	4	-	6	10	-	30	1	0
64	1	7	4	-	6	10	-	30	1	0
65	2	7	4	-	6	10	-	30	1	0
66	4	7	4	-	6	10	-	30	1	0
67	0	7	4	-	6	10	-	30	1	0
68	30	7	2	-	12	10	-	30	6	1
69	1	7	4	-	6	10	-	30	1	0
70	3	7	4	-	6	10	-	30	1	0
71	3	7	4	-	6	10	-	30	1	0
72	2	7	4	-	6	10	-	30	1	0
73	2	7	4	-	6	10	-	30	1	0
74	1	7	4	-	6	10	-	30	1	0
75	3	7	4	-	6	10	-	30	1	0
76	0	7	4	-	6	10	-	30	1	0
77	2	7	4	-	6	10	-	30	1	0
78	3	7	4	-	6	10	-	30	1	0
79	2	7	4	-	6	10	-	30	1	0
80	0	7	4	-	6	10	-	30	1	0
81	1	7	4	-	6	10	-	30	1	0
82	3	7	4	-	6	10	-	30	1	0
83	1	7	4	-	6	10	-	30	1	0
84	0	7	4	-	6	10	-	30	1	0
85	30	7	3	-	7	10	-	30	6	1
86	1	7	4	-	6	10	-	30	1	0

Simulation Part II (continue)

Node No.	Task Creation Rate	Task Completion Rate	TET Range	ITC Range	# of Cluster	# of Over-Lap
-----	-----	-----	-----	-----	-----	-----
87	2	7	4 - 6	10 - 30	1	0
88	1	7	4 - 6	10 - 30	1	0
89	2	7	4 - 6	10 - 30	1	0
90	2	7	4 - 6	10 - 30	1	0
91	0	7	4 - 6	10 - 30	1	0
92	2	7	4 - 6	10 - 30	1	0
93	2	7	4 - 6	10 - 30	1	0
94	3	7	4 - 6	10 - 30	1	0
95	2	7	4 - 6	10 - 30	1	0
96	1	7	4 - 6	10 - 30	1	0
97	4	7	4 - 6	10 - 30	1	0
98	0	7	4 - 6	10 - 30	1	0
99	3	7	4 - 6	10 - 30	1	0

Simulation Part IV

N_Node = 100

TET Average = 10

ITC Average = 5

TL Average = 1

Node No.	Task Creation Rate	Task Completion Rate	TET Range	ITC Range	# of Cluster	# of Over-Lap
-----	-----	-----	-----	-----	-----	-----
00	8	8	3 - 17	0 - 10	1	0
01	5	8	3 - 17	0 - 10	1	0
02	6	8	3 - 17	0 - 10	1	0
03	6	8	3 - 17	0 - 10	1	0
04	6	8	3 - 17	0 - 10	1	0
05	8	8	3 - 17	0 - 10	1	0
06	2	8	3 - 17	0 - 10	1	0
07	10	8	3 - 17	0 - 10	1	0
08	6	8	3 - 17	0 - 10	1	0
09	4	8	3 - 17	0 - 10	1	0
10	7	8	3 - 17	0 - 10	1	0
11	9	8	3 - 17	0 - 10	1	0
12	7	8	3 - 17	0 - 10	1	0
13	4	8	3 - 17	0 - 10	1	0
14	7	8	3 - 17	0 - 10	1	0
15	6	8	3 - 17	0 - 10	1	0
16	7	8	3 - 17	0 - 10	1	0
17	3	8	3 - 17	0 - 10	1	0
18	4	8	3 - 17	0 - 10	1	0
19	2	8	3 - 17	0 - 10	1	0

Simulation Part IV (continue)

Node No.	Task Creation Rate	Task Completion Rate	TET Range	ITC Range	# of Cluster	# of Over- Lap
-----	-----	-----	-----	-----	-----	-----
20	2	8	3 - 17	0 - 10	1	0
21	4	8	3 - 17	0 - 10	1	0
22	7	8	3 - 17	0 - 10	1	0
23	7	8	3 - 17	0 - 10	1	0
24	2	8	3 - 17	0 - 10	1	0
25	9	8	3 - 17	0 - 10	1	0
26	7	8	3 - 17	0 - 10	1	0
27	5	8	3 - 17	0 - 10	1	0
28	7	8	3 - 17	0 - 10	1	0
29	4	8	3 - 17	0 - 10	1	0
30	10	8	3 - 17	0 - 10	1	0
31	6	8	3 - 17	0 - 10	1	0
32	10	8	3 - 17	0 - 10	1	0
33	10	8	3 - 17	0 - 10	1	0
34	5	8	3 - 17	0 - 10	1	0
35	4	8	3 - 17	0 - 10	1	0
36	8	8	3 - 17	0 - 10	1	0
37	4	8	3 - 17	0 - 10	1	0
38	9	8	3 - 17	0 - 10	1	0
39	6	8	3 - 17	0 - 10	1	0
40	4	8	3 - 17	0 - 10	1	0
41	10	8	3 - 17	0 - 10	1	0
42	7	8	3 - 17	0 - 10	1	0
43	7	8	3 - 17	0 - 10	1	0
44	8	8	3 - 17	0 - 10	1	0
45	8	8	3 - 17	0 - 10	1	0
46	4	8	3 - 17	0 - 10	1	0
47	10	8	3 - 17	0 - 10	1	0
48	4	8	3 - 17	0 - 10	1	0
49	4	8	3 - 17	0 - 10	1	0
50	10	8	3 - 17	0 - 10	1	0
51	2	8	3 - 17	0 - 10	1	0
52	7	8	3 - 17	0 - 10	1	0
53	5	8	3 - 17	0 - 10	1	0
54	8	8	3 - 17	0 - 10	1	0
55	8	8	3 - 17	0 - 10	1	0
56	7	8	3 - 17	0 - 10	1	0
57	7	8	3 - 17	0 - 10	1	0
58	3	8	3 - 17	0 - 10	1	0
59	2	8	3 - 17	0 - 10	1	0
60	6	8	3 - 17	0 - 10	1	0
61	6	8	3 - 17	0 - 10	1	0
62	3	8	3 - 17	0 - 10	1	0
63	6	8	3 - 17	0 - 10	1	0
64	1	8	3 - 17	0 - 10	1	0
65	8	8	3 - 17	0 - 10	1	0

Simulation Part IV (continue)

Node No.	Task Creation Rate	Task Completion Rate	TET Range	ITC Range	# of Cluster	# of Over- Lap
-----	-----	-----	-----	-----	-----	-----
66	5	8	3 - 17	0 - 10	1	0
67	3	8	3 - 17	0 - 10	1	0
68	7	8	3 - 17	0 - 10	1	0
69	5	8	3 - 17	0 - 10	1	0
70	2	8	3 - 17	0 - 10	1	0
71	10	8	3 - 17	0 - 10	1	0
72	8	8	3 - 17	0 - 10	1	0
73	2	8	3 - 17	0 - 10	1	0
74	2	8	3 - 17	0 - 10	1	0
75	3	8	3 - 17	0 - 10	1	0
76	5	8	3 - 17	0 - 10	1	0
77	7	8	3 - 17	0 - 10	1	0
78	6	8	3 - 17	0 - 10	1	0
79	10	8	3 - 17	0 - 10	1	0
80	3	8	3 - 17	0 - 10	1	0
81	8	8	3 - 17	0 - 10	1	0
82	6	8	3 - 17	0 - 10	1	0
83	9	8	3 - 17	0 - 10	1	0
84	8	8	3 - 17	0 - 10	1	0
85	2	8	3 - 17	0 - 10	1	0
86	2	8	3 - 17	0 - 10	1	0
87	2	8	3 - 17	0 - 10	1	0
88	5	8	3 - 17	0 - 10	1	0
89	10	8	3 - 17	0 - 10	1	0
90	3	8	3 - 17	0 - 10	1	0
91	8	8	3 - 17	0 - 10	1	0
92	6	8	3 - 17	0 - 10	1	0
93	9	8	3 - 17	0 - 10	1	0
94	8	8	3 - 17	0 - 10	1	0
95	2	8	3 - 17	0 - 10	1	0
96	2	8	3 - 17	0 - 10	1	0
97	2	8	3 - 17	0 - 10	1	0
98	5	8	3 - 17	0 - 10	1	0
99	10	8	3 - 17	0 - 10	1	0