On File and Task Placements and Dynamic Load Balancing in Distributed Systems

Po-Jen Chuang and Chi-Wei Cheng

Department of Electrical Engineering
Tamkang University
Tamsui,Taiwan 251, R.O.C.
E-mail: pjchuang@ee.tku.edu.tw

Abstract

Two distributed system problems, the file and task placement problem and the dynamic load balancing problem, are investigated in this paper. To find the placement of files and tasks at sites with minimal total communication overhead, we propose using the Simulated Annealing approach and multiple objective functions. Experimental results show that our proposed approach depicts superior performance with much less complexity over the previously introduced Genetic Algorithm approach.

Dynamic load balancing is employed to equalize processor loads in a distributed system. It allows excessive tasks at a heavily loaded processor to be migrated to another processor with a light load during execution. To effectively lift up the acceptance rates for such task migration requests, we propose an efficient new scheme that yields much improved acceptance rates, followed by reduced unnecessary request messages and communication overhead, when compared with the standard sender-initiated scheme and the fairly complicated GA-based approach.

***Key Words*:** Distributed Systems, Dynamic Load Balancing, File and Task Placements, Genetic Algorithms, Objective Functions, Request Acceptance Rates, Sender-Initiated, Simulated Annealing

1. Introduction

Two distributed system problems, the file and task placement problem and the dynamic load balancing problem, are investigated in this paper. A distributed system is built to work on a distributed computing architecture composed of several sites which are connected by a communication network with a given topology. Each of the sites has its own memory and processors, stores a restricted number of files and is capable of running multiple tasks. Each task may access the files on its own site or access those on another site with some communication overhead that is determined by the topology and the speed of each link of the network. Hence, the main concern for the distributed file and task placement problem is to find the placement of files and tasks at the sites with *minimal total communication overhead*. To this end, a Genetic Algorithm (GA) has been developed [1].

The GA operates on a pool of chromosomes which represent candidate solutions to the problem. Chromosomes are selected following "survival of the fittest" and are passed on to the next generation in a process called "reproduction". An objective function is supplied and used to weigh the relative merits (the so-called "fitness value") of the chromosomes in the pool, and the reproduction is realized by the genetic operators, such as selection, crossover and mutation, to

generate new points in the search space. The GA approach claims better performance than the greedy heuristic approach in terms of communication overheads that arise from accessing the files required by the tasks [1]. Nevertheless, the genetic operations involved are quite complicated and time consuming. For improvement, we propose using the Simulated Annealing (SA) approach [2] together with multiple objective functions to upgrade the placement of files and tasks. The Guided Evolutionary Simulated Annealing (GESA) approach [3], an extension of the SA approach, is also evaluated to see its fitness for the placement problem. As experimental results demonstrate, our proposed SA approach depicts superior performance with much less complexity than both the GA and GESA approaches in obtaining desirable file and task placements.

On the other hand, in a distributed system where processors are loosely connected by a communication network, the random arrival of tasks at each processor is likely to bring about uneven loads, that is, some processors may be heavily loaded while others may not. To equalize the loads at all processors, dynamic load balancing is usually employed. It allows excessive tasks at a heavily loaded processor to be migrated to another processor with a light load during execution. Various dynamic load balancing schemes, such as the standard sender-initiated scheme [4] and the scheme based on the genetic algorithm (GA) [5], have been introduced - with certain disadvantages. For example, in the standard sender-initiated scheme, a request for task migration is initially issued from a heavily loaded processor (the sender) to a selected processor (the potential receiver) randomly. When the selected receiver is not lightly loaded, the request might be rejected and sent back and forth repeatedly until a suitable receiver is found. The whole process can be very time-consuming due to a large number of unnecessary requests. The GA-based approach attempts to reduce such unnecessary requests and by doing so to lift up the request acceptance rate, but with its complex genetic operations, the effort proves unavailing. To realize efficient dynamic load balancing, we present a new scheme that is able to yield more desirable results with much simplified operations. Experimental evaluation shows that with only slightly higher time complexity over the standard sender-initiated scheme, our simplified new scheme realizes very remarkable elevation in request acceptance rates when performing dynamic load balancing, significantly reducing the number of unnecessary request messages as well as communication overheads.

2. A Simulated Annealing Approach to Distributed File and Task Placements

The distributed file and task placement problem, known to be NP-complete [1], is indeed the generalization of both the file allocation problem in distributed systems and the fragment allocation problem in distributed databases [6-9]. As mentioned, the major concern for such a problem is to find the placement of files and tasks at the sites with minimal total communication overhead so that the execution time can be trimmed down accordingly. As the genetic algorithm (GA) developed for the purpose [1] involves too complicated and time consuming operations, we propose using the simulated annealing (SA) approach together with multiple objective functions to obtain more desirable file and task placements.

2.1 Background

A distributed system is composed of tasks, files and sites. Assume that α tasks (t_1, t_2, …, t_α) are to be run, β files (of sizes f_1, f_2, …, f_β) are needed by the tasks, and γ sites (of storage capacity s_1, s_2, …, s_γ) can execute tasks and store files. Each file is required by at least one of the tasks. The distributed file and task placement problem is to find the placement of the β files at sites $p_f(j)$ ($1 \leqq j \leqq \beta$) and α tasks at sites $p_t(i)$ ($1 \leqq i \leqq \alpha$), where $1 \leqq p_t(i), p_f(j) \leqq \gamma$, with minimal total communication overhead. The feasibility of the placement must subject to the problem constraints, e.g. the total size of the files placed in a site can not exceed the storage capacity of the site.

As an adaptive search technique, the genetic algorithm (GA) provides an alternative to traditional optimization techniques by using directed random searches to locate optimal or near optimal solutions of complex problems and is rooted in the mechanisms of evolution and natural genetics [10-11]. It operates on a pool of chromosomes which represent candidate solutions to the problem under investigation. The placement problem is encoded into a chromosome with two parts: In one part, each gene represents the placement for a particular task; in the other, each represents that for a particular file. Thus the length of the chromosome is equal to the total numbers of tasks and files.

The GA selects chromosomes following "survival of the fittest" and passes them on to the next generation in a process called "reproduction". An objective function is supplied and used to weigh the fitness values of the chromosomes. Since the placement of the β files and α tasks must ensure the minimization of the total communication overhead, an objective function — the total communication overhead $\sum_{i=1}^{\alpha}\sum_{j=1}^{\beta} r_{i,j} C_{p_t(i),p_f(j)}$ - needs to be minimized. It is assumed in the function that $r_{i,j}$ is a boolean value to indicate whether file j (f_j) is required by task i (t_i), and $C_{pt(i),\ pf(j)}$ indicates the least communication overhead for t_i at site $p_t(i)$ to access f_j at site $p_f(j)$. Besides, to ensure the feasibility of each candidate solution, it is subject to the constraint that the number of tasks at any site, $\Sigma_{\{i|pt(i)=k\}}$ 1 for all k's where $1 \leqq k \leqq \gamma$, is minimized (for balancing the task loads among all sites), and that the aggregate capacity of any site is not exceeded, that is, $\Sigma_{\{j|pf(j)=k\}} f_j \leqq s_k$ for all k's where $1 \leqq k \leqq \gamma$. Reproduction is realized by genetic operators, such as selection, crossover and mutation, to generate new points in the search space.

As mentioned, a *single* objective function is considered in [1] to calculate the fitness value of each chromosome, with the communication overhead taking into account only the tasks' accessing of required files at remote sites. To be more practical, the communication overhead due to task dependencies is also counted in our paper, and the constraint for pursuing load balancing in [1] also becomes an objective function in which total task length, instead of the number of tasks, at each site is considered. Thus *multiple* objective functions are supplied and properly used in our paper to calculate the fitness value of each candidate solution and the optimal solution aims to minimize the multiple objective functions, not just to minimize a single one. Similar minimization can be found in [12].

For solving the placement problem, the GA approach claims superior performance than the greedy heuristic approach in terms of communication overheads that result from accessing the files required by tasks, but its operations are too intricate and time consuming. (The greedy heuristic approach first assigns tasks to sites uniformly throughout the network of sites to ensure "absolute" load balancing. Then each file is placed to the feasible site so as to minimize the total communication overhead.) To attain more desirable performance, we adopt the simulated annealing (SA) approach [2] and the use of multiple objective functions.

2.2 The Simulated Annealing Approach

The SA approach is another alternative to traditional optimization techniques. Basically, it is an iterative random search procedure with adaptive moves to locate optimal or near optimal solutions of complex problems. As the name indicates, it needs an *annealing* schedule of the temperatures besides a random generator of "moves" and an objective function. By permitting "uphill moves" under the control of probabilistic criterion (a Boltzmann machine-like mechanism), the temperature is able to keep the algorithm from getting stuck. With the higher the temperatures, the larger the probability to do "uphill moves", it tends to avoid the first local minima encountered. The approach has been successfully applied in different combinatorial optimizations, such as the Travel Salesman Problem [2].

The SA approach randomly generates one initial solution which then generates a new solution based on the neighborhood structure. The two solutions then compete by using the Boltzmann machine-like mechanism. In the process of minimization, if the objective function value of the new solution is lower than that of the initial one, the new solution is selected. If the new solution has a higher value, it can still be selected under some probability which is usually assumed to result from the Boltzmann probability distribution function of the objective function value difference between the two "competing" solutions. The selected solution will generate another new solution and the competing process is repeated again. The iterative process continues until convergence or for a specified length of times. To solve the distributed file and task placement problem through our proposed SA approach, we let the solution be encoded in the same way as the chromosome in the GA approach (say $p_t(1)p_t(2)\ldots p_t(\alpha)p_f(1)p_f(2)\ldots p_f(\beta)$ for α tasks and β files). For convenience, each of $p_t(i)$, $1 \leqq i \leqq \alpha$, and $p_f(j)$, $1 \leqq j \leqq \beta$, is called an *element* of the encoded solution.

To be more practical, multiple objective functions are considered. The constraint for pursuing load balancing considered in [1] now becomes an objective function

$$max \sum_{\{i|p_t(i)=k\}} t_i \ \ for\ all\ k\ where\ 1 \leqq k \leqq \gamma \qquad (2.1)$$

in which total task length (assume that each task i

is characterized by its task length t_i), rather than the number of tasks, at each site is considered because the task length is a better indicator of the "load". If expression (2.1) is minimized, the load could become balanced. We also believe that when expression (2.1) is considered as an objective function instead of a constraint, with a suitable value it can be helpful to reach solutions with much less communication overhead. In addition, we consider the communication overhead not only due to the tasks' accessing of the required files at remote sites (considered in [1] as a single objective function)

$$\sum_{i=1}^{\alpha}\sum_{j=1}^{\beta} r_{i,j} C_{p_t(i),p_f(j)} \tag{2.2}$$

but also due to task dependencies

$$\sum_{i=1}^{\alpha}\sum_{j=1}^{\beta} d_{i,j} C_{p_t(i),p_f(j)} \tag{2.3}$$

where $d_{i,j}$ is a boolean value indicating the dependency between tasks i and j. Thus, the total communication overhead can be calculated by summing up equations (2.2) and (2.3). Multiple objective functions considered by us are henceforth denoted by $\phi_1(x)$, $\phi_2(x)$, and $\phi_3(x)$, respectively representing expressions (2.1), (2.2) and (2.3). In fact, multiple objective functions have been used to evaluate and compare various solutions in [12].

The iterative process of the SA algorithm applied to the placement problem is given in the following.

(1) Initialization

Initialize the iteration count and the temperature. Generate one initial solution randomly. Set the initial solution the selected solution (x).

(2) Iterative steps

(a) Generation: Generate a new solution (x') from the selected solution (x) based on the neighborhood structure — that is, randomly choose an element from the selected encoded solution and change it to any other randomly chosen site number to generate a new solution. Calculate the difference between the objective function values of the new and the selected solutions, say $\Delta\phi_k = \phi_k(\mathrm{x}') - \phi_k(\mathrm{x})$, for $1 \leqq k \leqq 3$. If all the objective function values of the new solution are not higher than those of the selected one, that is when $\Delta\phi_k \leqq 0$ for $1 \leqq k \leqq 3$, the new solution becomes the selected solution. Otherwise, the new solution is selected with the probability $\exp\left(\frac{-\sum_{k=1}^{3}\frac{\Delta\phi_k J_k}{\phi_k(x)}}{T}\right) > r$, where T is the temperature at the iteration, r is a random number uniformly distributed between 0 and 1, and if $\Delta\phi_k > 0$, $J_k = 1$; otherwise, $J_k = 0$ (the same consideration as in [12]). Note that the above probability results from the Boltzmann probability distribution function to permit "uphill moves".

(b) Cooling (lowering the temperature to reduce the probability of "uphill moves")

(c) Convergence check

(d) Terminal check according to the initialized iteration count

2.3 Experimental Performance Comparison

The SA approach to solving the placement problem has been programmed to obtain desirable file and task placements. Extensive simulation runs are conducted to collect results from various data sets for both the GA approach and our SA approach with more practical objective functions. Under our simulation model, It is assumed as in [1] that a requirement matrix, say R, is built to indicate the files required by each task, i.e., an element in the matrix, say $r_{i,j}$, is a boolean value indicating the requirement by task i for file j. Each file is required by at least one of the tasks with the probability of 0.8, and the communication overhead between sites is provided by a matrix, say C, where an element, say $c_{i,j}$, indicates the least communication overhead needed for communication between sites i and j. Any two sites are adjacent with a given probability (= 0.8) which then determines the network topology. The least communication overhead required for each link traverse is assumed to follow a uniform distribution *uniform*(1,10) (i.e., the overhead is uniformly distributed between 1 and 10). If any two sites are not able to connect each other due to the network topology, their corresponding element in the communication overhead matrix is assigned a big constant as a "penalty" [13]. The penalty makes it possible to avoid selecting some unfeasible solutions and to ensure feasible ones. A task dependency matrix, say D, is also provided where an element, say $d_{i,j}$, is a boolean value indicating the dependency between tasks t_i and t_j, and the probability that any two tasks are dependent is assumed to be 0.8. The task length and file size distributions are both assumed to be *uniform*(1,10).

Simulation results under the above simulation model for the GA and SA approaches are listed in

Table 1 for comparison. (The results for the GA approach implemented by using a *steady-state* [14] and a *generational* [10,15] population model are shown in Table 2 for reference. In the steady-state model, a single pool is used; in the generational model, offsprings are saved in a separate pool until there are enough to replace the original pool. As can be seen, the results for both models are virtually the same.) The site capacity distribution for the four data sets, which represent rather large practical problems, are respectively assumed to be *uniform*(20,30), *uniform*(30,40), *uniform*(40,50) and *uniform*(50,55). Performance is evaluated from multiple objective functions. In the Tables, ϕ_1, ϕ_2 and ϕ_3 are the same as in Section 2.2. The total communication overhead can be calculated by $\phi_2 + \phi_3$, where $\phi_2 = \sum_{i=1}^{\alpha}\sum_{j=1}^{\beta} r_{i,j} C_{p_t(i),p_f(j)}$ (the communication overhead considered in [1]) and $\phi_3 = \sum_{i=1}^{\alpha}\sum_{j=1}^{\beta} d_{i,j} C_{p_t(i),p_f(j)}$ (the communication overhead due to task dependencies). These results, which are obtained over 15 independent runs with the average value represented by $\overline{\phi_1}$ (the same for $\overline{\phi_2}$ and $\overline{\phi_3}$), are reasonably accurate. For instance, for the first $\overline{\phi_1}$ value 17, given 95% confidence, the calculated confidence interval half-width over the 15 replications is 0.81, indicating we are 95% confident the true result would fall into the interval 17 ± 0.81, or equivalently, 17 ± 4.76%, with only less than 5% error.

Table 1. Simulation results for the GA and SA approaches

Data Set			GA (Generational population model)				SA			
t	f	s	$\overline{\phi_1}$	$\overline{\phi_2}$	$\overline{\phi_3}$	τ	$\overline{\phi_1}$	$\overline{\phi_2}$	$\overline{\phi_3}$	τ
30	50	20	17	2818	1521	50	22	2154	1174	17
40	60	20	21	5194	3131	52	28	3649	2350	27
60	100	20	27	12882	7161	116	36	9478	5347	63
120	200	40	30	49468	29032	395	38	39775	23380	250
t: the number of tasks f: the number of files s: the number of sites										

Table 2. Simulation results for the GA approach with different models

Data Set			GA (Generational population model)				GA (Steady state population model)			
t	f	s	$\overline{\phi_1}$	$\overline{\phi_2}$	$\overline{\phi_3}$	τ	$\overline{\phi_1}$	$\overline{\phi_2}$	$\overline{\phi_3}$	τ
30	50	20	17	2818	1521	50	18	2866	1500	93
40	60	20	21	5194	3131	52	20	5224	3150	111
60	100	20	27	12882	7161	116	28	12955	7129	236
120	200	40	30	49468	29032	395	30	49564	28978	805
t: the number of tasks f: the number of files s: the number of sites										

As observed from the Tables, the placements of files and tasks with the GA approach result in more communication overheads ($\overline{\phi_2} + \overline{\phi_3}$) than the SA approach by 24%~39%. It is also interesting to observe that the SA approach gives slightly more $\overline{\phi_1}$ value (for pursuing load balancing) for all the cases. The two facts practically demonstrate the advantage of putting the GA's load balancing constraint in our multiple objective functions to be considered with the total communication overhead. As a matter of fact, to reach the main goal of the placement problem, that is, to place files and tasks with minimal communication overhead, load balancing alone may not be helpful. For instance, the "absolute" load balancing achieved by the greedy heuristic approach results in much inferior performance and the GA approach, though attaining more balanced load for each site than the SA approach, apparently suffers worse communication overheads. By contrast, our SA approach is able to give suitable ϕ_1 values more effectively, which may result in less balanced load but can reduce overheads significantly.

Simulation runs are carried out in a Sun Sparc system. The times needed for the results (τ) are also collected. The GA approach is shown to take

more time for both the steady state and the generational population models when compared with the SA approach for every data set. To give an example, for the largest data set (120 tasks, 200 files and 40 sites), the SA takes only 250 seconds, while the GA takes an average of 805 and 395 seconds. The result indicates when employed to find the optimal file and task placement, our SA gives rise to better performance with reduced time complexity compared with the rather complicated GA.

2.4 The Guided Evolutionary SA Approach and Discussions

The Guided Evolutionary Simulated Annealing (GESA) approach, an extension of the SA approach, has been proposed in [3]. To see if the new approach can be applied to the placement problem with more desirable performance would be interesting. The GESA approach allows many candidate solutions (not just one as in the SA) to be generated at the same time, and the generation of new solutions are guided into promising regions through local and global competitions. A set of initial solutions, say M solutions, are generated randomly, each of which is called a "parent" of a family. Each parent then generates a new set of solutions, say N solutions, which are called the "children" of the family. The M parents and the $M \times N$ children then compete by using the Boltzmann machine-like mechanism and the best solution will be selected as the parent of the next generation. Apparently, the better a family is (i.e., having a large number of good solutions), the more family members will be selected as new parents - the amount of members the family will have in the next generation is thus determined.

Listed below are the iterative steps of the GESA algorithm applied to the placement problem.

(1) Initialization

Initialize the iteration count and the temperature as in the SA. Generate randomly a set of initial solutions, say M parents, $x_1 \sim x_M$. Find the best parent.

(2) Iterative steps

(a) Generation: Generate a set of solutions from each parent by choosing randomly an element and changing it to any randomly chosen site number. Find the best child in each family and then the global best child among the best children.

(b) Selecting parents of the next generation: The best child of each family is compared with its parent to yield the difference: $\triangle \phi_k = \phi_k(x_{i,j}) - \phi_k(x_i)$, where i is the ith family and $x_{i,j}$ is the jth child, i.e., the best child in the family. If all the objective function differences $\Delta\phi_k < 0$ for $1 \leqq k \leqq 3$, the best child is chosen as the parent of its family for the next generation. Otherwise, the best child will be accepted as the new parent with the probability $\exp(\frac{-\sum_{k=1}^{3} \frac{\Delta\phi_k' J_k}{\phi_k(x)}}{T}) > r$, where $\Delta\phi_k' = \phi_k(x_{i,j}) - \phi_k(x)$, J_k = 1 (if $\Delta\phi_k' > 0$) or 0 (otherwise), x is the best parent, T is the temperature at the iteration, and r is a random number uniformly distributed between 0 and 1.

(c) Calculating the number of members accepted to the next generation for each family: Every child of each family is compared with its parent. Find the difference $\Delta\phi_k = \phi_k(x_{i,j}) - \phi_k(x_i)$ for $1 \leqq k \leqq 3$, where i is the ith family and j is the jth child in the family. If all the objective function differences $\Delta\phi_k < 0$, the weight factor w_i of family i for the next generation increases by 1. Otherwise, w_i increases by 1 with the probability $\exp(\frac{-\sum_{k=1}^{3} \frac{\Delta\phi_k' J_k}{\phi_k(x)}}{T}) > \gamma$ (with the same representation as mentioned above except that x is the global best child here). After calculating all weight factors for all families, sum up these weight factors: $s = \sum_{i=1}^{M} w_i$. The number of accepted members for each family will be $A_i = M \times N \times \frac{w_i}{s}$.

(d) Cooling

(e) Convergence check

(f) Terminal check

The GESA approach is also programmed to obtain desirable file and task placements with simulation results depicted in Table 3. It is observed that the performance of the GESA approach does not, as anticipated, advance that of the SA approach (on the contrary, it is inferior) and its time complexity is also worse than the SA. Thus when applied to the file and task placement problem, the SA apparently demonstrates better performance with less complexity than both the GA and GESA. On the other hand, when employed to solve problems

with continuous solutions, both the GA and GESA approaches may work better than the SA approach since they are able to generate a number of candidate solutions at the same time (the SA approach finds only one solution at a time). For the distributed file and task placement problem whose solutions are not continuous or located in a specific area, the neighboring solution cannot be obtained in the same way as when searching for continuous solutions (i.e., by adding a "small value" to the initial solution); instead, it will be obtained by changing one element in the encoded initial solution while leaving the rest of the elements unchanged. To randomly select one element in an encoded solution and change it to another randomly chosen site number indicates changing the placement of either a file or a task, which will largely affect the entire system. If the change is with file/task placement, the files stored/load for some site will be changed accordingly and so will the total communication overhead of the system. The situation is quite different from that of searching for continuous solutions where the difference between the initial and neighboring solutions are very small. To sum up, since the solutions for the distributed file and task placement problem are not continuous, the search for solutions must be carried out by looking through more "possible" solutions. That is, performance will be bettered if more generations are involved in the search. In this sense, the SA approach can work more effectively because with its simplified computation for each generation, the SA is able to obtain the optimal solution by increasing the number of generations to be searched. As for the GA and GESA, since the computation for each generation is rather complicated and time consuming, there exists the dilemma: If the generations are increased to ensure a more favorable solution, the search can be fairly lengthy; if the generations are decreased to trim down time overheads, the solution may turn out unsatisfactory.

Table 3. Simulation results for the GESA approach

Data Set			GESA			
t	f	s	$\overline{\phi_1}$	$\overline{\phi_2}$	$\overline{\phi_3}$	τ
30	50	20	18	3271	1814	50
40	60	20	24	5777	3642	71
60	100	20	32	13079	8032	158
120	200	40	36	50074	29611	665
t: the number of tasks f: the number of files s: the number of sites						

3. An Efficient Dynamic Load Balancing Scheme for Distributed Systems

As mentioned, the random arrival of tasks at each processor is likely to bring about uneven processor loads in a distributed system. To equalize processor loads, dynamic load balancing is usually employed in which excessive tasks at a heavily loaded processor can be migrated to another processor with a light load during execution. To realize efficient dynamic load balancing then means to lift up the acceptance rate of task migration requests and to trim down communication overheads accordingly. The standard sender-initiated scheme [4] and the GA-based scheme [5] are schemes established to balance such processor loads, but they either ignore enhancing the acceptance rate or involve too complicated operations. For improvement, we propose a simplified new scheme which is very efficient in performing dynamic load balancing in a distributed system.

3.1 Background

To balance the uneven processor loads in a distributed system, thresholds (expressed in units of load) can be employed [4]. If the load at a processor exceeds a threshold T_h, it is heavy; if falling below a threshold T_l, it is light. In the sender-initiated dynamic load balancing scheme [4], a request for task migration is initially issued from a processor with a heavy load (the sender) to another randomly selected processor (the potential receiver). If the selected receiver is not lightly loaded, the request might be rejected and sent back and forth repeatedly until a suitable receiver is found. To restrain the overhead, the number of requests is restricted by a request limit. Task migration will not happen if no suitable receiver is found within the request limit, and the sender processor will have to execute the task itself. The whole process can be very time-consuming due to unnecessary requests.

To realize efficient dynamic load balancing, the acceptance rate for migration requests must be uplifted in the first place, that is, a potential receiver should be decided swiftly and a "qualified" receiver must be selected within less tries. The simplest way to decide a potential receiver is, as in the standard sender-initiated scheme, to select it randomly [4,16]. But, having no concern for the acceptance rate, it fails to reduce the number of unnecessary requests. To improve it, another scheme based on the Genetic

Algorithm (GA) is proposed [5]. The GA-based scheme applies genetic operators, such as selection, crossover and mutation, to a population of strings kept in each processor. Each string, defined as a binary-coded vector $<v_0, v_1, \ldots, v_{n-1}>$ (assuming there are n processors in the system), represents the combination of processors to which a request message from a heavily loaded processor should be sent off. That is, a request message will be sent off to processor P_i if $v_i = 1$, while none will be sent if $v_i = 0$, where $0 \leqq i \leqq n\text{-}1$.

Each string is associated with its payoff values and has its own fitness value. If any of the requests according to a string is accepted, the string is awarded with a positive payoff value inversely proportional to the number of requests sent; if no request is accepted, the payoff value is zero. It is obvious that the payoff will be higher when the requests sent over are less. For example, when at least one request according to a string is accepted, the payoff value of the string can be defined to be $\frac{1}{\sqrt{x}}$, where x is the number of requests sent. The fitness value of a string is an average of the last ζ payoff values, where ζ is predetermined. A string is selected at a probability proportional to its fitness, and requests are sent to processors indicated by the string. For instance, in a system with 8 processors, suppose the selected string <0,1,0,1,-,0,1,0> is in P_4; P_4 will send request messages to, say, P_1, P_3, and P_6. If an *accept* message is sent back from P_3 which is lightly-loaded, P_4 will then migrate the task to P_3. In case two or more *accept* messages are returned, one is selected randomly. After task migration, P_4 calculates the fitness value of the string <0,1,0,1,-,0,1,0> and applies genetic operators to its own population.

With such complicated genetic operations in deciding potential receivers, the GA approach fails to lift the acceptance rate as much as anticipated. Besides, in order to calculate the fitness values and to apply the genetic operators, a request message must be sent off to all processors with corresponding bits in the string being 1's, resulting in more unnecessary requests and redundant acceptances.

For improvement, we propose a simple and efficient new scheme which is a modification of the standard sender-initiated scheme. In our new scheme, a request for task migration will be sent to a processor according to a list of state values kept in each processor, and, by a simple list lookup, the request can be sent to the most fitting processor.

3.2 Our New Scheme

Unlike the standard sender-initiated scheme which selects a potential receiver processor randomly, our proposed new scheme selects it according to a list of state values kept in each processor. The list records the state values of all the other processors. Each state value, initialized to be the allowed task queue length in a processor, is determined by the unused task queue length in the processor.

The list of state values can be used to achieve desirable dynamic load balancing as follows. Let P_x be a processor, x be the state value, w indicate the length (in terms of the execution time) of a task, and m / n respectively refer to the heavy / light load threshold values for the processor. Before a task in the waiting queue of P_x enters the system for execution, go through the *check_load* process first to get x and then compare x with m. If $x > m$ and $x > w$, the task directly enters the task queue of P_x. If $x < m$ (that is, P_x is in heavy load) or $x > m$ but $x < w$ (that is, the task queue is unable to accommodate the task), another processor should be found to assist.

During the *find_processor* process, P_x will operate the *select* procedure by checking through the state values of all the other processors recorded in the list, and send the request (the length of the task to be transferred) to another processor, say P_y. Upon receiving the request, P_y goes through the *check_load* process to get the value of the unused task queue length y, and compare y with n (the light load threshold value). If $y > n$, P_y is in light load and is able to accept the request. It will send y over to P_x. (If P_y receives another request before taking in the task to be transferred from P_x, it has to deduct w from y and takes the updated value (y - w) as its new state value after the operation of *check_load*. Then compare the new state value with n.) After receiving value y, comparing y with n and realizing P_y is able to accommodate the task, P_x will send the task over to the task queue in P_y through the *task_migration* operation, and meanwhile update the state value of P_y into y - w.

On the other hand, if $y < n$, P_y is not in light load and therefore cannot accept the task from P_x. It will send y to P_x which, after comparing y with

n, realizes the fact, changes the state value of P_y in the list into y, and then continues the *select* procedure to find another potential receiver. In this way, P_y simply gives response to P_x with value y whatever the situation is, and P_x gains a chance through such communication to update the state value of P_y. Some simple calculations (*state_adjust* functions) can be employed at the same time to further lift up the acceptance rate: If a request is accepted/rejected by a processor, *state_adjust* functions are applied to the unused queue length of the processor to get the state value so that a processor with bigger/smaller unused queue length gets more proportion of increase/decrease in its state value and hence more/less chance to be selected.

As stated above, our scheme involves only a simple list lookup in determining a receiver processor. Experimental evaluation shows that with slightly higher time complexity over the standard sender-initiated scheme, our new approach realizes very significant elevation in the acceptance rates and, as a result, is able to reduce unnecessary request messages as well as communication overheads. In contrast to the complicated operations the GA-based approach adopts in determining potential receivers, our approach is apparently simpler and more effective.

3.3 Experimental Results

Extensive simulation runs are conducted to collect the results for the standard sender-initiated, the GA-based, and our new schemes. In our simulation model, 16 processors are connected via a network at a communication speed 10K bytes per milliseconds, with incoming tasks arriving at only 12 of the 16 processors for easy observation of the results. Independent tasks come randomly at the same mean arrival rate to each of the 12 processors. Execution times of the tasks and task sizes are exponentially distributed with a mean 100 milliseconds and a mean 10K bytes respectively. The size of a *request*, *accept*, or *reject* message is 1024 bytes. The queue length (in terms of execution time) for a processor indicates its load, and a processor is considered to be heavily loaded when its queue length is over 400 (the threshold T_h = 400) or lightly loaded when less than 200 (the threshold T_l = 200). For the standard sender- initiated scheme and our new scheme, the request limit is 8. For the GA-based scheme, the number of strings in each population is 10; the crossover rate and the mutation rate are 0.04 and 0.05 respectively. If any of the requests according to a string is accepted, the string is awarded with a payoff value equal to the difference between the number of processors and the number of request messages sent, i.e., the number of 1's in the string. (Note that the payoff values calculated in this way result in the best performance for the GA-based scheme, even better than calculated by $\frac{1}{\sqrt{x}}$ [5], where x is the number of requests sent.) If no request is accepted, the payoff value is zero. The fitness value of a string is the average of the last 5 payoff values (i.e., ζ = 5).

The state value in our new scheme is initialized to be the allowed task queue length (= 1000) in a processor and is determined by the unused task queue length in it. If a request is accepted by a processor, *state_adjust* $\frac{\xi^2}{799}$ is applied to the unused queue length ξ of the processor to get the state value. In this way, a processor with bigger ξ gets more proportion of increase in its state value, and hence more chance to be selected. For instance, the processors with ξ = 810, 820 and 850 each get state values = 821, 842 and 904 (the individual increase is 11, 22, and 54). On the other hand, if a request is rejected by a processor, *state_adjust* $\frac{\xi \times 100}{\eta - 100}$ (where η is the queue length) is applied to the ξ of the processor to get the state value, making the processor with smaller ξ get bigger proportion of decrease in its state value and hence less chance to be selected. The simulation continues until 10000 tasks are executed.

Simulation results listed in Tables 4 to 7 are obtained over 10 independent runs and are reasonably accurate. For instance, for the first mean response time value 157, given 95% confidence, the calculated confidence interval half-width over the 10 replications is 2, meaning that we are 95% confident the true result would fall into the interval 157 ± 2, or equivalently, 157 ±1.3%, with less than 2% error.

Table 4. Mean response time for different schemes

arrival rate	0.3	0.4	0.5	0.6	0.7	0.8	0.9
tandard sender-initiated	157	184	215	255	292	334	378
GA-based	157	184	218	255	294	333	375
our new scheme	157	183	214	247	279	311	353

Table 5. Total numbers of messages sent during the simulation period

arrival rate	0.3	0.4	0.5	0.6	0.7	0.8	0.9
standard sender-initiated	202	536	1115	2378	4613	9182	18453
GA-based	1504	3310	7004	12044	18480	28918	40934
our new scheme	192	488	1011	1937	3194	5160	10993

Table 6. Frequency for tasks to enter heavily loaded processors

arrival rate	0.3	0.4	0.5	0.6	0.7	0.8	0.9
standard sender-initiated	90	232	455	872	1438	2228	3221
GA-based	100	229	491	878	1433	2246	3138
our new scheme	92	231	459	846	1306	1918	2745

Table 7. Mean acceptance rates for different schemes

arrival rate	0.3	0.4	0.5	0.6	0.7	0.8	0.9
standard sender-initiated	0.94	0.93	0.9	0.86	0.79	0.7	0.57
GA-based	0.14	0.15	0.15	0.16	0.17	0.16	0.15
our new scheme	0.98	0.98	0.95	0.93	0.9	0.87	0.74

The mean response time listed in Table 4 is the time from the task's arrival till its execution completed. As can be seen, our proposed scheme depicts consistently less mean response time, though the difference is not eminent. It should be observed that when load balancing is applied, each scheme brings about the same degree of balance that makes the queuing time for each task equal and therefore results in almost equal response time. The only difference may come from the communication overhead for load balancing. Since the communication time per *request*, *accept* or *reject* message in our simulation model is assumed to be much smaller than the task execution time and the queuing time, the above difference of response time would become bigger when the overhead per message increases.

Communication overheads are listed in Table 5 with the numbers of total messages (including the *request*, *accept*, and *reject* messages) sent. The number for the GA-based scheme is conspicuously big because once a task enters a heavily loaded processor, it has to send request messages to all of the processors with corresponding bits in the string being 1's . By contrast, the number for our scheme is apparently smaller at any arrival rate, and when the arrival rate grows (that is, when the system load increases), the difference becomes more eminent. For our scheme, the significant reduction in numbers of messages sent is indeed the plain effect of lifted acceptance rates. Table 6 lists the frequency for tasks to enter heavily loaded processors. It shows that tasks are less likely to enter heavily loaded processors for our scheme, esp. when the arrival rate grows higher, helping reduce unnecessary messages and communication overheads. Table 7 gives the mean acceptance rates. The acceptance rate will be 0 if all the request messages sent out are rejected. When a request message is eventually accepted (that is, when task transfer is to happen), the acceptance rate will be the reciprocal of the total number of request messages sent at this attempt, and the mean acceptance rate can be so obtained (average over the number of times listed in Table 6). The acceptance rate for our scheme, as demonstrated, is remarkably higher than the other two schemes, a very significant indicator for performing efficient dynamic load balancing. As a matter of fact, the much improved acceptance rate and the consequent reduced communication overhead for our

proposed scheme are achieved mainly due to our ability to reduce unnecessary request messages.

4. Conclusions

It has been realized that one of the major concerns for a distributed computing system is to find the placements of files and tasks at the sites with minimal total communication overhead. To this end, a Genetic Algorithm (GA) has been developed, but its operations are fairly complicated and time consuming. For improvement, we propose to use the Simulated Annealing (SA) approach. The SA approach is able to avoid the first local minima encountered by maintaining an annealing schedule of the temperatures to keep the searching for the optimal solution from getting stuck. Multiple objective functions are also supplied and properly used to calculate the objective function value of each candidate solution (with the optimal solution aiming to minimize the multiple objective functions), resulting in more practical and favorable solutions. For further evaluation and comparison, we also investigate the Guided Evolutionary Simulated Annealing (GESA) approach, an extension of the SA approach. Experimental results demonstrate that in obtaining desirable file and task placements, our proposed SA approach depicts superior performance with much less complexity over the GA and GESA approaches.

In dealing with the dynamic load balancing problem in a distributed system, the standard sender-initiated scheme and the GA-based scheme are found to be either inefficient or too intricate. To realize more efficient dynamic load balancing, we propose a simple new scheme in which a potential receiver for a task migration request message is selected according to a list of state values kept in each processor. By only a simple lookup at the list, the selection of a most fitting receiver processor for a task migration request message can be achieved. Experimental results show that, with only slightly higher time complexity than the standard sender-initiated scheme, our new scheme is able to yield much enhanced acceptance rates, significantly reducing the number of unnecessary request messages and hence the communication overhead.

References

[1] Corcoran, A. L. and Schoenefeld, D. A., "A Genetic Algorithm for File and Task Placement in a Distributed System," *Proc. 1st IEEE Conf. on Evolutionary Computation*, pp. 340-344 (1994).

[2] Kirkpatrick, S., Gelatt, C. D., Jr. and Vecchi, M. P., "Optimization by Simulated Annealing," *Science*, Vol. 220, pp. 671-680 (1983).

[3] Yip, P., "The Role of Regional Guidance: The Guided Evolutionary Simulated Annealing Approach," Ph.D. Dissertation, Department of Electrical Engineering and Applied Physics, Case Western Reserve University, Cleveland, U.S.A. (1993).

[4] Shivaratri, N. G., Krueger, P. and Singhal, M., "Load Distributing for Locally Distributed Systems," *IEEE Computer*, Vol. 25, pp. 33-44 (1992).

[5] Munetomo, M., Takai Y. and Sato, Y., "A Genetic Approach to Dynamic Load Balancing in a Distributed Computing System," *Proc. 6th Int'l Conf. on Genetic Algorithms*, pp. 418-421 (1994).

[6] Bell, D. A., "Difficult Data Placement Problems," *Computer Journal*, Vol. 27, pp. 315-320, (1984).

[7] Chang, C. C. and Shielke, "On the Complexity of the File Allocation Problem," *Proc. Conf. on Foundations of Data Organization* (1985).

[8] Dowdy, L. W. and Foster, D. V., "Comparative Models of the File Assignment Problem," *ACM Computing Surveys*, Vol. 14, (1982).

[9] Ceri S., Martella, G. and Pelagatti, G., "Optimal File Allocation for a Distributed Database on A Network of Minicomputers," *Proc. ICOD 1 Conf.* (1980).

[10] Holland, J. H., *Adaptation in Natural and Artificial Systems*, Univ. of Michigan Press, Ann Arbor, U.S.A. (1975).

[11] Srinivas, M. and Patnaik, L. M., "Genetic Algorithms: A Survey," *IEEE Computer*, Vol. 27, pp. 17-26 (1994).

[12] Corana, A., Marchesi, M., Martini, C. and Ridella, S., "Minimizing Multimodal Functions of Continuous Variables with the Simulated Annealing Algorithm," *ACM Trans. on Mathematical Software*, Vol. 13, pp. 262-280 (1987).

[13] Richardson, J. T., Palmer, M. R., G. E. Liepens G. E. and Hilliard, M., "Some Guidelines for Genetic Algorithms with

Penalty Functions," *Proc. 3rd Intl. Conf. on Genetic Algorithms* (1989).
[14] Whitley, D. and Kauth, J., "genitor: A Different Genetic Algorithm," *Proc. Rocky Mountain Conf. on Artificial Intelligence*, pp. 118-30 (1988).
[15] Goldberg, D. E., *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Reading, U.S.A. (1989).
[16] Eager, D. L., Lazowska, E. D. and Zahorjan, J., "Adaptive Load Sharing in Homogeneous Distributed Systems," *IEEE Trans. on Software Engineering*, Vol. 12, pp. 662-675 (1986).

Manuscript Received: Jul. 24, 2001
and Accepted: Nov. 6, 2002