Genetic Algorithm for Task Scheduling in Distributed Heterogeneous System

Priyanka G. Gonnade(Kohale)  
Student, Mtech(3rd Sem), Dept. of CSE, 
Priyadarshini Institute of Engineering and Technology, 
Nagpur

Sonali Bodkhe  
Assistant Professor, Dept. of CSE, 
Priyadarshini Institute of Engineering and Technology, 
Nagpur

Abstract: In this paper, a genetic algorithm based approach for task scheduling in distributed system considering dynamic load balancing is discussed. The underlying distributed system has central scheduler and task scheduling is done by this central node. Scheduling of tasks in distributed system involves deciding not only when to execute a process, but also where to execute it. A proper task scheduling will enhance the processor utilization, reduces execution time and increases system throughput. A Genetic algorithm will give the optimal solution for scheduling of task. The task scheduling is centralized and genetic algorithm is applied to central node. This task scheduling policy considers load balancing to prevent the node connected in the system from getting overloaded or become idle ever(if possible).

Keywords-Genetic algorithm, Heterogeneous distributed system, task scheduling, fitness function

I. INTRODUCTION

Distributed heterogeneous computing is being widely applied to a variety of large size computational problems. These computational environments are consists of multiple heterogeneous computing modules, these modules interact with each other to solve the problem. In a Heterogeneous distributed computing system (HDCS), processing loads arrive from many users at random time instants. A proper scheduling policy attempts to assign these loads to available computing nodes so as to complete the processing of all loads in the shortest possible time. There are number of techniques and methodologies for scheduling processes of a distributed system. These are task assignment, load-balancing, load-sharing approaches [4,7,8]. Due to heterogeneity of computing nodes, jobs encounter different execution times on different processors.

In task assignment approach, each process submitted by a user for processing is viewed as a collection of related tasks and these tasks are scheduled to suitable nodes so as to improve performance. In load sharing approach simply attempts to conserve the ability of the system to perform work by assuring that no node is idle while processes wait for being processed. In load balancing approach, processes submitted by the users are distributed among the nodes of the system so as to equalize the workload among the nodes at any point of time. Several methods have been proposed to solve scheduling problem in DCS. The proposed methods can be generally classified into three categories: Graph-theory-based approaches, mathematical models-based methods and heuristic Techniques [3, 5, 8, 10].

Heuristics can obtain suboptimal solution in ordinary situations and optimal solution in particulars. Since the scheduling problem has been known to be NP-complete, using heuristic Techniques can solve this problem more efficiently. Three most well-known heuristics are the iterative improvement algorithms [3], the probabilistic optimization algorithms, and the constructive heuristics. In the probabilistic optimization group, GA-based methods [1,4,5] and simulated annealing [2,12] are considerable which extensively have been proposed in the literature.

One of the crucial aspects of the scheduling problem is load balancing. While recently created processes randomly arrive into the system, some processors may be overloaded heavily while the others are under-loaded or idle. The main objectives of load balancing are to spread load on processors equally, maximizing processors utilization and minimizing total execution time [3,7,8]. In dynamic load balancing, processes must be dynamically allocated to processors in arrival time and obtain a near optimal schedule, therefore the execution of the dynamic load balancing algorithm should not take long to arrive at a decision to make rapid task assignments have proposed scheduling algorithms considering load balancing.

II. SYSTEM MODEL

Heterogeneous distributed computing system

Heterogeneous distributed computing system (HDCS) utilizes a distributed suite of different high-performance machines, interconnected with high-speed links, to perform different computationally intensive applications that have diverse
computational requirements. Distributed computing provides the capability for the utilization of remote computing resources and allows for increased levels of flexibility, reliability, and modularity.

In heterogeneous distributed computing system the computational power of the computing entities are possibly different for each processor as shown in figure 1[1, 3, 4]. A large heterogeneous distributed computing system (HDCS) consists of potentially millions of heterogeneous computing nodes connected by the global Internet. The applicability and strength of HDCS are derived from their ability to meet computing needs to appropriate resources [2, 3, 9]. Resource management sub systems of the HDCS are designated to schedule the execution of the tasks that arrive for the service. HDCS environments are well suited to meet the computational demands of large, diverse groups of tasks. The problem of optimally mapping also defined as matching and scheduling.

We consider a heterogeneous distributed computing system (HDCS) consists of a set of $m \{M_1, M_2, \ldots M_m\}$ independent heterogeneous, uniquely addressable computing entity (computing nodes). Let there are $n$ number of jobs with each job $j$ has a processing time $t_j$ are to be processed in the HDCS with $m$ nodes. Hence the generalized load balancing problem is to assign each job to one of the node $M_i$ so that the loads placed on all machine are as “balanced” as possible [5].

![Figure: 1 Distributed Computing System](image)

Dynamic load distribution algorithms

A dynamic load distribution algorithm must be general, adaptive, stable, fault tolerant and transparent to applications. Load balancing algorithms can be classified as (i) global vs. local, (ii) centralized vs. decentralized, (iii) Non-cooperative vs. cooperative, and (iv) adaptive vs. nonadaptive [7,10].

In this paper we have used centralized load balancing algorithm, a central node collects the load information from the other computing nodes in HDCS. Central node communicates the assimilated information to all individual computing nodes, so that the nodes get updated about the system state. This updated information enables the nodes to decide whether to migrate their process or accept new process for computation. The computing nodes may depend upon the information available with central node for all allocation decision.

Typically, a load distributing algorithm has four components: (i) a transfer policy that determines whether a node is in a suitable state to participate in a task transfer, (ii) a selection policy that determines which task should be transferred, (iii) a location policy that determines to which node a task selected for transfer should be sent, and (iv) an information policy which is responsible for triggering the collection of system state information [1, 3, 7, 11].

Scheduling of tasks in a load balancing distributed system involves deciding not only when to execute a process, but also where to execute it. Accordingly, scheduling in a distributed system is accomplished by two components: the allocator and the scheduler. The allocator decides where a job will execute and the scheduler decides when a job gets its share of the computing resource at the node.

III. GENETIC ALGORITHM

Genetic algorithms (GAs) are search methods based on principles of natural selection and genetics. GAs encode the decision variables of a search problem into finite-length strings of alphabets of certain cardinality. The strings which are candidate solutions to the search problem are referred to as chromosomes, the alphabets are referred to as genes and the values of genes are called alleles. For example, in a problem such as the traveling salesman problem, a chromosome represents a route, and a gene may represent a city. In contrast to traditional optimization techniques, GAs work with coding of parameters, rather than the parameters themselves.

To evolve good solutions and to implement natural selection, we need a measure for distinguishing good solutions from bad solutions. The measure could be an objective function that is a mathematical model or a computer simulation, or it can be a subjective function where humans choose better solutions over worse ones. In essence, the fitness measure must determine a candidate solution’s relative fitness, which will subsequently be used by the GA to guide the evolution of good solutions.
Another important concept of GAs is the notion of population. Unlike traditional search methods, genetic algorithms rely on a population of candidate solutions. The population size, which is usually a user-specified parameter, is one of the important factors affecting the scalability and performance of genetic algorithms. For example, small population sizes might lead to premature convergence and yield substandard solutions. On the other hand, large population sizes lead to unnecessary expenditure of valuable computational time. Once the problem is encoded in a chromosomal manner and a fitness measure for discriminating good solutions from bad ones has been chosen, we can start to evolve solutions to the search problem using the following steps:

1. **Initialization.** The initial population of candidate solutions is usually generated randomly across the search space. However, domain-specific knowledge or other information can be easily incorporated.

2. **Evaluation.** Once the population is initialized or an offspring population is created, the fitness values of the candidate solutions are evaluated.

3. **Selection.** Selection allocates more copies of those solutions with higher fitness values and thus imposes the survival-of-the-fittest mechanism on the candidate solutions. The main idea of selection is to prefer better solutions to worse ones, and many selection procedures have been proposed to accomplish this idea, including roulette-wheel selection, stochastic universal selection, ranking selection and tournament selection.

4. **Recombination.** Recombination combines parts of two or more parental solutions to create new, possibly better solutions (i.e. offspring). There are many ways of accomplishing this and competent performance depends on a properly designed recombination mechanism. The offspring under recombination will not be identical to any particular parent and will instead combine parental traits in a novel manner (Goldberg, 2002).

5. **Mutation.** While recombination operates on two or more parental chromosomes, mutation locally but randomly modifies a solution. Again, there are many variations of mutation, but it usually involves one or more changes being made to an individual’s trait or traits. In other words, mutation performs a random walk in the vicinity of a candidate solution.

6. **Replacement.** The offspring population created by selection, recombination, and mutation replaces the original parental population. Many replacement techniques such as elitist replacement, generation-wise replacement and steady-state replacement methods are used in GAs.

7. Repeat steps 2–6 until a terminating condition is met.

**IV. GA based task scheduling**

In this section, we detail our scheduling algorithm which utilizes GA for task scheduling in HDCS. Genetic algorithms work with a population of the potential solutions of the candidate problem represented in the form of chromosomes. Each chromosome is composed of variables called genes. Each chromosome (genotype) maps to a fitness value (phenotype) on the basis of the objective function of the candidate problem. Jobs arrive at unknown intervals for processing and are placed in the queue of unscheduled tasks from which tasks are assigned to processors. Each task is having a task number and a size. GA is particularly applicable to problems which are large, nonlinear and possibly discrete in nature, features that traditionally add to the degree of complexity of solution. Due to the probabilistic development of the solution, GA does not guarantee optimality even when it may be reached. However, they are likely to be close to the global optimum. This probabilistic nature of the solution is also the reason they are not contained by local optima. The proposed algorithm for task scheduling considering load balancing is presented in figure.

**Algorithm: GA_Loadbalancing**

1. Initialization()
2. Load Checking()
3. Repeat through step 6 until task queue is empty.
4. String Evaluation()
5. Genetic Operation
   a. Mutation()
   b. Reproduction()
   c. Crossover()
6. Request Message Evaluation()
7. End

**Fig 2. GA based task Scheduling**

A fixed number of tasks, each having a task number and a size, is randomly generated and placed in a central task pool from which tasks are assigned to different computing nodes (processors). As load balancing is performed by the centralized
GA-based method, the first thing to do is to initialize a population of possible solutions [7, 8]. This can be achieved using the sliding window technique.

The window size is fixed, with the number of elements in each string equal to the size of the window. Every time when a job arrived at queue of unscheduled tasks (task pool) and placed in corresponding queue. After a interval of time we will apply GA and apply the jobs to the corresponding processors.

If we apply GA at every arrival of task the overhead will be more. So that we applying GA after a random interval of time. Now the jobs in the corresponding queues will be appeared as a two dimensional array, to facilitate the cross over operation the task with size is represented as one dimensional array. The initial population is created by swapping the tasks order randomly for some fixed number of times.

After generating the population we have to perform the selection operation. This operation can be performed by using fitness function as shown below.

\[
\text{Fitness} = \frac{1}{\text{makespan}} \left( \frac{\text{UM}}{m} \right) \left( \frac{\text{acceptable queue size}}{m} \right)
\]

Instead of waiting for the GA to converge, it will be allowed to run for a fixed number of k cycles. The decision was made because solutions generated in less than k generations may not be good enough. On the other hand, running the GA for more than k generations may not be very feasible, as too much time will be devoted to genetic operations. When the GA terminated after k cycles, the fittest string in the pool will be decoded and used as the task schedule.

V. CONCLUSION

Scheduling in distributed operating systems has a significant role in overall system performance and throughput. The scheduling in distributed systems is known as an NP-complete problem even in the best conditions. We have presented and evaluated new GA-Based method to solve this problem. This algorithm considers multi objectives in its solution evaluation and solves the scheduling problem in a way that simultaneously minimizes maxspan and communication cost, and maximizes average processor utilization and load-balance. Most existing approaches tend to focus on one of the objectives.

REFERENCES


