Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, PDPTA 2008, Las Vegas, Nevada, USA, July 14-17, 2008, 2 Volumes. CSREA Press 2008, ISBN 1-60132-084-1

A Genetic Algorithm Based Dynamic Load Balancing Scheme for Heterogeneous Distributed Systems

Bibhudatta Sahoo[1], **Sudipta Mohapatra**[2], **and Sanjay Kumar Jena** [1]
[1]Department of Computer Science & Engineering, NIT Rourkela, Orissa, India
[2] Department of Electronics & Electrical Communication Engineering, IIT Karagpur, India

Abstract - *Load balancing is a crucial issue in parallel and distributed systems to ensure fast processing and optimum utilization of computing resources. Load balancing strategies try to ensure that every processor in the system does almost the same amount of work at any point of time. This paper investigates dynamic load-balancing algorithm for heterogeneous distributed systems where half of the processors have double the speed of the others. Two job classes are considered for the study, the jobs of first class are dedicated to fast processors. While second job classes are generic in the sense they can be allocated to any processor. The performance of the scheduler has been verified under scalability. Some simulation results are presented to show the effectiveness of genetic algorithms for dynamic load balancing.*

Keywords: Heterogeneous distributed system, dynamic load balancing, makespan, genetic algorithm.

1 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.

The *resource manager* schedules the processes in a distributed system to make use of the system resources in such a manner that resource usage, response time, network congestion, and scheduling overhead are optimized. There are number of techniques and methodologies for scheduling processes of a distributed system. These are *task assignment*, *load-balancing*, *load-sharing* approaches [7, 9, 10]. Due to heterogeneity of computing nodes, jobs encounter different execution times on different processors. Therefore, research should address scheduling in heterogeneous environment.

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. Processes might have to be migrated from one machine to another even in the middle of execution to ensure equal workload. Load balancing strategies may be static or dynamic [1, 3, 7].

To improve the utilization of the processors, parallel computations require that processes be distributed to processors in such a way that the computational load is spread among the processors. Dynamic load distribution (also called load balancing, load sharing, or load migration) can be applied to restore balance [7]. In general, load-balancing algorithms can be broadly categorized as centralized or decentralized, dynamic or static, periodic or non-periodic, and those with thresholds or without thresholds [3, 7, 11]. We have used a centralized load-balancing algorithm framework as it imposes fewer overheads on the system than the decentralized algorithm

The load-balancing problem, aim to compute the assignment with smallest possible makespan (i.e. the completion time at the maximum loaded computing node). The load distribution problem is known to be NP-hard [4, 5] in most cases and therefore intractable with number of tasks and/or the computing node exceeds few units. Here, the load balancing is a job scheduling policy which takes a job as a whole and assign it to a computing node [2].This paper considers the problem of finding an optimal solution for load balancing in heterogeneous distributed system. The rest of the paper is organized as follows. The next section discusses Heterogeneous distributed computing system (HDCS) structure and the load-balancing problem. *Section 3* describes the different dynamic load distribution algorithms. We have simulated the behavior of different load balancing algorithm with our simulator developed using Matlab, where each task t_i is with the expected execution time e_{ij} and expected completion time c_{ij}, on machine M_j. The results of the simulation with scalability of computing nodes and tasks are presented in *Section 4.* Finally, conclusions and directions for future research are discussed in *Section 5.*

2 System and problem model

2.1 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.

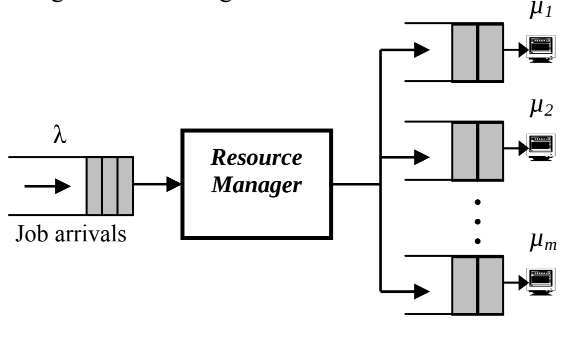


Figure: 1 Distributed Computing System

We consider a heterogeneous distributed computing system (HDCS) consists of a set of m $\{M_1, M_2, \dots Mm\}$ 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 Mi so that the loads placed on all machine are as "balanced" as possible [5].

2.2 Mathematical model for load balancing

This section presents a mathematical model for load balancing problem based on *minmax* criterion. Objective of this formulation is to minimize the load at the maximum loaded processor. Let A(i) be the set of jobs assigned to machine M_i; hence the machine M_i needs total computing time $T_i = \sum_{j \in A(i)} t_j$, which is otherwise known as (L_i) load on machine M_i. The basic objective of load balancing is to minimize *makespan*[11]; which is defined as maximum loads on any machine ($T = max_i \ T_i$). This problem can be expressed as linear programming problem, with the objective to Minimize L (load of the corresponding assignment)

Minimize L

$$\sum_i x_{ij} = t_j \text{, for all } j \in A(i)$$

$$\sum_j x_{ij} \leq L, \text{ for all } i \in M$$

$x_{ij} \in \{0, t_j\}$

$$x_{ij} = \{0, t_j\}, \qquad \text{for all } j \in A(i),\ i \in M_j$$

$$x_{ij} = 0, \quad \text{for all } j \in A(i),\ i \notin M_j$$

Where $M_j \subseteq M$; set of machines to which the job j can be assigned.

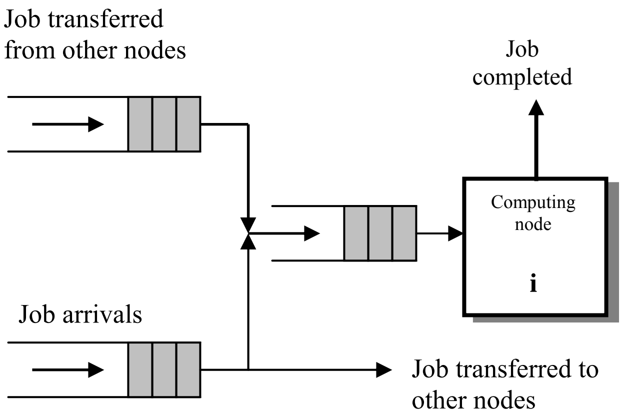
The problem of finding an assignment of minimum makespan is NP-hard [5]. The solutions to this can be obtained using a dynamic programming algorithm $O(n L^m)$, where L is the minimum makespan.

Due to the complexity of load balancing problem, most of researchers proposed heuristic algorithms, while optimal algorithm are developed for only restricted cases or for small problems[4]. Genetic algorithms (GAs) are evolutionary optimization approaches which are an alternative to traditional optimization methods. GA is most appropriate for complex non-linear models where location of the global optimum is a difficult task. Hence genetic algorithms have been used to solve hard optimization problem. In this paper we have analyze the performance HDCS where half the total processors have double speed than others.

3 System Model and Methodology

3.1 System and Workload Models

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, 13]. When a new job arrives at the node (Figure 3.1) the transfer policy looks at the node's job queue length. The job is allowed to execute at the node if the job queue length is less than a predetermined threshold. Otherwise Job is assigned to the central scheduler.



Job transferred from other nodes
Job completed
Computing node
i
Job arrivals
Job transferred to other nodes

Figure: 3.1 Job flow at computing node

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. In this paper we have used the computing resource model as discussed in [6]

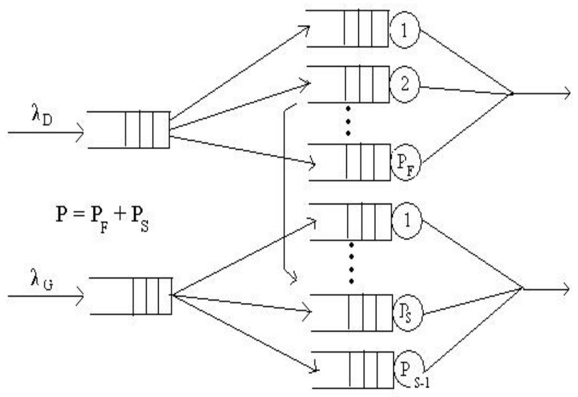


Figure 3.2: Central scheduler Queuing Model

Each heterogeneous computing node is multitasking, can accommodate maximum K no of jobs for some acceptable QoS. The heterogeneous distributed computing system addressed here can be expressed by Kendall notation[14] like ***M/M/m/K/n***, where: (i) First M: represents exponential inter arrival times between jobs(tasks) distribution (Poisson process), (ii) Second *M*: represents exponential execution time of jobs distribution, (iii) *m*: represents number of heterogeneous computing nodes,(iv) *K*: represents maximum number of tasks that can be in a computing Node under the multitasking, and (v) *n*: represents number of jobs . Let λi be the arrival rate of jobs at computing node *i* , Hence the arrival rate at resource manager is λ, where $\lambda = (\lambda_1 + \lambda_2 + \lambda_3 + \Lambda + \lambda_m)/m$ We have assume that the service rate of all m heterogeneous computing nodes are different, i.e. $\mu_j \neq \mu_i$ for any two computing node.

In this paper we have use a heterogeneous distributed computing system, with two different type of computing nodes connected via a high-speed network as shown in figure 1. Half of the computing nodes (nodes) execute at double the speed of the others. The jobs assigned for the execution are assumed to be highly independent. That means when a job is scheduled for execution, no job ever ideally waits for communication with any other jobs. This system can be modeled as an open queuing network [1,6]. Let M_F and Ms be the number of fast or slow computing nodes (machine), so that M_F = Ms = m/2. We have assumed that the jobs are classified into types as *dedicated* and *generic* jobs with inter-arrival time λ_G, and λ_D respectively. The jobs of first class are dedicated to fast processors and second class jobs are generic in the sense that can be allocated to any processor. There is one arrival stream for dedicated jobs and one for generic jobs. Model of the system is shown in figure 3.2. The generic jobs arrive at a rate λ_G, and can process by any of the computing node. We shall assume that all arrival streams are Poisson process. All jobs have identically distributed service requirements. One allocated to a particular computing node, a job can not be reassigned and must be process to completion by that node. The dedicated jobs are mostly the local loads of the computing nodes; if a computing node is loaded above a threshold it is not available for generics jobs for a period of time.

3.2 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. non-adaptive[7,13]. 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 deicide 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.

The scheduling policies can be probabilistic, deterministic and adaptive. In probabilistic case, the dedicated jobs are dispatched randomly to the first processor with equal probability while the generic jobs are randomly dispatched to the slow processors.

In deterministic case the routing decision is based on system state. Two different policies are examined for this case. In both policies, the dedicated jobs join the shortest of the fast processor queues. However, the first policy requires the generic jobs join the shortest queue of the slow processors while the second policy assigns generic jobs to the (slow or fast) processor expected to offer the least job response time. However, when a generic job is assigned to a fast processor, job start time depends on an aging factor. In adaptive case, job migration from slow to fast processors employed. This is a *receiver-initiated* load sharing method employed to improve the performance of generic jobs. The policy is initiated when a generic job is queued on a slow processor and a fast processor becomes idle. Only the migration of non-executing jobs is considered. Executing jobs are not eligible for transfer because of complexity issues. When a job is transferred from a slow to fast processor for remote processing, the job incurs additional communication cost. Only jobs that are waiting in the queues are transferred. The benefit of migration depends on migration cost[6,13].

We have referred the workload model that is characterized by three parameters:

- The distribution of job arrival
- The distribution of processor service time
- The distribution of the migration overhead.

3.3 Job Scheduling Policies

Here we examined only the non-preemptive scheduling policies only with a assumption that the scheduler has perfect information on *(i) The length of all processor queue,* qnd (ii) *The queuing time of dedicated jobs in the fast processor queues*. We have used the scheduling strategy used by Karatza *et al.*[6]. The scheduling strategies used for load balancing decision are

- Least expected response time for generic jobs maximum wait for dedicated jobs (LERT-MW)
- LERT-MW with migration having idea about execution times

LERT-MW: In this policy also dedicated jobs are dispatched to the fast processor which is having the least queue length, and generic job will sent to either fast or slow processor expected to offer the least job response time. The minimum job response time (makespan) is based on the user's view of how to improve performance. This algorithm needs global information on queue lengths for the generic and dedicated jobs, and also it requires additional information about the time-dedicated jobs waiting in a queue.

LERT-MWM: In the above method we don't have priori knowledge about the execution times. So, we can't evenly distribute the load among all the nodes. The results some processors remain idle, while others are overloaded. This requires the migration of jobs form overloaded processors to idle processors. By this process migration overhead may be more for small jobs & results lower processors utilization. So we are going for GA, which will use the LERT-MW in the phase of scheduling.

3.4 GA based Load Balancing Method

In this section, we detail our scheduling algorithm which utilizes GA for load balancing 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. The algorithm we have developed us based upon one developed by Zomalya *et al.*[11, 12]. 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 follows the concept of solution evolution by stochastically developing generations of solution populations using a given fitness statistic. They are particularly applicable to problems which are large, non-linear 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 do 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 load balancing is presented in figure 3.3.

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 [11, 12]. 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.

As load-balancing is performed by the centralized GA-based method, the first thing to do is to initialize a population of possible solutions. Every time when a job arrived at queue of unscheduled tasks (task pool), the job is scheduled by using LERT-MW method 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. Here we are generating 6 populations for our problem. After generating the population we have to perform the selection operation. This operation can be performed by using fitness function..

```
ALGORITHM: GA_Loadbalancing
[1]   Initialization()
[2]   Load cheking()
[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
```

Figure 3.3: Genetic algorithm framework for load balancing

An objective function is the most important component of any optimization method, including a GA, as it is used to evaluate the quality of the solutions. The objective function here is to arrive at task assignments that will achieve minimum execution time, maximum processor utilization, and a well-balanced load across all processors. Then, the objective function is incorporated into the fitness function of the GA. This fitness function will then be used to measure the performance of the strings in relation to the objectives of the algorithm.

The first objective function for the proposed algorithm is the *makespan* as described in section 2.2. Considering the fact that a computing node M_i may not always be idle, The total task completion time can be expressed as sum of current load of M_i (CL_i) and new load of M_i (NL_i).

$$T_i = CL_i + NL_i$$

For simplicity the computing nodes are referred as single processor, however a single node may have more than one processor as dedicated computing unit. We have use average node (processor) utilization as one of metric to study the performance of load balancing algorithm. As high average processor utilization implies that the load is well balanced across all nodes(processors). By keeping the processors highly utilized, the total execution time should be reduced. The expected utilization of each processor based on the given task assignment must be calculated. This is achieved by dividing the task completion times of each processor by the *makespan* value. The utilization of the individual processors (UM_i) can be given by:

$$UM_i = T_i / makespan$$

The overall task assignment being evaluated may have a small *makespan* and high average processor utilization.. However, assigning these tasks to the processors may still overload some of the processors. Therefore, the third objective is to optimize the number of acceptable node queues. Each node queue is checked individually to see if assigning all the tasks on the node queues will overload or under-load the processors. Whether a processor queue is acceptable or not is determined by the light and heavy thresholds used [12].

Low Threshold: Average Load * 0.8

High Threshold: Average Load * 1.2

To facilitate the design of genetic algorithm for load balancing, the three objectives discussed above are incorporated into a single fitness function and given by the following equation:

$$Fitness = \frac{1}{makespan} \times \left(UM_i / m \right) \times \left(\frac{acceptable\_queue\_size}{m} \right)$$

The fitness function is used to evaluate the quality of the task assignments using string_evaluation() as shown in figure 3.3.

Instead of waiting for the GA to converge, it will be allowed to run for a fixed number of k cycles (k=10 in this paper). 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 is terminated after k cycles, the fittest string in the pool will be decoded and used as the task schedule. We have analyzed the centralized dynamic load-balncing mechanism using a discrete event simulator developed by us using Matlab 6.0.

4 Performance analysis

The following results summarize the overall model performance. Here we are simulating the model by using the metrics like throughput, number tasks waiting in the queue with in interval. We have used the *M/M/m/K/n* queuing model for the simulation. From the results in figure: 4.1, it concludes that the LERT-MWM with execution times method is best when compared to LERT-MW which is not having the priori information about the execution time of the jobs. So if we know the execution times of all the jobs we can effectively distribute the load that can be showed in figure 4.1.

The next experiment compares the LERT-MWM and Genetic Algorithm using the LERT-MW method in scheduling phase. These comparisons are shown in the below Figures 4.2 and 4.3. The test runs were based on a set of default values: number of iterations: 500, number of processors: 50, number of generation cycles: 3, population size: 6, maximum size of each task: 100, High Threshold multiplier: 1.2, and Low Threshold multiplier: 0.8. The performance comparisons were done in two types.

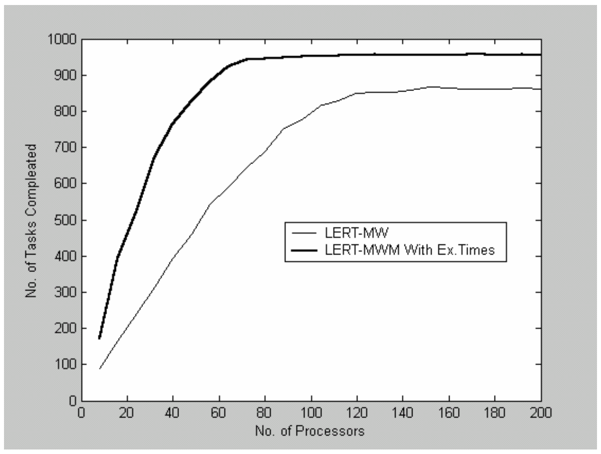


Figure 4.1: Comparison of LERT-MWM and LERT-MW with and with out knowing the execution times.

4.1 Changing the Number of Tasks:

Default values were used for all the parameters except for the number of tasks to be assigned. The number of tasks was varied from 500-2000 and the effects on the total completion time and throughput are given below.

The Figure: 4.2 show that the total time taken for all three algorithms increased linearly as the number of tasks was increased. It was also noted that the GA performed better among the three algorithms. When comparing the results of the GA and the LERT-MWM algorithm, one can observe that the gap between these two curves was widening as the number of tasks was increased. This shows that the GA actually reduced the total completion time by a considerable amount (greater speedup) in comparison to the LERT-MWM algorithm as the number of tasks increased. This also indicates reliable performance of the GA_loadbalancing when the number of tasks increases. Again we compared our GA with another GA technique using normal scheduling, means assigning jobs sequentially(First Come First Serve) to the processors one by one. For all the cases the proposed GA shows better performance.

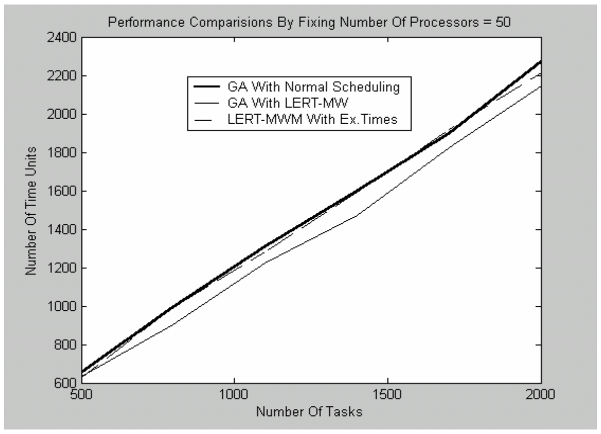


Figure 4.2: Comparison of GA, With Normal Scheduling and LART-MWM by Fixing the Number of Processors.

4.2 Changing the Number of processors:

Here we have studied the performance of load balancing algorithms against the scalability of computing nodes (processors). In simulation the number of processors was varied from 10-160 and the effects on the total completion time and throughput are shown in figure 4.3.

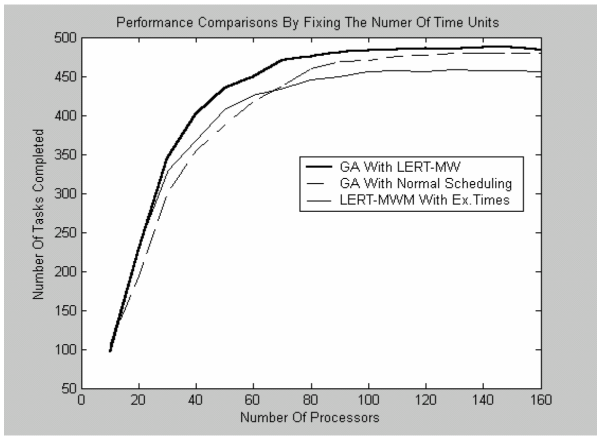


Figure 4.3: Comparison of GA, GA with Normal Scheduling and LART-MWM by Fixing the Number of Time Units

The total jobs completed with in an interval by varying the total number of processors increased linearly first and after that it stabilizes at some point. In most cases, the GA out performed the other two algorithms in terms of processor utilization. Hence if we know the execution times of the jobs we can effectively balance the loads among all the nodes.

5 Conclusions

This paper studies performance of genetic algorithm based approach to solve dynamic load balancing in heterogeneous computing system. Simulation results indicate that the performance of best method depends on system load. We analyzed the system performance and scalability of computing nodes with load balancing. The simulation result shows GA based algorithm works better when the numbers of tasks are large. As distributed systems continue to grow in scale, in heterogeneity, and in diverse networking technology, they are presenting challenges that need to be addressed to meet the increasing demands of better performance and services for various distributed application.

6 References

[1] Sivarama P. Dandamudi, Sensitivity evaluation of dynamic load sharing in distributed systems, *IEEE Concurrency,6*(3), 1998, 62-72.

[2] Jie Li, & Hisao Kameda, Load balancing problems for multiclass jobs in distributed/parallel computer systems, *IEEE Transactions on Computers, 47*(3), 1998, 322-332.

[3] Veeravalli Bharadwaj, Debasish Ghose, Venkataraman Mani, & Thomas G. Robertazzi, *Scheduling Divisible Loads in Parallel and Distributed Systems* (Wiley-IEEE Computer Society Press, 1996).

[4] Gamal Attiya & Yskandar Hamam, Two phase algorithm for load balancing in heterogeneous distributed systems, *Proc. 12th IEEE EUROMICRO conference on Parallel, Distributed and Network-based processing,* Coruna, Spain 2004, 434-439.

[5] Jon Kleinberg & Eva Tardos, *Algorithm Design* (Pearson Education Inc. 2006).

[6] Helen D. Karatza, & Ralph C. Hilzer, Load sharing in heterogeneous distributed systems, *Proceedings of the Winter Simulation Conference*, *1*, San Diego California, 2002 Page(s): 2002, 489 – 496.

[7] Jie Wu, *Distributed system design*,(CRC press, 1999)

[8] Y.Zhang, H.Kameda & S.L.Hung, Comparison of dynamic and static load-balancing strategies in heterogeneous distributed systems, *IEE proceedings in Computer and Digital Techniques,144*(2), 1997, 100-106.

[9] Bora Ucar, Cevdet Aykanat, Kamer Kaya, & Murat Ikinci, Task assignment in heterogeneous computing system, *Journal of parallel and Distributed Computing*, *66*, 2006, 32-46.

[10] Marta Beltran, Antonio Guzman, & Jose Luis Bosque, Dealing with heterogeneity in load balancing algorithm, *Proc. 5th IEEE International Symposium on Parallel and Distributed Computing,* Timisoara, Romania**,** 2006, 123-132.

[11] A. Y. Zomaya, C. Ward, & B. Macey, Genetic Scheduling for Parallel Processor Systems: Comparative Studies and Performance Issues, *IEEE Transaction Parallel and Distributed Systems*, *10*(8), 1999, 795-812.

[12] A. Y. Zomaya, & Y. H. Teh, Observations on using genetic algorithms for dynamic load-balancing, *IEEE Transactions on Parallel and Distributed Systems, 12*(9), 2001, 899-911.

[13] B. A. Shirazi, A. R. Hurson, & K. M. Kavi, Scheduling and load balancing in parallel and distributed systems, CS press, 1995.

[14] K. S. Trivedi, Probability and statistics with reliability, queuing and computer science applications, Prentice Hall of India, 2001.