Indian Journal of Science and Technology, *Vol 9(31), DOI: 10.17485/ijst/2016/v9i31/89615, August 2016*

ISSN (Print) : 0974-6846
ISSN (Online) : 0974-5645

Task allocation in distributed systems

P. Neelakantan* and S. Sreekanth

Department of CSE, SITAMS, Chittoor - 517127, Andhra Pradesh, India;
pneelakantan@rediffmail.com, pranavsree_2000@rediffmail.com

Abstract

Objectives: In this paper, the response time of the tasks is minimized by migrating the tasks from the overloaded computers to the under load computers using a load balancing technique. **Methods/Statistical analysis:** Load balancing algorithms belong to the nearest neighbor technique considers only the neighbor computers for migrating the tasks to reduce communication cost between the computers. Though several nearest neighbor techniques are available, this paper uses the diffusion technique for balancing the tasks between the computers in a distributed system, as the strength of the diffusion technique lies in all port communication model and asynchronous implementation. **Findings:** To evaluate the performance of the system, the load on the system is varied with the mean inter arrival time of the tasks at each computer. The simulation has been carried out to observe the performance of the proposed algorithm (Dynamic Distributed Diffusion) with the existing algorithms SID, GDE, AN by considering tasks arrive to a computer with a Poisson process and follow n* M/D/1 Queuing model. The simulations show the proposed algorithm reduced the load balancing time, compared to the existing algorithms in the literature. **Application/Improvements:** The proposed algorithm can be suited to any topology and it reduces the load balancing time and hence the response time of the tasks is minimized.

Keywords: Diffusion, Load Balancing, Migration, Nearest Neighbor, Response Time, Task Allocation

1. Introduction

Load-balancing in a distributed system can be done statically before the execution of any task, or during the runtime of the task. Static load balancing[1,2] uses the average behavior of the system in making the load decisions, and their principal advantage is the lower overhead cost needed to execute them. Static load balancing policies does not adapt their decision to fluctuations in workload. In contrast, Dynamic load-balancing[2,3] attempts to dynamically balance the loads reflecting the current system state and that are thought to be able to further improve the system performance. Dynamic load balancing is suited for the applications that have uncertain computational requirements or unpredictable communication patterns. Adaptive calculations, circuit simulations and VLSI design, N-body problems, parallel discrete event simulation, and data mining are just a few of those applications.

Dynamic Load-Balancing (DLB)[4] redistributes the load among the computers during run time, so that each computer contains the same or nearly the same amount of work load. In dynamic load balancing the tasks are transferred from the heavily loaded computers to the lightly loaded computers over a dedicated network with the aim of reducing the response time of the task.

In Literature, many authors proposed iterative load balancing algorithms considering the loads as real numbers and those loads can be split arbitrarily. However, medium and coarse grain applications, the tasks are not infinitely divisible and hence they are treated as integer numbers. The load balancing algorithm designed for discrete load model cannot produce a global load balance even after the after the termination of the load balancing algorithm[5].

The issue of when to invoke a load balancing operation, which computer makes load balancing decision, how to collect information and migration of the load between the computers are the issues to be resolved in Dynamic load balancing strategy[6].

An allocation of tasks that causes the computers to contain approximately an equal amount of workload will

**Author for correspondence*

increase the efficiency of a computation. Increasing the overall efficiency will typically reduce the response time of task that is the ultimate, practical goal. The tasks are to be executed in parallel for faster processing.

When task times known in advance, tasks are allocated to the computers in a system before computation. However, there are many applications whose time is not known in advance. The service time of tasks will change over time and tasks with unpredictable behavior will be allocated to the computers at run time. In static load balancing, tasks are allocated to the computers in a system during compile time, while in dynamic load balancing policies, the tasks are allocated to the computers at run time.

A practical solution of the dynamic load-balancing problem involves four distinct phases[4] .In this approach, a computer involved in the load balancing process will perform the load balancing by considering the following four aspects.

- **Load Evaluation:** To find an imbalance in the system, a load index of a computer will serve as a basis. The load at every computer computed in determining load imbalance. Computers containing low load index value said to be lightly loaded computers and computers with higher load index value said to be overloaded computer. To make the load index same in all the computers few tasks from the overloaded computer transferred to the lightly loaded computer to reduce the imbalance in the system.
- **Load balancing profitability determinatio**n: When a load imbalance exists in the system, the load balancing algorithm transfers tasks from overloaded computers to lightly loaded computers. While transferring tasks to the remote computers, communication delay & migration cost has taken into account to determine the profitability of load balancing.
- **Task migration strategy**: In task migration strategy, the load balancing algorithm chooses the overloaded computer to remove tasks from its queue and these tasks are transferred to the lightly loaded computer queue. The source and destination computers are chosen in such way that minimizes the communication overhead incurred during the load balancing process.
- **Task selection strategy:** The source computer must take care in selecting the tasks for transferring. The tasks selected should meet the objective of load balancing.

In this paper, the proposed algorithms focus on second and third aspects. The second phase involves a load balancing decision that uses two different approaches on the type of information they use. In a centralized approach[7,8] the information is collected from all the computers in a system, so the load balancing will be accurate. However, they impose more communication overhead, because the computers in the distributed systems are widely dispersed and not scalable when the number of computers in the system increases. On the other hand distributed approaches produce less overhead in taking the load balancing decision because of involvement of few computers.

The load index of each computer at regular intervals of time and it is known as the load evaluation instant. The load information collected from the computer should balance the load gathering frequency and the ageing of load information. The use of obsolete values by the load balancing algorithm can be avoided by implementing the following three load information collection rules:

- On-demand Load information: Computers collect load information from each other whenever a load-balancing operation is about to begin or be initiated[9].
- Periodical Load information: The load information is exchanged among the computers periodically. This information may be useful or not useful to the other computers in the system[10]
- On-state-change-driven Load information: when utilization of computers increases by a certain degree, they inform their load to the other computers in the system[11].

Collection of load information from the computers in a system by using an on-demand method minimizes the communication overhead incurred but delays the load balancing operation, which could result in a increase of processing time of tasks. Bidding methods use on-demand load collection policy, where lightly loaded computer's request the load information from the other computers in a system and choose the best computer for doing load balancing[9].

Conversely, the periodic method[10,11] allows computers to initiate the load balancing operation immediately based on the load information of the computers. The setting of interval for information gathering from the computers is the main issue in the periodical method. Heavy communication overhead is incurred when a short interval is set for load balancing operation, while the accuracy of the load information used in the load balancing operation is nullified when the length of the interval for the load information from the computers is set high.

The on-state-change-driven load information is the combination of on-demand load information and periodic load information schemes. In this case, the neighbors will receive a status message from the computer, when its load is changed by certain degree or if the updated interval is elapsed since the last update. This mechanism reduces the frequent updates and hence the communication cost is also reduced. The load index of a computer serves as load information which is used to know the state of the computer. The zero or a low value of the load index indicates that the computer is lightly loaded and large load index indicates that the computer is overloaded. The overloaded computer is not considered in allocating the tasks and load balancing algorithm must remove some of the tasks from the overloaded computer to reduce its load index. The load index is a non-negative value and value of load index increases over time as the tasks are added to the computer.

In the design of load balancing algorithms, the utilization of idle computers must be increased to improve the performance of the system. The calculation of load indices of a particular computer must impose less overhead in a system and it should not interfere with the performance of the system. While calculating the load index of a computer in a system, the following aspects must be considered as it serves as a basis for improving the system performance:

- The load index measure of a computer must include processing time on the task, memory and I/O requirements.
- Future loads will have more impact on the performance on the system than the current load. So, future loads must be predicted to reduce the side effects of load balancing.
- Allow for a simple relation to the load index, so that these values are easi*ly translatable;*
- Adopt stable indices, rejecting variable values.

The reason behind the wide range of load indices in literature is there is no agreement about the efficiency and effectiveness of the single load index. Different load indices have been computed depending on various parameters such as CPU queue length, the average CPU queue length over a period, CPU utilization, response time and the amount of main memory available.

The load indices can be divided into two groups: Specific indices and generic indices. Specific indices allow obtaining a single value to represent the load index of a computer, while generic indices contain two or more values, which require to be averaged. Generic indices used when the behavior of the application is not known[12]. Even then generic load indices have not shown any performance improvement over the single load index value, instead it poses more overload on the system which affects the performance of the system. In general the most widely accepted load index in literature is the usage of queue length and execution /response time.

The formula proposed[8] for obtaining the load index of a computer is a linear combination of the task execution time s_j over the resource r_j and resource r_j contains a queue length q_j which is given by:

$$L_i = \sum_{j=1}^{N} S_j\, q_j \tag{1}$$

Where N is the total number of resources having queues in a computer. Fontlupt et al.[9] proposed the load index of a computer as the number of items residing in the queue of a computer. The total load of the system is given by

$$W = \sum_{i=1}^{p-1} W[i] \tag{2}$$

1.1 Load imbalance detection

For load balancing to be useful, one must first determine when to load balance. Load balancing among the computers comprises detecting the load imbalance and determining the cost of load balancing. The load imbalance detected in a synchronous manner by comparing the average load of the system with loads of every computer in the system or in an asynchronous fashion, when the load of a computer exceeds certain threshold.

1.2 Work Transfer Vector Calculation

The imbalance among the computers in the distributed system is reduced by an appropriate transfer strategy. The transfer strategy includes choosing the sender and the destination computer pair and the amount of the load to be transferred between the sender computer to the destination computer. The destination computer shall be chosen (i) Randomly (ii) Fixed (iii) Evaluated

1.3 Task selection

Once the amount of the load to be moved between the computers has been calculated, tasks to be selected will meet the amount of the load to be transferred between the computers. The selection of the tasks will have an impact on the quality of load balancing. Tasks can be transferred in a single direction between two computers or in mutual direction. The problem of selecting which tasks to move is weakly NP-complete, since it is simply the sum of subset problem. Fortunately, approximation algorithms exist which allow the subset sum problem to be solved to a specified nonzero accuracy in polynomial time[6,13,14].

Another factor that influences the task transfer between the computers includes communication delay. In general, a cost is associated with each task transfer and a task with a lowest cost considered for migration.

1.4 Task Migration

In addition to selecting which tasks to move, a load balancing framework must also provide mechanisms for actually moving those tasks from one computer to another. Task movement must preserve the integrity of a task's state and any pending communication.

1.5 System level

In this level, a group of computers is involved in the load balancing process and the decisions taken at this level will affect a group of computers. There are three approaches depending on the set of computers that participate in the load balancing process.

1.5.1 Centralized

In Centralized policies[14,] only one computer acts as load balancing decision maker called a master computer. Tasks arriving from the external world are sent to the master computer which allocates tasks to the computers in the system to do load balancing. The master computer collects the information from all the computers in the system to do load balancing. The allocation will be fair and results in best allocation strategies. The centralized policies employ a client-server model in distributed systems.

Because there is only one computer is involved in the load balancing decision, the number of messages between the computers in the system is minimized and fair allocation of tasks to the computers is guaranteed. The major disadvantage of the centralized load balancing policy is a single point of failure and heavily loads on master computer.

1.5.2 Totally distributed

In a distributed scheme[15], load balancing decision involves all the computers in the system. Collecting information from all the computers in the system to do load balancing will pose high communication overhead, which forfeits the purpose of load balancing. To put the communication overhead minimal, load-balancing operation is restricted to the domains where the domain of the given computer is the set of computers directly connected to it. The overlapping domains and non overlapping domains can be distinguished by observing the relationship between different domains.

The nearest neighbor approach is referred to be as the domain containing a computer and a direct communication link with the other computers. Nearest-neighbor load-balancing methods operate to reduce load imbalance between a computer and its neighboring computers with the aim of diffusing load among the computers converging toward a system-wide balance.

Otherwise, load-balancing strategies categorized as non-nearest-neighbor approaches. In Non-nearest neighbor approach, computers collect load information from the computers present in entire system, which are not restricted to its domain. The scope of the domain extends to a large radius that may also include the neighbors' neighbors and so on. Load balancing done by using totally distributed approach offer high reliability when compared to the centralized approach, but they incur more communication overhead compared to the centralized approach. In this paper, the proposed algorithms for load balancing using the nearest neighbor approach.

1.5.3 Partially distributed

When a system contains more than hundreds of computers, neither centralized nor distributed approaches proved inappropriate. Partially distributed strategies (also called semi-distributed) are proposed as a trade-off between centralized and fully distributed mechanisms.

The main idea is to divide a distributed system into domains and allocate tasks to these domains[13]. These strategies can be viewed at two levels: (i) load balancing within a domain and (ii) load-balancing among all the domains. Each level of the strategy contains different solutions. Each domain is usually managed by a single master-computer using a centralized strategy and, or every computer is involved in taking the load balancing decision, master-computers may (or may not) exchange aggregated information about their corresponding clusters[9].

1.5.4 Synchronous versus Asynchronous

The load balancing operations carried out by all the computers at the same instant of time and the computers could not proceed with normal computation referred as synchronous strategy. If the computer carries load-balancing operations regardless of what other computers are doing referred as asynchronous strategy[16]

2. Proposed Method

Let $\mathbf{V} = \{1, 2, \ldots \mathbf{N}\}$ is a set of computers and E is a set of edges connecting computers in a distributed system with N computers represented as an undirected graph G= $\{\mathbf{V}, \mathbf{E}\}$. The set of computers that are connected directly with computer i are called neighbor computers and is represented by $\mathbf{D_i} = \{\mathbf{j} | \mathbf{j} \in \mathbf{V} \text{ and } (\mathbf{i}, \mathbf{j}) \in \mathbf{E}\}$.

2.1 Notations

"N: Number of computers in a distributed system

V: set of computers in a distributed system N=|V|

$\mathbf{L_i(t)}$: Computer i load at time t $\forall$ i

$\mathbf{D_i}$: Computer i domain at time t = $\{\mathbf{j} | \mathbf{j} \in \mathbf{V} \text{ and } (\mathbf{i}, \mathbf{j}) \in \mathbf{E}\}$

$\mathbf{L_{j \in D_i}(t)}$: Computer $\mathbf{j}$ load that belongs to domain of computer $\mathbf{i}$ at time t

$\bar{\mathbf{L}}_\mathbf{i}(\mathbf{t})$: Computer i domain average load given by

$$\bar{\mathbf{L}}_\mathbf{i}(\mathbf{t}) = \frac{\mathbf{L_i(t)} + \sum_{\mathbf{j \in D_i}} \mathbf{L_j(t)}}{|\mathbf{D_i}| + 1}$$

$\mu(\mathbf{t})$: System average load $\frac{1}{N}\sum_{i=1}^{N} L_i(t)$ at time t

$\mathbf{Var(t)}$: System variance, $\sigma^2(t) = \frac{\sum_{i=1}^{N}[(L_i(t) - \mu)]^2}{N}$

$\mathbf{dev_{ij}(t)}$:Computer j deficit load at time t , $\forall$ j$\in$ $\mathbf{D_i}$

$\mathbf{deviation_i(t)}$: Computer i excess load at time t

$\mathbf{L_i^{max(t)}}$: Computer i domain maximum load

$\mathbf{L_i^{min(t)}}$: Computer i domain minimum load

$\mathbf{x_{ij}(t)}$: Apportion of the load sent to the deficit neighbor j in domain of computer i

$\varphi_\mathbf{i}(\mathbf{t})$: load distribution done by computer i

Assumptions

"It is assumed that a distributed system consists of identical high performance computers (homogeneous system) connected by a set of high bandwidth communication links. It has been assumed a distributed system consists of independent parallel jobs where they can be run at any time, in any order and at any computer. The arrival of the jobs is a Poisson distribution."[17]

It is assumed that the load at a particular computer is the sum of jobs that are allocated to that computer. Let *A (i)* be the set of jobs assigned to computer i, then the load at computer i is $L_i = \sum_{k \in A(i)} t_k$, where t_k is the job *k*. In real world applications, the load cannot be arbitrarily divided, but only to some extent. So a discrete model for the load is needed.

2.2 Mathematical Model

"Let $\mathbf{L_i(t)}$ is the load value of computer $\mathbf{i}$ in a system of N computers at a given time instant t and the load values among the computers in the system are represented by using the load vector $\mathbf{L(t)} = (\mathbf{L_1(t) \ldots L_N(t)})$. The load values of a computer can be a real or non negative integer values depending on the granularity of the application. Let the load value of every computer in a system is represented as an integer quantity. When the load is represented as an integer quantity the maximum load difference in the system between any two computers is to be 0 or 1.

Let the load balancing algorithm is initiated at time t and it takes some time t_1 where t_1>t to balance the load among the computers. Let the average system load at time t is given by

$$\bar{L} = \mu(t) = \frac{1}{N}\sum_{i=1}^{N} L_i(t) \quad (3)$$

Since load balancing algorithms are the load conservative, i.e., they do not neither create nor destroy load, but only move it around the system such that load values of individual computer changes due to load balancing actions .In static load situations the value of μ does not change over time. Thus

$$\mu(\mathbf{t}) = \mu(\mathbf{0}) = \mu \quad \forall \mathbf{t}$$

The imbalance of the system load at the time t is measured with a synthetic indicator, the variance of the load of the computers, i.e., their quadratic deviation from μ .

$$\sigma^2(\mathbf{t}) = \frac{\sum_{\mathbf{i=1}}^{\mathbf{N}}(\mathbf{L_i(t)} - \mu)^2}{\mathbf{N}} \quad (4)$$

If variance among the system is minimized, each computer in the system would contain equal loads to process so as to minimize the response time of process or job."[17]

2.3 Description of the Model

"In a distributed system, the computers exchange their load information at periodic intervals of time called information exchange interval T_s. The information exchange consists of load information of a computer and at the instant this information exchange takes place is called an information exchange epoch. In order to reduce the communication overhead, the information exchange is restricted only to the neighboring computers.

Each computer i receives a load information message from its neighbors, which is kept in the computer i memory. Due to communication delays induced by the network, each computer i will have an estimate of the load at the neighbor computers, because within the communication delay d $□_{ij}$ some load may be added to the computer j or removed from the computer j. The load information from computer j to computer i is represented $L_{ij}(t) = L_j(\tau_{ij}(t))$ where $\tau_{ij}(t)$ is a certain time instant satisfying $0 \le \tau_{ij}(t) \le t$. The computer i, as d_{ii} =0 (delay is zero for the computer i) will have exact information about its load.

A set of time instants is associated with each computer for doing load balancing. At a given time instant, the computer i executes the load balancing algorithm by comparing its load with the estimated load of its neighbors that are stored in the computer i local memory during status exchange epoch. In order to analyze the DDD behavior, the variable t is discriminated by assuming the values t=0, 1, 2….

When loads among the computers are distributed randomly, a single iteration of the proposed load balancing algorithm consists of two procedures: procedure LB and procedure Accurate LB. In procedure LB, when the load of the computer is greater than the average load of the domain of the computer *i*, then that the computer is said to be an overloaded computer, so it sends its excess load to the under load neighbors.

In the procedure AccurateLB of DDD, the computer that initiated the load balancing algorithm checks its underlying domain for the balanced state. The domain attains balanced state if the load difference between two computers in it is not greater than 1. If it is not balanced, a computer that initiated the load balancing algorithm balances its domain, in a refined way by sending messages to the overloaded computers in its domain to distribute the loads to the under loaded neighbors

The computer i compute the load average of its domain by taking the load information of the neighbors kept in the memory which is rounded to the nearest lowest integer value, which is given by

$$\bar{L}_i t) = \frac{L_i(t) + \sum_{j \in D_i} L_j(t)}{|D_i| + 1} \qquad (5)$$

After computing the load average, it evaluates the relative load weight to detect whether it is an overloaded computer or an under loaded computer. For this purpose it uses below formula

$$\mathbf{deviation_i(t) = L_i(t) - \bar{L}_i(t)} \qquad (6)$$

From the equation (5) If the value $deviation_i(t)$ *>0* indicates computer i is overloaded and it has to send its excess load to one of its deficient neighbors. The value $\mathbf{deviation_i(t)}$ <0 indicates that the computer is under loaded and there no need to transfer the load and hence no need to invoke the load balancing algorithm. Depending on the value of $deviation_i(t)$*)* the load balancing algorithm is initiated by the computer i."[17]

2.3.1 Load Transfer Calculation

"Once the computer i determine that it is having an excess load, it has to distribute its excess load to the deficient neighbors. The computer i form two sets $\mathbf{Active_i}$ and $\mathbf{Send_i}$ depending on the deficit and excess load values. Computers having the deficit loads form the Active set which is denoted by $\mathbf{Active_i}$ and computers having the excess loads form the set $\mathbf{Send_i}$. After forming the two sets, for each deficit neighbor in set $\mathbf{Active_i}$, load deficit for an individual computer is stored which is given by"[17]

Active$_i$ (t) = $\{j|\ L_{\downarrow}j\ (t) \le L^{-}{}_{\downarrow}i\ (t)\}$ where $j \in D_i \cup \{i\}$

Send$_i$(t) = $\{k|\ L_{\downarrow}j\ (t) > L^{-}{}_{\downarrow}i\ (t)\}$ where k $\in D_i \cup \{i\}$ $\mathbf{dev_{ij}(t) = \bar{L}_i(t) - L_j(t)}$

The formula for the total deficit of the domain of computer i is

$$TD_i(t) = \sum_{j \in \{Active_i\}} dev_{ij}(t) \qquad (7)$$

The computer i determines the total deficit load and then calculates how much portion of its excess load is to be sent to each of its deficit neighbors by use of the below formula

$$x_{ij}(t) = \frac{dev_{ij}(t)}{(TD_i(t))(deviation_i(t))} \quad (8)$$

2.3.2 Accurate Load Movements

"The procedure AccurateLB is used by computer i to check its domain for accurate balance. To do this, some additional parameters are required to probe for unbalanced domains. These parameters will do accurate load movements to balance the domain and hence to decrease the variance of the domain. The additional parameters introduced in the algorithm are:

- Maximum load value of the computer in the domain of computer i : $Max\{Send_i\}$
- minimum load value of the computer in the domain of computer i : $Min\ \{Active_i\}$

To detect the imbalance in the domain of $node\ i$, the above two parameters will be used . If the difference of the maximum load value of a computer in $Send_i$ and a minimum load value of a computer in $Active_i$ is greater than 1 which is given by $(L_i^{max(t)} - L_i^{min(t)} > 1)$ then the domain is said to be unbalanced, which requires some load movements to reduce the variance in the system."[17]

2.3.3 Load Adjustments between Computers

The computer i detect the imbalance in its domain and moves some of the load units to the neighboring computers to reduce the load variance in its domain. The following steps are used to reduce the load variance in the domain of computer i.

Step 1: In the domain of computer i, if computer i is the computer with maximum load and other computers in the domain contain an equal amount of load, then computer i will distribute $\varphi_i(t)$ units of load to its neighbors one by one. The computed value of $\varphi_i(t)$ in (9), is used to lower bound the load value of the computer i by the load average of its domain. The computer i distribute its load based on the neighbor computers order kept in its memory. The value of $\varphi_i(t)$ will always be smaller than the number of neighbors.

$$\varphi_i(t) = L_i^{max(t)} - L_i^{min(t)} - 1 \quad (9)$$

Step 2: The load value of computer i is maximum in its domain and other computers in have different load values in domain of computer i, then a unit of load is distributed among the least loaded neighbors in its domain.

Step3: If computer i is not having the maximum load in its domain, instead some other computer k is having the maximum load, then it would send a message, instructing computer k to send one unit of a computer k load to one its least loaded neighbor until the value of $\varphi_i(t)$ runs out.

The difference of the load between the computers in the domain of computer i is greater than 1, $\varphi_i(t)$ load units need to be transferred. Two constraints must be satisfied while transferring $\varphi_i(t)$ load units to the computers of domain of computer i.

Constraint 1: The excess load units sent by the computer in the set $Send_i$ to the deficit neighbors in the set $Active_i$, the sender computer load value must be equal or greater than one load unit to the next higher loaded computer in $Send_i$ to avoid the task shuttle between the computers. From this, it will be concluded that the largest computer in $Send_i$ remains as largest or equal to the next higher loaded computer in $Send_i$, after sending $\varphi_i(t)$ units of a load to a least loaded deficit computer in $Active_i$.

Constraint 2: After receiving $\varphi_i(t)$ load units from the computer in the set $Send_i$, the least loaded computer load value in the set $Active_i$ must be equal to the load value of the next least loaded computer in $Active_i$. The above two constraints play a key role in avoiding task thrashing effect.

The above operations have been computed by $node\ i$ depending on the load values stored in its memory. From above it is clear that the amount of load sent by $node\ i$ to its neighboring computers in a time t and the load value of the $node\ i$ in time t+1 is given by "[17]

$$L_i(t+1) = L_i - \sum_{j=1}^{N} x_{ij}(t) - \varphi_i(t) + \sum_{j=1}^{N} x_{ij}(t) \quad (10)$$

3. Simulations

Here, the results of a simulation study and performance comparison of the proposed algorithm with the existing algorithms in the literature are presented. The information that is made available at a given instant of load balancing decision for transferring the load from overloaded computers to under loaded computers will have a significant effect on the relative performance of the algorithms. The below information is used by the algorithms for balancing loads among the computers in the distributed system.

N: Number of Computers

E[T]= Mean task inter-arrival time to a computer

λ_i = Arrival rate at computer i($\lambda_i = \frac{1}{E[T]}$)

M = Exponential distribution of tasks arrival process

μ_i = Service rate at computer i

ρ_i = Utilization of computer i=$\frac{\lambda_i}{\mu_i}$

ρ = System load=$\sum_{i=0}^{i=N} \frac{\rho_i}{N}$

H= Hyper- Exponential distribution describing task service demands

E[S]= Expected service time of a task

σ_S = Standard deviation of task service time

The distributed system consists of computers connected by communication links. Here it is assumed that the computers in a distributed system have the same processing capabilities and communication delay is not arbitrarily large. An arbitrary topology generated consisting of 10 to 50 computers with edges connecting the computers. To evaluate the performance of the system, the load at the system is varied by reducing or increasing the mean inter arrival time for tasks at each computer.

The simulation has been carried out to observe the performance of the proposed algorithm (Dynamic Distributed Diffusion) with the existing algorithms SID, GDE, AN by considering task arrive to a computer with a Poisson process with different system load levels [0.1 to 0.9]. The task size is constant with 100Kbytes. Here it is assumed N* M/D/1 Queuing model.

The simulation conditions are kept identical for all the algorithms same. It is assumed that all the computers in a system have the service rate of 1 and the service time for each task is constantly distributed with a service time of 20 sec. The simulation runs for 5000 task arrivals to measure the throughput of the system and the load balancing time of the algorithms. Each run was replicated five times with different random seeds and the results averaged over replications where 90% of a confidence interval is obtained for each data point.

3.1 Influence of Computer Size on Load Balancing Time

The load balancing time is tested for the varying load index value of the computers in a distributed system. The input value is varied from under loaded situations to overloaded situations by changing the parameter ρThe scalability of the algorithms are tested by varying the size from 10 to 50 computers in the distributed system. It has been found that all the algorithms inclusive of the proposed algorithm done well in the light load conditions (ρ=0.1).However the proposed algorithm has done well when compared to the existing algorithms in the literature under moderate and heavy load conditions (ρ=0.4& ρ=0.9).

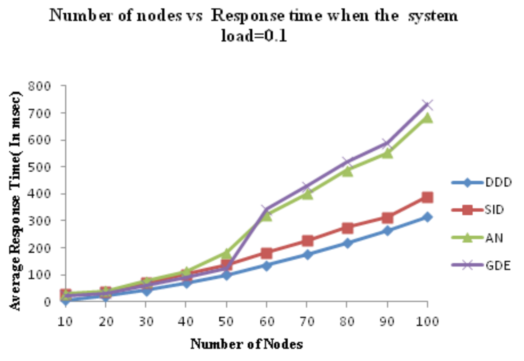

Figure 1. The effect of computer size on Load balancing time when ρ =0.1.

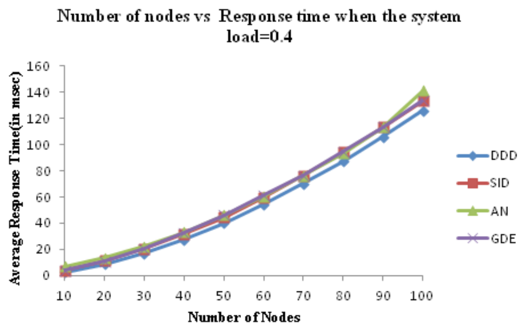

Figure 2. The effect of computer size on load balancing time when ρ =0.4.

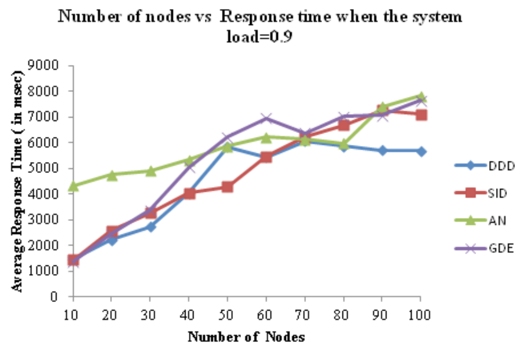

Figure 3. The effect of computer size on load balancing time when ρ =0.9.

The load balancing times taken by the algorithms are shown in the graphs presented in the Figures 1, 2, 3. The execution time increases with the increase in the number of computers has been observed while running the simulation.

4. Conclusions and Future Work

In this paper, the proposed algorithm considers arbitrary topology for load balancing in distributed systems. Still the load balancing problems is an NP-complete problem and remains as a challenging task in the research field. Since the algorithms proposed in this paper are iterative in nature, the issue that arises in designing the load balancing algorithms of such nature is the stable load distribution. The proposed algorithms do not attain the stable state (1 in 25 tests), and they are terminated by inducing an upper delay in the program.

However, for the majority cases, the proposed algorithm DDD starts working fine when compared to the existing algorithms proposed in the literature. The goal of minimizing the response time is achieved through the proposed algorithm. The proposed load-balancing algorithm had less load balancing overhead that has an impact on average response time of the system. When it comes to scalability, most of the load balancing algorithms existing in the literature has failed for the larger system sizes. However, the proposed algorithm attained stable distribution in both static load situations as well as in dynamic load situations and runs well for the computer sizes up to 100.

In the future work, a load balancing algorithm shall be developed for homogeneous systems by considering communication delay while transferring tasks to the different computers in distributed systems. In this paper, the proposed algorithm assumes the scheduling policy is First Come & First Serve basis. Both in homogeneous and heterogeneous systems OS Scheduling policies like round robin, priority scheduling must be taken into account while doing load balancing. Another aspect to be considered by the load balancing algorithm includes the deadline of the task, where the task is scheduled in such a way that it has to meet its deadline. This paper does not consider tasks with deadline as it requires extensive research and different techniques to be incorporated.

5. References

1. Tang X, Chanson ST. Optimizing static job scheduling in a network of heterogeneous computers. Proceedings of the International Conference on Parallel Processing, Toronto, Ont. 2000. p. 373–82.
2. Zuhair K et al. Mizan: a system for dynamic load balancing in large-scale graph processing. Proceedings of the 8th ACM European Conference on Computer Systems. Enosys'13, NY. 2013. p. 169–82.
3. Cortes A, Cedo F et al. On the Stability of a Distributed Dynamic Load Balancing Algorithm. Proceedings of the 1998 International Conference on Parallel and Distributed Systems, Tainan. 1998. p. 435–46.
4. Corradi A, Leonardi L, Zambonelli F. Diffusive Load-Balancing Policies for Dynamic Applications. IEEE Concurrency. 1999; 7(1):22–31.
5. Cortes A, Ripoll A, Cedo F, Senar MA, Luque E. An asynchronous and iterative load balancing algorithm for discrete load model. Journal of Parallel Distributed Computing. 2002; 62(12):1729–46.
6. Elsasser R, Monien B, Preis R. Diffusive load balancing schemes on heterogeneous networks. Proceedings of The Twelfth Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA '00). ACM, New York, NY, USA. 2000; 30–8.
7. Saletore VA. A Distributed and Adaptive Dynamic Load Balancing Scheme for Parallel Processing of Medium-Grain tasks. In Proceedings of the 5th Distributed Memory Computing Conference. 1990; 2. p. 94–9.
8. Ferrari D, Zhou S. An empirical investigation of load indices for load balancing applications. Proceedings of Performance '87, the 12th Int'l Symposium on Computer Performance Modeling, Measurement and Evaluation. 1988; 23:515–28.
9. Fontlupt C, Marquet P, Dekeyser J. Data parallel load balancing strategies. Parallel Computing. 1998; 24(11):1665–84.
10. Antonis K, Garofalakis J, Mourtos I, Spirakis P. A hierarchical adaptive distributed algorithm for load balancing. Journal of Parallel and Distributed Computing. 2004; 64(1):151–62.
11. Yeon MS, Jeong BS. Multi-level load balancing methods for hierarchical web server clusters. Indian Journal of Science and Technology. 2015 Sep; 8(21):1–5.
12. Couturier R, Vernier F et al. A decentralized convergence detection algorithm for asynchronous parallel iterative algorithms. IEEE Transactions on Parallel Distributed Systems. 2005; 16(1):4–13.
13. Lavanya M, Ravi A, Aditya A, Samyuktha R, Vaithiyanathan V, Saravanan S. An enhanced load balancing scheduling approach on private clouds. Indian Journal of Science and Technology. 2015 Dec; 8(35):1–4.

14. Singh A, Juneja D, Malhotra M. Autonomous Agent Based Load Balancing Algorithm in Cloud Computing. Procedia Computer Science. 2015; 45:832–41.
15. Panwar R, Mallick B. Load balancing in cloud computing using dynamic load management algorithm. 2015 International Conference on Green Computing and Internet of Things (ICGCIoT), Noida. 2015. p. 773–78.
16. Balaji N, Umamakeshwari A. Load Balancing in Virtualized Environment - A Survey. Indian Journal of Science and Technology. 2015 May; 8(S9):230–34.
17. Neelakantan P. Article: Load Balancing in Distributed Systems using Diffusion Technique. International Journal of Computer Applications. 2012 Feb; 39(4):1–10.