International Journal of Research in Computer Science
eISSN 2249-8265 Volume 3 Issue 3 (2013) pp. 9-15
www.ijorcs.org, A Unit of White Globe Publications
doi: 10.7815/ijorcs. 33.2013.063

AN ADAPTIVE LOAD SHARING ALGORITHM FOR HETEROGENEOUS DISTRIBUTED SYSTEM

P. Neelakantan
Department of CSE, SVUCE, Tirupati, A.P, INDIA
E-mail: pneelakantan@rediffmail.com

Abstract: *Due to the restriction of designing faster and faster computers, one has to find the ways to maximize the performance of the available hardware. A distributed system consists of several autonomous nodes, where some nodes are busy with processing, while some nodes are idle without any processing. To make better utilization of the hardware, the tasks or load of the overloaded node will be sent to the under loaded node that has less processing weight to minimize the response time of the tasks. Load balancing is a tool used effectively for balancing the load among the systems. Dynamic load balancing takes into account of the current system state for migration of the tasks from heavily loaded nodes to the lightly loaded nodes. In this paper, we devised an adaptive load-sharing algorithm to balance the load by taking into consideration of connectivity among the nodes, processing capacity of each node and link capacity.*

Keywords: *Load balancing, Distributed System, heterogeneous, response time.*

I. INTRODUCTION

An important attribute in a dynamic load balancing policy is to initiate the load balancing activity. The balancing activity specifies which node is responsible for detecting imbalance of the load among the nodes[9]. A load-balancing algorithm is invoked when load imbalance among the nodes is detected. The initiation of load balancing activity will have a higher impact on complexity, overhead and scalability. The load balancing algorithm is designed in such a way to make the overloaded node to transfer its excess load to the underloaded node which is called sender – initiated and when underloaded node requests the load from the overloaded node then it is called receiver- initiated [6] [8].

Domain balancing is used to decentralize the balancing process by minimizing its scope and decrease the complexity of the load balancing algorithm. A domain is defined as subset of nodes in a system, such that a load balancing algorithm can be applied for this subset of nodes in a single step. Domain balancing is used in load balancing algorithms to decentralize the balancing. The balancing domains are further divided into two types: The first type is overlapped domains, which consists of node initiating the balancing activity and balances its load by migrating the tasks or load units with the set of surrounding nodes [3].

Global balancing is achieved by balancing every domain and by diffusing the excess load throughout the overlapped domains in a distributed system. Another important attribute in load balancing algorithm is the degree of information. The degree of information plays an important role in making the load balancing decisions. To achieve global load balancing in a few steps, the load balancing should get absolute information instead of getting the obsolete information from the nodes. In general, the collection of information by a node is restricted to the domain or nearest neighboring nodes (which are directly connected to a node) [4].

Although collecting information from all the nodes in a distributed system gives the exact knowledge of the system, it introduces large communication delay, so from this perspective, it will have a negative impact on the load balancing algorithm. In such cases, it has been observed, that the average response time is kept minimum without load balancing instead of doing the load balancing which induces overhead in migrating the load from one node to another node in the system [5].

In this section, an abstract view of the software details is presented for load balancing. The distributed system consists of several nodes and the same load balancing software is installed to run on all the nodes in the distributed system. By installing the same software in all the nodes, the load balancing decision is taken by a node locally (decentralized) by collecting the information from the neighboring nodes as opposed to the centralized load balancing policy[14].

The program must use a multi-threaded concept to implement load balancing in a distributed system. Two communication ports are available: TCP and UDP. UDP is preferable as it incurs less communication overhead. In general the architecture provides three

layers: Communication layer, Load balancing process and application layer[14][10]. For storing information two data structures were used.

The communication link is responsible for four phases: node status information phase, node status reception, tasks reception and task migration. The node status information is responsible for disseminating the load information to the node that has requested it. The exchange of the information has a profound effect on the load balancing decision; it has to be done according to the predefined intervals of time specified on each node [7] [14].

The status reception is responsible for receiving the status information from the other nodes and it will be updated in the local node list which is running the status reception phase. Here it is possible to distinguish the old information from the new information. The technique that is used to find is to associate the timestamp for the information that it has received from some node (say $TS_j^i(Inf)$, the time stamp attached to the information received from j to i). The local node say i maintaining the status about the node j is kept in the memory. If any estimate regarding node j exists in the node i memory, it will be compared to the received time stamp message and drops the old time stamp and the new timestamp message has been saved in the memory as the old time stamp has the obsolete information[11][1][2].

Once a node collects the above information, then it knows whether it is overloaded or underloaded. In case if it is overloaded node, then it transmits the excess tasks (loads) to the underloaded nodes in a "tasks transmission" phase. The next initiation of load balancing activity will be done only when the current migration of load units to the underloaded nodes is completed.

The "task reception" is responsible for listening to the requests and accepts the tasks sent from the other nodes. As we can observe from the above situations, the minimum time to initiate the new load balancing activity takes three time instants. One instant for receiving the status of all the nodes and second time instant for determining the underloaded nodes and computing the excess load and third time instant for transferring the excess load to the underloaded nodes which has been determined in the second time instant. So, the new load balancing activity takes place only at the fourth time instant [12] [14].

In a few papers [3] [9] [10], it is assumed that the nodes will not fail. The problem arises when the nodes fail which is common in the distributed systems. Sometimes a communication link will also fail, so the node will be unreachable. These two aspects i.e., failure of a node and the communication link will affect greatly the load balancing algorithms. Let us assume the following scenario. The overloaded node has collected the load information from the neighboring nodes and found some of the nodes are low loaded as discussed earlier. Now at the given time instant when the node tries to send its excess load to the overloaded node, it will not succeed because of the failure of the node. The node may fail after sending the status information. If this happens, an alternative must be chosen to avoid a failure of the load-balancing algorithm.

II. NOTATIONS & ASSUMPTIONS

N : Number of nodes

$V= \{1, 2,\ldots, N\}$ a set of nodes in a system

$x_i(t)$: Expected waiting time experienced by a task inserted into the queue at the i^{th} node in time t

$A_i(t)$: rate of generation of waiting time on i^{th} node caused by the addition of tasks in time t.

$S_i(t)$: rate of reduction in waiting time caused by the service of the tasks at the i^{th} node in time t.

$r_i(t)$: rate of removal(transfer) of the tasks from node j to node i at time t by the load balancing algorithm at node j.

ts_i : Average completion time of the task at node i.

b_i : Average size of the task in bytes at node i when it is transferred

d_{ij} : Transfer rate in bytes/sec between node i and node j

$\bar{x}_i(t)$: Average size of the queue calculated by node i based on its domain information at time t.

D_i: Neighboring nodes to i which is defined as $D_i = \{j | j \in V \text{ and } (i,j) \in E\}$ where $V= \{1, 2\ldots N\}$

$E_i(t)$: Excess number of tasks at node i at time t.

f_{ij} : Portion of the excess tasks of node i to be transferred to node j decided by the load balancing algorithm.

The following assumptions were made in this paper:

1. It is assumed that a distributed system consists of N heterogeneous nodes interconnected by an underlying arbitrary communication network. Each node i in a system has a processing weight $P_i > 0$ and processing capacity $S_i > 0$. The load is defined to be $L_i = P_i/S_i$. In homogenous case the value of $L_i = P_i$.
2. It has been assumed that tasks arrive at node i according to Poisson process with rate $\lambda_i(t)$. A task arrived at node i may be processed locally or migrated through the network to another node j for remote processing. Once the task is migrated, it remains there until its completion.

3. It is assumed that there is a communication delay incurred when task is transferred from one node to another before the task can be processed in the system. The communication delays are different for each link.

Each node contains an independent queue where arrived tasks are added to the queue, which results in accumulation of waiting time. Load balancing must be done repeatedly to maintain load balance in the system. The proposed algorithm is distributed in nature meaning that each node runs load-balancing algorithm autonomously.

The second level of the system is a load-balancing layer, which consists of load balancing algorithms. The load balancing process is initiated by suing predefined time instants or randomly generating which is kept in a file. The algorithm determines the portion of the excess load to be sent to the underloaded node based on the current state of the node and availability of the nodes in the network. The load balancing algorithm must consider the communication delay while migrating the tasks to the other nodes. The algorithm selects the tasks to migrate to other nodes by setting their status as inactive to avoid execution of the tasks by current node application during the transition period. After completion of the task transmission activity, the status of the tasks is set to active when they are not transmitted to any node. when the tasks are transmitted to other nodes during the task transmission phase then those tasks are removed from the task queue of the current node.

Application layer consists of two threads of control: Task input and task execution threads. The task input creates a number of tasks defined in the initialization file and inserts them in the task queue. This task input is also responsible for adding the new tasks to the task queue either from the current node or from other nodes in the system. The task execution thread is responsible for execution of the tasks and updating the QSize variable by removing the task from the task queue.

The load balancing policy must take into account of processing capacity of the node while migrating the tasks to it. The selected node may become a candidate for one or more overloaded node in a given time instant because of the decentralized policy. Another issue to be considered is variable task completion times. Taking these issues in priori is not possible so a load balancing strategy must be adaptive to the dynamic state changes in the system and act accordingly to transfer the tasks. Even this can result in task shuttle between the nodes, so a migration limit for a task should be set to avoid task thrashing.

Another issue to be considered while migrating the tasks from one node to another node in a system is communication overhead. Large communication delays will have a negative impact on the load balancing policy, so, the transfer delays must be taken into account while migrating the task. When the completion of the task time in current node is greater than the completion time on task in another node inclusive of communication overhead, then only a task is considered for migration.

III. MATHEMATICAL MODEL

The mathematical model for load balancing in a given node i is given by [1][2]

$$\frac{dw_i(t)}{dt} = A_i - S_i + r_i(t) - \sum_{j=1}^{\neq N_i} f_{ij} \frac{ts_i}{ts_j} r_j(t - \tau_{ij}) \quad (1)$$

$$E_i(t) = q_i(t) - \bar{q}_i(t)$$

$$r_i(t) = G_i(E_i(t))$$

$$f_{ij} \geq 0, f_{ii}=0, \sum_{j=1}^{\neq N_i} f_{ij} = 1$$

$$E_i(t) = \begin{cases} E \; if \; y \geq 0 \\ 0 \; if \; y < 0 \end{cases}$$

When a task is inserted into the task queue of node i, then it experiences the expected waiting time which is denoted by $w_i(t)$.

Let the number of tasks in i[th] node is denoted by $q_i(t)$.

Let the average time needed to service the task at node i ts_i .

The expected (average) waiting time is given by at node i is given by $w_i(t) = q_i(t)ts_i$.

Note that $w_i(t)/ts_i = q_i$ is the number of tasks in the node i queue.

Similarly $w_k(t)/ts_k = q_k$ is the queue length of some node k. If tasks on node i were transferred to some node k, then the waiting time transferred is $q_i ts_k = \frac{w_i(t) ts_k}{ts_i}$, so that the fraction ts_k/ts_i converts waiting time on node i to waiting time on node k.

A_i : Waiting time generated by adding the task in the i[th] node.

S_i : Rate of reduction in waiting time caused by the service of tasks at the i[th] node is given by $S_i = (1 * tp_i)/tp_i$=1 for all $w_i(t) > 0$.

$r_i(t)$: The rate of removal (transfer) of the tasks from node i at time t by the load balancing algorithm at node i. f_{ij} is the fraction of i[th] node tasks to be sent out to the j[th] node. In more detail $f_{ij}r_i(t)$ is the rate at which node i sends waiting time (tasks) to node i at time t where f_{ii}>=0 and f_{ii}=0.That is, the transfer from node i of expected waiting time (tasks) $\int_{t_1}^{t_2} E_i(t)dt$ in the interval of time $[t_1, t_2]$ to the other nodes is carried out with the j^{th} node receiving the fraction $p_{ij}(t_{p_j}/t_{p_i}) \int_{t_1}^{t_2} u_i(t)dt$ where the ratio t_{p_j}/t_{p_i} converts the task from waiting time on node i to waiting time on

node j. As $\sum_{i=1}^{n}(f_{ij}\int_{t_1}^{t_2}E_i(t)dt) = \int_{t_1}^{t_2}E_i(t)dt$, this results in removing all of the waiting time $\int_{t_1}^{t_2}E_i(t)dt$ from node i.The quantity $f_{ij}E_i(t-\tau_{ij})$ is the rate of increase (rate of transfer) of the expected waiting time (tasks) at time t from node i by (to) node j where $\tau_{ij}(\tau_{ii}=0)$ is the time delay for the task transfer from node i to node j.

In this model, all rates are in units of the rate of change of expected waiting time, or time/time which is dimensionless. As $E_i(t) \geq 0$, node i can only send tasks to other nodes and cannot initiate transfers from another node to itself. A delay is experienced by transmitted tasks before they are received at the other node. The control law $E_i(t) = G_i * E_i(t)$ states that if the i^{th} node output $w_i(t)$ is above the domain average $(\sum_{j=1}^{n} q_j(t-\tau_{ij}))$/n, then it sends data to the other nodes, while if it is less than the domain average nothing is sent. The j^{th} node receives the fraction $\int_{t_1}^{t_2}F_{ij}\,(t_{p_i}/t_{p_j})\,u_i(t)dt$ of transferred waiting time $\int_{t_1}^{t_2}E_i(t)dt$ delayed by the time τ_{ij}.The model described in (1) is the basic model for load balancing, but an important feature is to determine f_{ij} for each underloaded node j. One approach is to distribute the excess load equally to all the underloaded neighbors.

$$f_{ij} = \frac{1}{n-1} \text{ for i}\neq\text{j.}$$

Another approach is to use the load information collected from the neighbors to determine the deficit load of the neighbors. The deficit load of the neighbors shall be determined by node i by using the formula (2)

$$q_j(t\text{-}\tau_{ij}) - \bar{q}_i \quad (2)$$

The above formula is used by node i to compute the deficiency waiting times in the queue of node j with respect to the domain load average of node i.

If node j queue is above the domain average waiting time, then node i do not send tasks to it. Therefore $(\bar{q}_i - q_j(t\text{-}\tau_{ij}))$ is a measure by node i as how much node j is behind the domain average waiting time. Node i performs this computation for all the other nodes which are directly connected to it and then portions out its tasks among the other nodes that fall below the domain queue average of node i.

$$f_{ij} = \frac{(\bar{q}_i - q_j(t-\tau_{ij}))}{\sum_{j=1}^{N_i}(\bar{q}_i - q_j(t-\tau_{ij}))} \quad (3)$$

If the denominator $\sum_{j=1}^{N_i}(\bar{q}_i - q_j(t-\tau_{ij}))=0$ then f_{ij} are defined to be zero then no waiting times are transferred. If the denominator $\sum_{j=1}^{N_i}(\bar{q}_i - q_j(t-\tau_{ij}))=0$, then$(\bar{q}_i - q_j(t-\tau_{ij}) \leq 0 \forall j \in N_i$.

However by definition of the average $\sum_{j=1}^{N_i}(\bar{q}_i - q_j(t-\tau_{ij}) + \bar{q}_i - q_i(t) = \sum_{j=1}^{N_i}(\bar{q}_i - q_j(t-\tau_{ij}))=0$ which implies

$$\bar{q}_i - q_j(t) = \sum_{j=1}^{N_i}(\bar{q}_i - q_j(t-\tau_{ij})) > 0$$

That is, if the denominator is zero, the node j is not greater than its domain queue average, so $E_i(t) = G_i . E_i(t))=0$, where G is Gain Factor.f_{ij} : Portion of the excess tasks of node i to be transferred to node j decided by the load balancing algorithm.

Except the last three parameters remaining information is known at the time of load balancing process. Before the instance of load balancing activity, every variable is updated.

IV. ALGORITHM

A. *Algorithm ALS*

The current node i, performs the followings:

a. Calculate the average queue size $(\bar{q}_i)$based on the information received from the neighbouring nodes.

$$\bar{q}_i = \frac{1}{\neq N_i+1}\sum_{j=1}^{\neq N_i}(q_i + q_j\frac{ts_j}{ts_i})$$

if $(q_i > \bar{q}_i)$then $E_i = (q_i - \bar{q}_i) * G$
else Exit.

b. Determine the participant nodes in load sharing process.
Participants= $\{j|\, q_j < \bar{q}_i, \forall j \in N_i\}$

c. Calculate the fraction of the load (f_{ij}') to be sent to the participants

$$f_{ij}' = \frac{\bar{q}_i - (\frac{ts_j}{ts_i})q_j}{\sum_{j=1}^{N_i}(\bar{q}_i - (\frac{c_j}{c_i})q_j}$$

d. Calculate maximum portion of the excess load (f_{ij}'')

$$f_{ij}'' = \frac{(q_i - E_i)\, ts_i \text{ dij}}{E_i b_i}$$

e. fij = Min (f_{ij}', f_{ij}'')

f. For j∈ Participants
   i. Announce to node j about its willingness to send $T_{ij} = fij * E_i$ tasks;
   ii. nowReceived = call procedure acceptanceFrom Nodej()
   iii. if(nowReceived >0)
      1. Transfer NowReceived to j
      2. $T_{ij} = T_{ij}$- NowReceived

End if

g. Repeat steps from (a) to (f).

Procedure acceptanceFromNodej()
if $((q_j + T_{ij}) < \bar{q}_j$ nowSend$= \bar{q}_j - q_j$;
else nowSend=-1;

return now Send;
end acceptanceFromNodej

In general it is assumed that keeping the Gain factor G=1 will give the good performance. But in a distributed system with largest delays and the nodes that have domain queue average outdated gives poor result. This phenomenon was first observed by the load balancing group at the University of New Mexico [7]. So the G values are set in the way that yields an optimal result. Another step that is added in the above algorithm is to test the node availability. It checks both node availability as well as the amount of waiting times it can receive. The node executing the ALS is permitted to send the tasks to the neighbors after receiving the acknowledgement specifying the amount of the load they can be able to process.. The time complexity of the proposed algorithm is O(d) shown in the table 1.

Table 1: ALS Operations

Sno	Actions	Operation	Quantity, (d is the number of neighbors)
1	Compute average queue size	Addition Division Multiplication	d+1 d d
2	Compute Ei	Subtraction Multiplication	1 1
3	Determine the participant nodes	Comparison	d
4	Compute f_{ij}'	Subtraction Division Multiplication	d+1 d+1 d+1
5	Compute f_{ij}''	Subtraction Division Multiplication	1 1 3
6	Compute T_{ij}	Multiplication	d
7	Message to node	Transfer	d
8	Compute nowReceived	Addition Comparison Message Transfer	d d d

V. SIMULATION

To test the performance of the newly proposed load-balancing policy, a Java program is developed to test the performance of the existing and proposed algorithms. The existing algorithms, ELISA and DOLB are used to compare with the proposed algorithm ALS. The DOLB is very much related to the above problem. The initial settings and parameters are shown in Table 2. The average network transfer rates between each node are represented by the cost adjacency matrix.

The proposed algorithm ALS is tested with DOLB & ELISA for the gain values *G* between 0.3 and 1 with 0.1 incremental steps. The α parameter introduced in the previous section was set to 0.05 by running several experiments and observing the behavior of the *tsi* parameter. Note that, the first time the load-balancing process was triggered was after 40s from the start of the system and then the strategy was executed regularly at 20s interval.

Table 2: Simulation Parameters

Number of nodes	16,32,64
Initial task distribution	[100..1000] tasks distributed randomly at each node
Average task processing time(ts in ms)	Processing time is randomly distributed in a range[300…800]
Size of task(in Mb)	1
Load balancing instance	First time the load balancing was triggered at 5s then for every 10s the load balancing is initiated
Bandwidth distribution (d_{ij})	A cost adjacency matrix denotes the transfer rate between the nodes.
Number of nodes	16,32,64
Initial task distribution	[100..1000] tasks distributed randomly at each node
Average task processing time(ts in ms)	Processing time is randomly distributed in a range[300…800]

This was done to ensure that the *ts* parameter had enough time to adapt and reflect the current computational power of each node before the occurrence of any tasks migration between the nodes. Note that the ratio $\frac{ts_i}{ts_j}$ are fixed over time. The proposed and rival methods were evaluated by conducting 10 runs for each value of *G between* 0.3 and 1 with 0.1 incremental step.

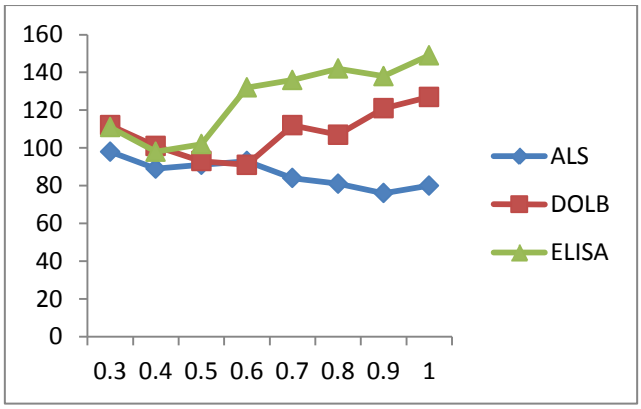

Figure 1: Completion time averaged over 5 runs vs different gain values K. The graphs shows the results of three policies for system size=64.

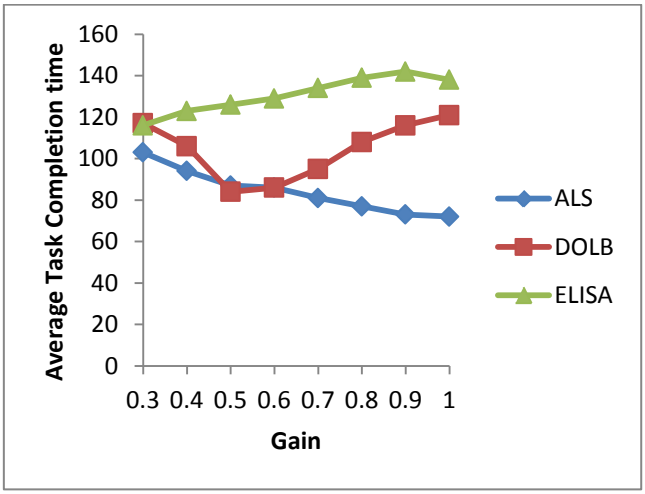

Figure 2: Completion time averaged over 5 runs vs. different gain values K. The graphs shows the results of three policies for system size=32.

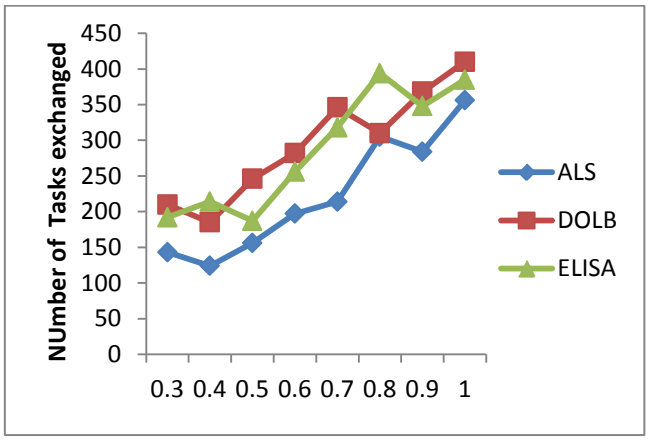

Figure 3: Total number of tasks exchanged averaged over 5 runs Vs different Gain values K. The graphs shows the performance of the three policies for system size=16.

VI. CONCLUSION

The proposed algorithm is better when compared to the existing algorithms in the literature. In simulation, we assumed the tasks with no precedence and with no deadlines. However, as a future work, the algorithm must focus on considering the tasks with dead line and tasks with precedence relations.

VII. REFERENCES

[1] M. M. Hayat, S . Dhakal, C. T. Abdallah I 'Dynamic time delay models for load balancing. Part II: Stochastic analysis of the effect of delay uncertainty, CNRS-NSF Workshop: Advances in Control of tirne-delay Systems, Paris France, January 2003.

[2] J. Ghanem, C. T. Abdallah, M. M. Hayat, S. Dhakal, J.D Birdwell, J. Chiasson, and Z. Tang. Implementation of load balancing algorithms over a local area network and the internet. 43rd IEEE Conference on Decision and Control, Submitted, Bahamas, 2004. doi: 10.1109/CDC.2004.1429411

[3] L. Anand, D. Ghose, and V. Mani, "ELISA: An Estimated Load Information Scheduling Algorithm for Distributed Computing Systems," Int'l J. Computers and Math With Applications, vol. 37, no. 8, pp. 57-85, Apr. 1999. doi: 10.1016/S0898-1221(99)00101-7

[4] WEI Wen-hong, XIANG Fei, WANG Wen-feng, et al. Load Balancing Algorithm in Structure P2P Systems[J],Computer Science, 2010, 37(4):82-85.

[5] Khalifa, A.S.; Fergany, T.A.; Ammar, R.A.; Tolba, M.F," Dynamic online Allocation of Independent tasks onto heterogeneous computing systems to maximize load balancing," IEEE International Symposium on Signal Processing and Information Technology, ISSPIT 2008,Page(s): 418 – 425. doi: 10.1109/ISSPIT.2008.4775659

[6] Andras Veres and Miklos Boda. The chaotic nature of TCP congestion control. In Proceedings of the IEEE Infocom, pages 1715-1723, 2000. doi: 10.1109/INFCOM.2000.832571

[7] J. Chiasson, J. D. Birdwell, Z. Tang, and C.T. Abdallah. The effect of time delays in the stability of load balancing algorithms for parallel computations. IEEE Conference on Decision and Control, Maui, Hawaii, 2003. doi: 10.1109/CDC.2003.1272626

[8] Ming wu and Xian-He sun, A General Self Adaptive Task Scheduling System for Non Dedicated Heterogeneous Computing, In Proceedings of IEEE International Conference on Cluster Computing, PP 354-361, Dec 2003. doi: 10.1109/CLUSTR.2003.1253334

[9] Z. Zeng and B. Veeravalli, "Design and Performance Evaluation of Queue-and-Rate-Adjustment Dynamic Load Balancing Policies for Distributed Networks", presented at IEEE Trans. Computers, 2006, pp.1410-1422. doi: 10.1109/TC.2006.180

[10] K. Lu, R. Subrata, and A. Y. Zomaya, Towards decentralized load balancing in a computational grid environment, in: Proceedings of the first International Conference on Grid and Pervasive Computing, 2006, Vol. 3947, pp. 466-477, Springer-Verlag Press. doi: 10.1007/11745693_46

[11] Acker, D., Kulkarni, S. 2007. A Dynamic Load Dispersion Algorithm for Load Balancing in a Heterogeneous Grid System. IEEE Sarnoff

Symposium, 1- 5, 2007. 10.1109/SARNOF.2007.4567375

[12] M. Luczak and C. McDiarmid. On the maximum queue length in the supermarket model. The Annals of Probability, 34(2):493–527, 2006. doi: 10.1214/00911790500000710

[13] Zhou, S. (1987). An Experimental Assessment of Resource Queue Lengths as Load Indices. Proc. Winter USENIX Conf., p.73-82.

[14] J. Ghanem, C. T. Abdallah, M. M. Hayat, S. Dhakal, J.D Birdwell, J. Chiasson, and Z. Tang. Implementation of load balancing algorithms over a local area network and the internet. 43rd IEEE Conference on Decision and Control, Submitted, Bahamas, pp 4199-4204, 2004.

How to cite

P. Neelakantan, " An Adaptive Load Sharing Algorithm for Heterogeneous Distributed System ". *International Journal of Research in Computer Science*, 3 (3): pp. 9-15, May 2013. doi: 10.7815/ijorcs. 33.2013.063