An incremental load balancing approach for heterogeneous distributed processing systems

Dharavath Ramesh , Alok Kumar Pani

Abstract. Distributed computing is a high performance computing that solves complicated tasks and provides powerful computing abilities. The main purpose of distributed computing is to share the computational power, storage memory, and network resources to solve a large problem. Efficient resource management and job scheduling algorithms are two key issues in distributed computing environment. Design of an efficient strategy to achieve high performance in distributed computing environment is a challenging task. In this paper, we propose a new load balancing algorithm (NLBA) for heterogeneous distributed processing systems to balance and process the load among different resources in a distributed environment. This algorithm follows a strategy to reduce the overall processing time of jobs and also improves the throughput of processes.

Keywords. Cpu utilization, Load balancing, Distributed systems, Maximum priority queue

1 Introduction

Load balancing algorithms are designated to equally share the load on nodes or processors and maximize their utilization to solve a submitted task by minimizing their total execution time [1], [3], [4]. In order to achieve such kind of instances, the load balancing mechanism should be strong enough in distributing the load across the nodes. This makes the boundary difference between the heavily loaded and the lightly loaded nodes to minimize the competency of load sharing. Therefore, the load information at each node must be updated persistently so that the balancing mechanism can be more effective. Moreover, the execution of the load balancing instance should not take long time to compete with task assignments [5], [7]. In general, load-balancing algorithms are broadly divided as static and dynamic. Dynamic algorithms can be further classified into centralized or distributed [6], [21].

Static load balancing algorithms use only information about the average behavior of the system by ignoring the current state. Tasks will be assigned during compilation time and load balancing decisions are not affected by the state of the system. On the contrary, a dynamic load balancing algorithm reacts to the system state that changes dynamically. Here, the tasks are assigned to the processors at the time of their arrival.

Department of Computer Science and Engineering
Indian School of Mines, Dhanbad, Jharkhand 826004
{rameshd.ism , alok.kumar.pani} @ gmail.com

The potential of static algorithms is limited by the fact that they do not react to the current system state. On the other hand, dynamic algorithms draw an attention by responding to the current system state and are quite better able to avoid those states with poor performance [13], [15]. In a centralized dynamic scheduling scenario, the system state information is collected at a single node at which all scheduling decisions are made. This node is called the centralized server node. All requests related to process scheduling are handled by the centralized server which decides about the placement of a new process using the state information stored in it. The centralized approach can efficiently make process assignment decisions because the centralized server knows about the load at each node and the service required by number of processes. The remaining nodes periodically send update status messages to the central server node. A problem associated with the centralized mechanism is that of reliability. If the centralized server goes down, all scheduling in the system would remain ceased. In contrast to the centralized scheme, in a distributed dynamic scheduling environment, the work involved in making process assignment decisions is physically distributed among various nodes of the system. This kind of distributed scheme avoids the bottleneck of collecting state information at a single node and allows the scheduler to react quickly to dynamic changes in the system state [16], [20]. We further extend this methodology by proposing a new algorithmic instance to make the environment more dynamic in making scheduling decisions. Our methodology fulfills this scheduling aspect in a more suitable fashion to provide reliability.

2 Background literature

In the recent literature, a new method has been suggested to balance the load for data intensive applications with divisible load methodology [1]. Another method related to dynamic load balancing scheme is proposed for multi-agent computing systems [2]. It uses linear regression equations to make the selection and location. This regression methodology requires so many factor implications to make the selection correct. That's why it has not explored the dynamic instance of load balancing in a better way. A new contribution made by evolutionary learning [3] on dynamic load balancing problem has been described to make the selection process better. But this methodology is lacking in calculating response time with increased load. A game theoretic static load balancing approach is suggested for distributed systems [4]. This approach is static in nature, so it could not explore to balance the load dynamically. Another different load balancing scheme (master-less) for pipelined computations on heterogeneous systems have been suggested [5]. In this, the scheduling decisions were made by the workers in a distributed fashion. Some other new methodologies [6], [7],[18] have been proposed for adaptive load balancing. But they have not scaled well with respect to the CPU utilization. Another method for prediction of neighbor's load is described in distributed environment [8]. It has made its significance in calculating neighbor's load only. A model for load balancing in open source software has been outlined with a load balancer structure [9]. But it has not specified in terms of CPU utilization and response time. A comparison of existing load balancing algorithms for distributed systems has been depicted in [10],[15], while [11] presents a non cooperative approach for load balancing in heterogeneous platform.

Another method called heuristic approach in cluster environment is described to select a neighbor [12], while prioritization of node using heuristic approach is described for balancing the load [19]. These two differs with each other in comparison. A fault tolerant approach through load balancing is presented in [13]. At the same time the work presented in [14],[17] explores about a new dynamic load balancing algorithm in LAN. The LAN characteristic differs with other environments like WAN and CAN. Different types of self-aggregation techniques for load balancing schemes are described in [16]. For the independent tasks, load balancing scheduling method is proposed in [20] while [21] outlines a hybrid load scheduling strategy. To improve reliability in heterogeneous distributed systems, a decentralized load balancing approach has been suggested [22]. All these works have their own limitations in calculating CPU utilization. We further evaluate this instance by proposing a new load balancing approach (*NLBA*) to emphasize the overburden. Our methodology will exemplify the balancing criteria well suited and makes the environment dynamic.

3 Key constraints in load balancing

There are two major concerns for dynamic load balancing: (i) migration of tasks and (ii) sharing of load. In a given environment, random arrival of new tasks can cause certain processors into heavily loaded action while making the remaining processors as lightly loaded or idle. Hence the main intention of load balancing is to formulate a task assignment algorithm which enables migrating tasks from heavily loaded to lightly loaded nodes. It ensures in avoiding the idleness of a processor while there are other tasks waiting to be processed. In general, a dynamic load-balancing algorithm consists of four major components: the load measurement rule, the information exchange rule, the initiation rule, and the load balancing operation [8], [10].

3.1 Load measurement

In order to quantify the load information of a node, we take a variable called load index. When the processor is not loaded, the load index is set to zero and increases upon the processor load getting heavier. To ensure an efficient calculation of load measurement, simple and lesser number of parameters should be used like response time or completion time of all tasks[2],[11].

3.2 Information exchange

For making load balancing decision, this method specifies the way of collection and maintenance of workload information. The dynamic policies require frequent exchange of state information among the nodes of the system. Dynamic load balancing algorithm faces a transmission dilemma because of the two opposing impacts of the transmission of a message that has on the overall performance of the system. On one hand, the transmission improves the ability of the algorithm to balance the load while on the other way, it also enhances the expected queuing time of messages due to more utilization of the communication channel. Hence an appropriate selection of the state information exchange policy is essential. There are three basic information exchange policies:

(i) In periodic information exchange, the individual processors broadcast their state information to each other periodically after the elapse of a predetermined time interval. In this case, the load-balancing operation can be initiated based on the workload information without any delay. But it suffers from poor scalability issue since the number of messages generated for state information exchanges will be too large for networks having many nodes.

(ii) Next, the on-demand information exchange in which a node broadcasts a StateInformationRequest message when its state switches from the normal load region to either the underloaded region or the overloaded region. On receiving this message, other nodes send their current state to the requesting node. By this method, the number of communication messages is minimized, but extra delay is incurred for load-balancing operations.

(iii) In the case of on-state information exchange, a processor will broadcast its state information only when the state of the node changes. The state of the node will be changed when a process arrives at that node or when a process departs from that node. A process may arrive at a node either from the external source or from some other node in the system. A process departs from a node when either its execution is over or it is transferred to some other node.

In order to maintain a global view of the system state, the large communication overhead makes this rule inefficient for large systems. Hence a dedicated processor should collect and maintain the system's load information.

3.3 The initiation rule

The initiation rule is used to decide the timing to begin load-balancing operations. The invocation decision should ensure the cost overhead against its expected performance benefits [9], [17], [19].

3.4 Load balancing operation

The load balancing process can be described by three rules: the location rule, the distribution rule, and the selection rule. The location rule decides to which node a process selected for transfer should be sent. Load-balancing domains can be either global or local. In case of a global domain, transfer of load may occur from one processor to any of the remaining processors in the system whereas a local domain allows balancing operation only within the neighbor processors. The distribution rule determines the way of distribution of workload among processors in local or global domain. The selection rule determines on the mode of load-balancing operation in preemptive or non preemptive manner. In case of preemptive, the task can be transferred to other processors in the middle of execution whereas in the later, tasks can only be transferred if they are newly created [12],[22].

4 Our approach

Our scenario consists of a set of computers called nodes connected through WAN. From these set of nodes, one node acts as a global scheduler. The global scheduler is also a node in the network to balance the load. It is assumed that initially all the nodes are busy with a certain volume of computations. Upon execution, the proposed New Load Balancing Algorithm (*NLBA*) first collects the percentage of CPU utilization of all the nodes in the network individually in the form of Load Percentage. For deciding the priority of the nodes, the algorithm subtracts the Load percentage of individual nodes from 100 and then divides the result by 10. Based on a 10 point scale, every node is assigned a priority number that varies from 0 to 10. This method takes a maximum priority queue (*Max_PriQ*) based on the priority of the nodes. The queue is initially assumed to be empty. Upon the arrival of a new job at global scheduler, the enqueue operation is performed which inserts the existing priority values of all the nodes into the maximum priority queue one by one. Then the procedure of Extract_Max is performed on the queue to obtain the highest priority numbered node designated as target node. Then the new job is assigned to the target node for smooth operation.

4.1 Algorithmic instance

// This algo runs at global scheduler (which is also a node in the network used to balance load)

```
Algo NLBA ($Node_{set}$ , $n_{gs}$)
Begin
When a new job arrives at global scheduler $n_{gs}$
    Max_PriQ=∅
    For each node $n_d \in Node_{set}$
            Percentage_cpu_util= $n_d$.LoadPercentage
            $n_d$.priority=(100-Percentage_cpu_util)/10
            ENQUEQE($Node_{set}$,Max_PriQ, $n_d$)  //Max_PriQ is
    based on $n_d$.priority
  End For
$n_t$=EXTRACT_MAX($Node_{set}$ ,Max_PriQ)  //  $n_t$ is target node
Submit the job to $n_t$
End

Algo ENQUEUE ($Node_{set}$ , Max_PriQ , $n_d$)
$Node_{set}$ .heapsize= $Node_{set}$ .heapsize+1
Max_PriQ [$Node_{set}$ .heapsize]= -∞
HEAP_INCREASE_KEY (Max_PriQ, $Node_{set}$.heapsize, $n_d$)

Algo HEAP_INCREASE_KEY (Max_PriQ, i, $n_d$)
If $n_d$.priority < Max_PriQ[i].priority
        Error "new priority is smaller than current priority"
Max_PriQ[i]= $n_d$
While i > 1 and Max_PriQ[PARENT(i)].priority <
    Max_PriQ[i].priority
            Exchange Max_PriQ [i] ↔ Max_PriQ[PARENT(i)]
    i= PARENT(i)
End While

Algorithm PARENT(i)
Return $\lfloor i/2 \rfloor$

Algorithm LEFT(i)
Return 2i

Algorithm RIGHT(i)
Return 2i+1

Algo EXTRACT_MAX($Node_{set}$ ,Max_PriQ)
If $Node_{set}$ .heapsize < 1 then
        Error " heap underflow"
Max= Max_PriQ[1]
Max_PriQ[1]=Max_PriQ[$Node_{set}$ .heapsize]
$Node_{set}$ .heapsize= $Node_{set}$ .heapsize-1
MAX_HEAPIFY($Node_{set}$ ,Max_PriQ,1)
Return Max

Algo MAX_HEAPIFY($Node_{set}$ ,Max_PriQ,i)
L=LEFT (i)  ,   R=RIGHT (i)
If L≤$Node_{set}$.heapsize
    &&Max_PriQ[L].priority>Max_PriQ[i].priority
then
        Largest=L
Else
        Largest=i
If R ≤ $Node_{set}$ .heapsize and Max_PriQ[R].priority >
    Max_PriQ[Largest].priority
then
```

```
        Largest=R
If Largest ≠ i then
    Exchange Max_PriQ[i] ↔ Max_PriQ [Largest]
    MAX_HEAPIFY (Node_set, Max_PriQ, Largest)
```

4.2 Working scenario

Our algorithm (*NLBA*) takes input as $Node_{set}$ and n_{gs}. $Node_{set}$ is the set of nodes participating in load balancing process and n_{gs} is one of the nodes acting as global scheduler. When a new job arrives at global scheduler n_{gs}, it determines percentage of CPU utilization, calculates priority, sets priority of that node n_d and calls *ENQUEQE* ($Node_{set}$, *Max_PriQ*, n_d) functionality. *ENQUEQE* operation takes $Node_{set}$, *Max_PriQ* and node n_d as input and calls the method *HEAP_INCREASE_KEY (Max_PriQ,* $Node_{set}$ *.heapsize,* n_d*)* to place the node n_d at its proper position in max priority queue. Finally, it calls *EXTRACT_MAX (*$Node_{set}$*, Max_PriQ)* functionality to extract a node with maximum priority (least CPU utilized) and treats it as target node n_t to which the job is assigned by the global scheduler. *EXTRACT_MAX* operation calls *MAX_HEAPIFY* ($Node_{set}$, *Max_PriQ, 1*) in order to heapify the max heap used to construct the max priority queue. We validate our algorithmic functionality with different setup instances in **section 5**.

5 Evaluation of results

We explore our proposed NLBA instance with some working examples. First, we consider six (6) nodes to validate our set up with different percentage of CPU utilization. We have recorded the utilization of CPU at each node before assigning and after assigning new tasks. The percentage of utilization at each node is shown in **Table 1**. Second, we consider fifteen (15) nodes to further validate our set up with (i) three nodes having same CPU utilization (ii) some other nodes having different CPU utilization. The percentage of utilization of CPU at each and every node is shown in **Table 2.**

Table 1. Utilization of CPU at each node (for 6 nodes)

Node Number	CPU Utilizati-on (%) before arrival of new task	Priority	CPU utilization(%) after assignment of new task
1	91	(100-91)/10=0.9	91
2	86	1.4	86
3	56	4.4	78
4	83	1.7	83
5	80	2.0	80
6	92	0.8	92

From the above table 1, it is observed that **node 3** is least busy among all the nodes and ultimately having highest priority number. At this stage, a new job arrives and it is assigned to **node 3** due to its highest priority.

While evaluating the scenario of Table **2**, before the arrival of new task the algorithm finds that 3 nodes (node 3, 9 and 14) have same percentage CPU utilization, i.e 42%. It implies that these 3 nodes are equally idle (100-42=58%) resulting in equal priority. Then the procedure inserts all the 15 node priorities into the maximum priority queue one by one. The *Extract_Max* operation on a maximum priority queue always gives the output as the highest value present in the queue. This queue contains 15 values among which 3 values are equal (5.8). But since node 3 was inserted prior to node 9 and 14, hence the maximum value comes out to be of node 3 and the newly arrived job is assigned to it.

Table 2. Utilization of CPU at each node (for 15 nodes)

Node Number	CPU Utilization (%) before arrival of new task	Priority	CPU utilization(%) after assignment of new task
1	81	(100-81)/10=1.9	81
2	93	0.7	93
3	42	5.8	73
4	76	2.4	76
5	86	1.4	86
6	94	0.6	94
7	71	2.9	71
8	65	3.5	65
9	42	5.8	42
10	89	1.1	89
11	97	0.3	97
12	78	2.2	78

13	69	3.1	69
14	42	5.8	42
15	84	1.6	84

5.1 Performance evaluation

These experiments aim to quantify the performance of the proposed system from the user's point of view. The job assignment scenario for NLBA with different number of nodes and different CPU utilization aspects are recorded and depicted in **Fig 4, 5, 6, and 7.**

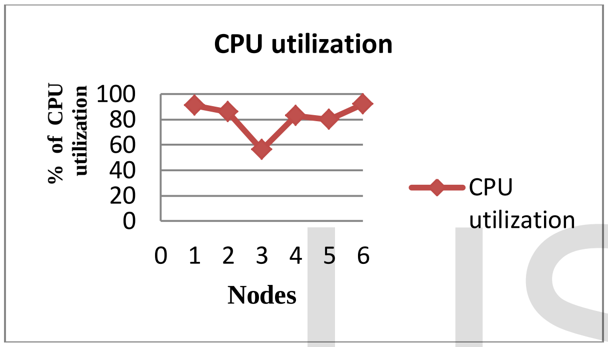


Fig. 4. Processing of load before execution

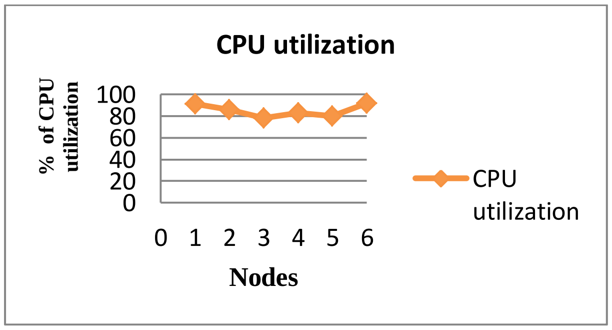
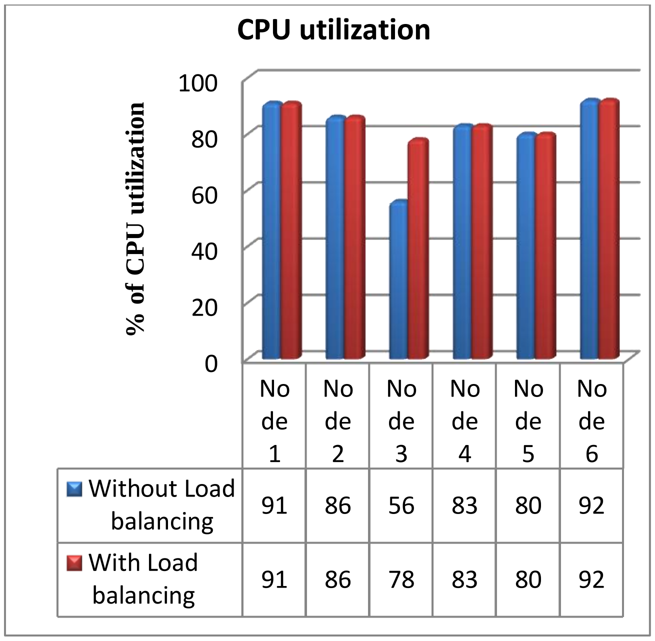


Fig. 5. Processing of load after execution



	Node 1	Node 2	Node 3	Node 4	Node 5	Node 6
🟦 Without Load balancing	91	86	56	83	80	92
🟥 With Load balancing	91	86	78	83	80	92

Fig. 6. Processing of load before and after execution of algorithm

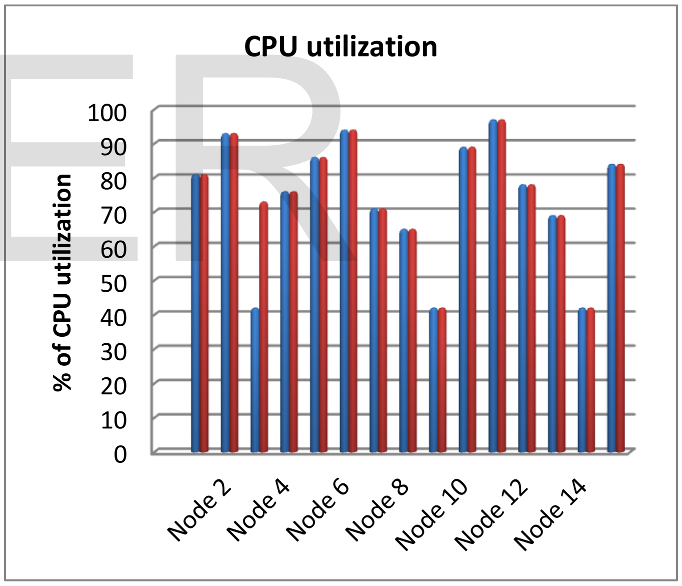


Fig. 7. Processing of load before and after Execution of Algorithm (15 Nodes)

6 Conclusions and future scope

In this paper, we have proposed a new load balancing algorithm (NLBA) for heterogeneous distributed processing systems to balance and process the load among different resources in a distributed environment. This algorithm follows a strategy to reduce the overall processing time of jobs and also improves the throughput of processes. The proposed new load balancing algorithm (NLBA) results in increased performance in terms of balancing and processing the load among resources. This strategy reduces the communication overhead and processing overhead of each user job. We further evaluate and make our algorithm strong enough to find out the response time of each task assigned. We consider another parameter to measure the throughput of entire setup and also make the comparison with existing methodological flows.

References

1. Claudia Rosas,Anna Morajko,Josep Jorba,Eduardo Cesar : "Workload balancing methodology for data intensive applications with divisible load" , 23rd International symposium on computer architecture and high performance Computing (2011) 48-55.
2. Maha A. Metawei, Salma A. Ghoneim, Sahar M. Haggag, Salwa M.Nassar : "Load balancing in distributed multi-agent computing systems" , Ain Shams Engineering Journal,Volume 3, Issue 3(2012) 237-249.
3. Jong-Chen Chen, Guo-Xun Liao, Jr-Sung Hsie, Cheng-Hua Liao : "A study of the contribution made by evolutionary learning on dynamic load-balancing problems in distributed computing systems", Expert Systems with Applications, Volume 34, Issue 1(2008) 357-365.
4. Satish Penmatsa, Anthony T. Chronopoulos : "Game-theoretic static load balancing for distributed systems" ,Journal of Parallel and Distributed Computing, Volume 71, Issue 4 (2011) 537-555.
5. Ioannis Riakiotakis, Florina M. Ciorba,Theodore Andronikos, George Papakonstantinou : Distributed dynamic load balancing for pipelined computations on heterogeneous systems", Parallel Computing,Volume 37, Issues 10–11 (2011), 713-729.
6. Qingqi Long, Jie Lin, Zhixun Sun :"Agent scheduling model for adaptive dynamic load balancing in agent-based distributed simulations ",Simulation Modelling Practice and Theory, Volume 19, Issue 4 (2011) 1021-1034.
7. Ankur Narang, Abhinav Srivastava, R.K. Shyamasundar:"High Performance Adaptive Distributed Scheduling Algorithm",IEEE 27th International Symposium on Parallel &Distributed Processing Workshops and PhD Forum(2013)1725-1734.
8. Jay W.Y. Lim, Poo Kuan Hoong, Eng-Thiam Yeoh : "Neighbor's Load Prediction for Dynamic Load Balancing in a Distributed Computational Environment ", TENCON IEEE Region 10 Conference(2012) 1-6.
9. Ajay Tiwari, Priyesh Kanungo : "A model for dynamic load balancing in open source software for distributed computing environment" ,Software Engineering (CONSEG), IEEE Conference (2012) 1-5.
10. Chhabra, A. , Singh, G.: " Qualitative Parametric Comparison of Load Balancing Algorithms in Distributed Computing Environment", Advanced Computing and Communications, IEEE conference (2006) 58-61
11. Nouri, S. , Parsa, S.: "A Non-cooperative Approach for Load Balancing in Heterogeneous Distributed Computing Platform ", Computer Sciences and Convergence Information Technology, IEEE conference (2009) 756-761.
12. Lim, J.W.Y. , Poo Kuan Hoong ,Eng-Thiam Yeoh: "Heuristic neighbor selection algorithm for decentralized load balancing in clustered heterogeneous computational environment" , IEEE conference (2012) 1215-1219.
13. Simon, M.: "Fault Tolerant Data Acquisition through Dynamic Load Balancing Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW) ", IEEE Symposium(2011) 2049-2052.
14. Huang Jie ,Huang Bei , Huang Qiucen : "A dynamic load balancing algorithm in LAN, Communication Technology (ICCT) ", IEEE Conference (2010) 137-140.
15. Randles, M. , Odat, E. , Lamb, D. , Abu-Rahmeh, O. ,Taleb-Bendiab, A.: "A Comparative Experiment in Distributed Load Balancing" ,Developments in eSystems Engineering (DESE) ,IEEE Conference (2009) 258-265.
16. Di Nitto, E. , Dubois, D. , Mirandola, R. , Saffre, F. ,Tateson, R.: "Self-Aggregation Techniques for Load Balancing in Distributed Systems", Self-Adaptive and Self-Organizing Systems, IEEE Conference (2008) 489-490.
17. Bansal, S.,Hota, C.:"Efficient Algorithm on heterogeneous computing system" , Recent Trends in Information Systems (ReTIS), IEEE Conference (2011) 57-61.
18. Narang, A. ,Srivastava, A. ,Shyamasundar, R.K.: "High Performance Adaptive Distributed Scheduling Algorithm ",Parallel and Distributed Processing Symposium Workshops & PhD Forum ,IEEE conference(2013) 1725-1734.
19. Jain, S. , Singh, H. , Chauhan, A. , Pandey, D. , Chandra, S.: "Heuristics-Aided Load Balancing in Distributed Systems and Node Prioritization: An Intelligent Approach", Computer Modelling and Simulation (UKSim), IEEE Conference (2010) 521-526.
20. Sruthi, R. , Chitra, P. , Poorna, R. , Karpagalakshmi, R. , Swathiya, R.: "Load balanced scheduling of independent tasks in heterogeneous computing systems" , Recent Trends in Information Technology (ICRTIT), IEEE Conference(2011) 628-632.
21. Jianhui Shi , Chunlei Meng , Lingli Ma : "The Strategy of Distributed Load Balancing Based on Hybrid Scheduling , Computational Sciences and Optimization (CSO) ",IEEE conference(2011) 268-271.
22. Pezoa, J.E. ,Dhakal, S. , Hayat, M.M.,:"Decentralized Load Balancing for Improving Reliability in Heterogeneous Distributed Systems" , Parallel Processing Workshops, IEEE Conference (2009) 214-221.