Journal of Computer Science 4 (1): 72-79, 2008
ISSN 1549-3636

Effective Load Metric and Efficient Initial Job Placement for Dynamic Load Balancing in Cluster

[1]P. Sammulal, [1]M. Venu Gopalachari and [2]A. Vinaya Babu
[1]Department of CSE, JNT University, Kakinada, India
[2]Department of CSE, JNT University, Hyderabad, India

Abstract: High performance clusters are being configured specially to give data centers that require extreme performance and the processing power they need. When the data is accessed across clusters the data latency time has significant impact on the performance. In the literature it is given that memory and I/O have become the new bottleneck, instead of processing power in achieving efficient load balance at higher performance for cluster computer systems. Initial job placement and load balancing are the key aspects affecting the performance. The proposed technique combines data access patterns, memory and CPU utilization and locality of memory to consider as load metric in the load balancing aspect across cluster. A scheduling algorithm based on this metric has been proposed to dynamically balance the load in the cluster. Initial job placement for a job in the cluster considers data access patterns and for load balance aspect metric constitutes CPU, memory utilization including locality of memory. Experimental results shown performance improvement to considerable levels with the implementation of the concept, specifically when the cost of data access from other clusters is higher and is proportionate to the amount of data.

Keywords: High Performance Cluster Computing, Load Balancing, Memory Management, Load Metric, Initial Job Placement.

INTRODUCTION

A cluster computer is a collection of computers interconnected with a High-speed network technology. The individual computers can be PCs or workstations. Ideally, a cluster works as an integrated computing resource and has a single system image spanning all its nodes. Hence, the users see only a single system. User processes can be executed on any node of the cluster. A cluster can be used for scientific applications that need supercomputing power and in domains such as databases, web service and multimedia, which have diverse QualityofService (QoS) demands. In addition, users can access any node within the cluster and run different types of applications simultaneously. The main goals are to minimize the total response time and maximize throughput.

However, a cluster system has the tendency to concentrate the system load on to certain nodes, resulting in coexistence of overloaded nodes and idle resources[3]. Therefore, the development of a load balancing system for utilizing computing resources of lightly loaded nodes is crucial to resolving the problem of load imbalance in the cluster system. Dynamic load balancing systems can be classified into initial job placement and process migration. An initial job placement system traces the node that best meets the task requirements before the execution[10]. A system based on process migration, however, functions by transferring tasks from an excessively loaded node to another node when a load imbalance occurs[8]. Employing either initial job placement or process migration alone is not as efficient as exploiting both of these methods simultaneously[5]. Initial job placement improves the resource utilization of the entire system by distributing the workload on to several nodes. However, we can expect further improvement in performance if the initial job placement system enhances resource utilization not only system-wide, but also in terms of each node[4]. Therefore, it is necessary for the initial job placement to consider the resource requirement of the job to be assigned. Many studies have been conducted on prediction of job resource requirement before starting execution; these include, estimation of the future behavior of a job resource requirement by the historical data[9], a statistical approach[4], providing user estimation about a job resource requirement to the load balancing system[7], and estimation by the process behavior during the initial one-second execution[6]. However, these approaches are likely to incur mistakes

Corresponding Author: P. Sammulal, Department of CSE, JNT University, Kakinada, India

because the resource usage is limited to the information provided in terms of estimation. Furthermore, these approaches can severely affect the execution time when using an inaccurate estimation. Scheduling is a challenging task in this context. The data intensive nature of individual jobs means it can be important to take data location into account when determining job placement. Despite the other factors which contribute performance in a cluster computing environment, optimizing memory management can improve, the overall performance.

Memory management becomes a prerequisite when handling applications that require immense volume of data for e.g., satellite images used for remote sensing, defense purposes and scientific applications. Here even if the other factors perform to the maximum possible levels and if memory management is not properly handled the performance will have a proportional degradation. Hence it is critical to have a fine memory management technique deployed to handle the stated scenarios. To address this problem, we have defined a combined memory management technique.

The proposed technique focuses on optimizing memory usage, assuming the other factors which contribute to performance are performing to the optimum level. Initial job placement in the cluster considers data access patterns to designate a node for a job. For this purpose, we have developed a new algorithm and a new load metric which contains information about both the system load and resource utilization. The parameters considered are queue length, instances of CPU and memory utilization, number of page faults. If any node failure is found in the middle, then those processes get high priory to migrate to light loaded nodes. A dynamic load balancing algorithm is designed and implemented using the load metric and its performance is evaluated.

RELATED WORK

The control of a cluster can be centralized and distributed. In a centralized cluster, all users interact with the cluster through a central node. The other nodes are processing nodes. User processes are allocated to processing nodes by the central node. The central node collects system state information and makes all scheduling decisions. In a distributed cluster, a user can connect directly to any one of the cluster nodes. There is no master node. Each node is considered a local controller. They run asynchronously and concurrently to each other. Each node is responsible for making scheduling decisions for the processes submitted by its users and for accepting remote processes.

Paul werstien and *et al*. proposed a dynamic load balancing algorithm which is decentralized to avoid bottlenecks and single point of failure, considered CPU and memory utilization and as load metric in addition with CPU queue length[1]. The experimentation results with proposed algorithm had shown better results than traditional one that considers only queue length as metric.

Sammulal *et al*. proposed an algorithm which assigns a cluster for an incoming job[2]. Here authors used data access patterns to decide the node to designate. And the simulation results shown better performance than using data availability for the node selection.

Min Choi *et al*. proposed a new load metric termed as number of effective tasks in order to solve the problem arising from inaccurate predictions[11]. The proposed algorithm designates a node for a job using this metric. The simulation results had shown better performance than history based algorithm.

Nayeem Islam *et al*. proposed a new resource management system, Octopus, which supports extensibility as well as fault tolerant. It contains mainly two components, hierarchical software architecture and flexible dynamic partitioning, but didn't focus on load balance aspect which differentiates from our work.

Shirazi *et al*.[12] summarized two general location policies to select the destination node to transfer the load. The node selected should be lightly loaded and have the correct environment to run the process.

- **Minimum load:** Select a node with the minimum current load
- **Low load:** Select the first node whose load is below some threshold value. This policy is applied to a transfer policy based on thresholds. There is a possible problem of several heavily loaded nodes transferring their processes to a lightly loaded node, causing it to become heavily loaded. A simple solution is to randomly select one of the lightly loaded nodes for transfer

Although many schemes exist, the policies should be decided according to the desired environment, such as application types or cluster environment. It is very difficult to say which algorithms are most efficient. There is no single algorithm which is optimal for all purposes. We can only find a best solution for a particular situation. In most clusters, processes will arrive randomly, and it is difficult to know their characteristics such as execution time. We can only take into account the current states of the nodes such as CPU utilization and CPU queue length.

Radha *et al*., projected a predictive and prefetching method to utilize remote memory in the grid[13]. The Remote memory paging could be a potential option in the presence of memory pressure due to the following facts: Internet has made almost all machines part of network, existence of idle memory in the machines in the network.

PROPOSED LOAD BALANCING TECHNIQUE

Initial job placement based on data access patterns: The scheduler after the reception of a new request makes an analysis to identify a particular node to which the request can be forwarded. The scheduler primarily takes in to consideration the load of the processors of the nodes of the concerned cluster before the task is assigned. But this process of designating nodes for processing tasks would not yield optimum performance because bandwidth is also a major factor in determining the performance levels. So to overwhelm this problem we have proposed a new algorithm using global memory and local memory.

The conventional scheduling algorithm blindly fixes a particular node taking into account the availability of data the as the sole criterion. This method of designating a particular node for a request would lead to performance degradation. To illustrate the above scenario let us consider a particular request requires certain the cluster that is identified for the given request is based on the presence of major portion of required data and the cost for accessing remaining data is not considered and if it is significantly higher, then it has to be treated in a separately[2].

At the same time, if the task is designated to a node irrespective of the percentage of data present in that node and considering the cost of accessing the remaining data from the rest of the clusters through global memory the performance can be optimized further.

Assumptions:

N_C → Total number of clusters
Cji → The cluster handling the current job i.
S F → Set of files requires for the file job (I)
SN_{WC} → Set of nodes having the SF_{WC} in the Mg within Cluster CJi
SS_C → Set of clusters having SF_{Mg}
SF WC → Is a set of files available in CJi
SF Mg → Is a set of files to be transferred from SS_C Through Mg

For Files within a Cluster
for each files in SF WC
for each node in SN WC
t = Calculate time
end
t_{min} = min (t)
Update SQN_{WC}
End

$$T_{WC} = \sum_{i=0}^{\text{sizeof}(SF_{WC})} t_{min}$$

For Files between Clusters

for each files in Mg SF
for each cluster in SS_C
t = Calculate time to transfer file from SS Ci through Mg
end
t_{min} = min (t)
Update S qc
End

$$T_{BC} = \sum_{i=0}^{\text{sizeof}(SF_{qc})} t_{min}$$

$$T = T_{WC} + T_{BC}$$

Repeat the above steps for all the clusters
$S_T = (T0, T1, T2 \ldots T_{NC})$
$T_Q = \min (S_T)$.

Corresponding node is chosen to allot the job as shown in Fig. 1.

Load balancing system based on proposed load metric including memory locality: Ideally, the load information should reflect the current CPU utilization, memory utilization and memory locality of a node. Traditionally, the load of a node at given time was described simply by CPU queue length. CPU queue length refers to the number of processes which are either executing or waiting to be executed. The processes which are waiting for other system resources are not included. So the CPU queue length does not reflect directly and memory utilization. In the proposed algorithm, CPU utilization, CPU queue length, and memory utilization considering memory locality are used. The system statistics such as CPU utilization, CPU queue length of a node changes during the life of processes. For example, the CPU utilization may be high in one second but low in the next second.

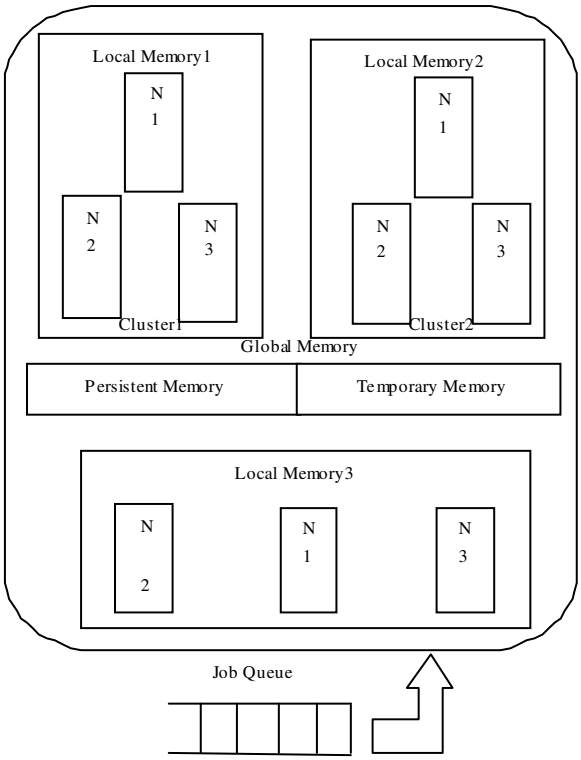

Fig. 1: Schematic diagram of designating a node for a job at initial job placement

Therefore it is reasonable to average these statistics over several seconds. Also number of page faults is considered to measure the memory locality for each process in every node. In the proposed algorithm, 5 seconds is set for the averaging interval. CPU utilization (CPUu), CPU queue length (Nop), memory utilization (mem_u), number of page faults (npf) are considered as load information parameters to measure load of a node.

The following equation is used to calculate each metric.

$$Load_i(par) = \frac{v1 + v2 + ... + vt}{t}$$

Where

$Load_I \rightarrow$ The average load metric of the specified parameter over the previous t seconds for a particular node.

Par $\rightarrow$ The information parameter of load. (Par is Nop, CPUu, or mem_u).

$V_I \rightarrow$ The value of a given parameter in a previous one second interval.

T $\rightarrow$ The number of time intervals. t is set to 5 for this research.

The averaged information including CPU, memory utilization is the load metrics used to describe the load on a node. And CPU queue length, number of page faults of each node are considered to measure the memory locality based on which the process migration is performed. The information exchange policy chosen for this research is a periodic policy with a time interval of one second.

The second part of load classification is to group the nodes into one of four classes. Using the threshold values of each parameter, the nodes will be grouped as idle, low, normal or high according to the following criteria. For each node, the CPU utilization, CPU queue length, memory utilization will be checked to decide whether it is in idle, high, low or normal level.

$$Load = \begin{cases} Idle & CPU \leq 1\% \\ High & (mem_u > 85\%) \text{or} \\ & \begin{pmatrix} CPU_u \text{is High} \\ \text{and} \\ mem_u \text{is high} \end{pmatrix} \\ & \text{or (Nop is High)} \\ Low & \begin{pmatrix} CPU_u \text{ is low} \\ \text{and} \\ mem_u \text{is low} \end{pmatrix} \\ & \text{or (Nop is low)} \\ Normal & Otherwise \end{cases}$$

After the load of each node has been classified, the next step of the process transfer policy is to decide if a newly arriving process should be run locally or on some other node.

The following pseudo code defines how this decision is made:

```
IF the local host is idle THEN
Run locally
  ELSE IF there is idle nodes THEN
Run on an idle node
    ELSE IF the local host is high loaded AND
There are low loaded nodes THEN
Select the node N_sel from which majority of page
Faults served
Run on the node N_sel
      ELSE
Run locally
ENDIF
```

This pseudo code gives preference to running a process locally if the local node is idle. The next choice

is any other idle node. The next choice is a node with a low load level if the local node is highly loaded. The final part is to migrate the process to make its locality of memory maximum. The number of page faults is calculated on high loaded nodes and the node will be selected by which majority of page faults are served to migrate the process. Thereafter if the selected node is high loaded the process will b run on the local host itself. If no node can be found in the previous choices, the process is assigned to the local host.

RESULTS AND DISCUSSION

The proposed initial job placement algorithm is based on data access patterns and load balancing algorithm is based on CPU utilization, CPU queue length, memory utilization. It is compared to the traditional CPU queue length based policy. This allows a comparison between the two load estimation policies.

The performance tests use a variety of different types of applications: CPU bound, memory bound, and mixed applications. All nodes in the cluster are homogeneous and have the same hardware and operating system. For the simulation results ,we evaluate the performance of the proposed dynamic load balancing system with the with proposed new load metric. We used a Pentium Dual-Core 3.2 GHz machine with 2 GB RAM as a global job scheduler with 8 computation nodes. Each node is a Pentium IV 2.8 GHz machine with 512MB RAM. The network is switched 100 Mbps Ethernet. Here the assumption is zero network latency for transfer load information among nodes in the cluster.

The experiments are done based on a variety of applications. These applications are meant to simulate what might occur in a cluster that is used by a computer science laboratory, for example. The tests are not meant for simulating parallel programming applications. These applications include two types: CPU-bound process and Memory-bound process. CPU-bound process is the program that computes a mathematical expression recursively. When run, the CPU utilization is about 100%. And Memory-bound process is the programs that is to simulate a memory-bound process uses the malloc() function to allocate 600 Mbytes or 700 Mbytes of memory. Then the processes gradually load the memory. Since the nodes only have 512 Mbytes of physical memory, the memory is exhausted, and virtual memory software has to move pages to swap space on a local disk.

Finally there are Mixed processes that include one program recursively reads a small file every second. This simulates a process with low CPU utilization. The average CPU utilization is about 3%. The terms low and high with respect to utilization are relative. That is, there is no absolute value that is considered low or high.

The tests consist of two parts:

- **Workload:** The workload of the tests includes a batch of programs which simulates a user's work. These programs are chosen from the above pool of programs as needed. The programs are randomly chosen each time. Between two programs, there is a random several seconds sleep time to simulate a user's thinking time.
- **Background programs:** A series of background programs are used to simulate different loadings of the nodes, such as some nodes with low CPU utilization, some nodes with low CPU utilization and high memory utilization, some nodes with high CPU utilization, and some nodes with high memory utilization. The background programs and random workload processes can make a node's loading random. In this situation, the proposed load balancing algorithm can make each process in the workload choose the proper node from the different loadings of the nodes.

Idle nodes based on CPU and memory utilization: As previously discussed, CPU queue length might not reflect correctly whether a node is idle. For some processes, a node may have a low CPU utilization but have high memory utilization. For these kinds of processes, the process is normally not on the CPU queue when collecting information every second. The CPU queue length based load estimation policy will determine the nodes running these kinds of processes as idle nodes, although the node has high memory utilization. In addition, the type of each new process is unknown. Ideally if there are idle nodes, the new process should run on an idle node. If an idle node cannot be detected correctly, the performance will be degraded. For example, if a new process needs high memory size, and a node with high memory utilization but zero CPU queue length is chosen as an idle node, the performance will be greatly reduced.

In the first part of evaluation a test is conducted to see whether an incoming process can be allocated to an idle node correctly. For this purpose, three test programs having different characteristics are executed on the cluster where CPU queue length for all these cases is assumed as zero. The three Cases are: CPU utilization is zero, low CPU - low memory utilization and low CPU – high memory utilization and forcing memory paging to the hard disk.

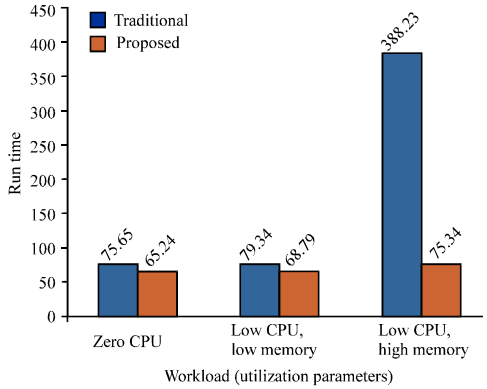

Fig. 2: Run times of different programs on the proposed and traditional systems

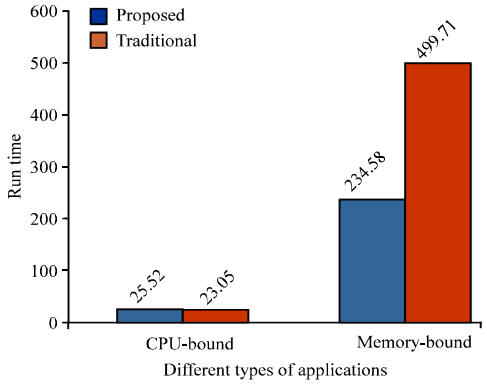

Fig. 3: Average run time for the test about different types of applications

The result is the average run time of the workload being presented to each of the nodes (Fig. 2). For the proposed load balancing algorithm, different nodes may be chosen as target nodes for each new process. In fact, for this algorithm, the node with zero CPU utilization is the only idle node, and it is always chosen. When using node with second case characteristics, there is a small overhead associated with remotely starting the new processes on first node. When using node with high memory and low CPU utilization, there is a slightly larger overhead waiting on the virtual memory system to allocate pages so the new processes can be remotely started on second node. The CPU queue length based model always chooses the local node to run since the CPU queue length is zero. When using this model, first case is truly idle and gives the best overall time. Second case is lightly loaded and takes a longer time due to its background processes. Third case takes a very long time because the virtual memory system is forced into paging to continue running the background processes and the new process. Thus the proposed load balancing algorithm performs better than a CPU queue length based algorithm in detecting truly idle nodes.

CPU-bound and memory-bound processes: In the second part of the evaluation, the assumption is the types of new processes are known (CPU-bound and memory-bound). The test is to explore the effect of the load estimation policy on different types of applications.

The methodology is to run a similar set of programs on various nodes with different loading characteristics and compare the performance of the two load balancing algorithms. The background processes are similar to the previous test. The difference is the addition of another node with another new characteristic, that is high CPU utilization and low memory utilization

There are two groups of workloads, one is CPU-bound workload, constitutes of 25 arithmetic computing processes. Another one is Memory-bound workload, consists of 25 processes which need a large amount of memory. Both workloads execute the programs independently and run one by one either locally or remotely.

Initially both groups are started with an interval of one second on all 4 nodes having different characteristics which are described above. Thereafter, based on the random wait between each new process on each node, the remaining execution performed according to the proposed algorithm. First test is to run CPU-bound processes and second test is to run memory–bound processes on all nodes at the same time. These tests run twice and the averages run times are considered as results.

The results shown that the performance of proposed scheme for only CPU-bound processes is about 11 % worse than CPU queue length based model. Whereas the performance for only memory-bound processes the proposed scheme is about 55 % better than the traditional one Fig. 3. The reason for this is that CPUbound processes mainly consume CPU time. The best node should be the node with the lowest CPU utilization regardless of whether there is high memory utilization. The CPU queue length based algorithm fits this workload type easily. That is, it will treat all nodes with low CPU utilization and zero CPU queue length as idle nodes. While the proposed model tries to choose the node with low CPU utilization and low memory utilization. In addition, the node with high memory utilization is considered a highly loaded node. Thus the number of selectable nodes is lower and more processes have to run locally.

In the second case, CPU queue-length model will not consider memory utilization to decide whether a node is idle. When the physical memory of a node is exhausted, it needs to page. In this situation, a new memory-bound process running on this node makes the performance significantly worse. Therefore memory utilization is worth considering in a load balancing algorithm.

Mixed types of applications: The strategy is the same as Section 5.2. But the workload on each node is mixed. The workload consists of 25 programs. Each program in the workload is randomly selected from the pool of programs. Thus we do not know the type of each new process a priori. The type of program is decided at run time. These tests run on the nodes with the described characteristics in the above section. Each test is run two times with a different random seed each time.

The cluster performance with the different algorithms is shown by the total run time of all programs on all cluster nodes. The maximum differential of the run time of each node can indicate the balance of the load on each node. According to the tests, the result shows that the overall performance of the proposed algorithm is about 50.5% better than the CPU queue-length based algorithm. The maximum differential of the run time of proposed algorithm is better than the CPU queue-length based algorithm. This indicates the proposed algorithm more effectively uses the cluster than the CPU queue-length based algorithm. When there is a reasonable amount of memory utilization, the proposed algorithm shows better performance.

CONCLUSION

In this study an efficient load balancing system has been developed with a new load metric that considers the CPU and memory utilization and CPU queue length. Also we developed a new initial job placement algorithm that designates a node for a job considering data access patterns as key issue which can perform efficiently than existing ones. Most of the cluster applications are memory bound, so without considering memory utilization will pose the performance degradation. And using number of page faults as parameter to represent memory locality for efficient process migration from heavily loaded to low loaded node will show the optimum performance.

A number of tests were performed on different scenarios and from these results we can conclude that the combination of the proposed initial job placement algorithm and the proposed load balancing algorithm exhibits better performance than traditional schemes that uses CPU queue length as the load metrics.

REFERENCES

1. Werstein, P., H. Situ and Z. Huang, Load balancing in a cluster computer. Proceedings of the 7th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT'06). DOI Bookmark: 10.1109/PDCAT.2006.77-22k
2. Sammulal, P. and A. Vinaya Babu, 2008. Efficient and collective global local memory management for high performance cluster computing. IJCSNS Int. J. Comput. Sci. Network Secur., 8 (4). http: //paper.ijcsns.org/07_book/200804/20080412.pdf
3. Suzuki, M., H. Kobayashi, N. Yamasaki and Y. Anzai, 2003. A task migration scheme for high performance real-time cluster system. 19th International Conference on Computers and Their Applications, pp: 228-231. http://doi.ieeecomputer society.org/10.1109/CLUSTR.2003.1253344.
4. Bubendorfer, K., 1996. Resource based policies for load distribution. Master's Thesis in Victoria University of Wellington, 1996. http://homepage**s**. mc**s**.vuw.ac.nz/~**k**ri**s**/ http://homepage**s**.mc**s**.vuw. ac.nz/~hine/jh/Paper**s**/**load**Balancing/uniform97.p **s**.gz.
5. HaMahmoud, Y., P. Sens and B. Folliot, 1999. Quantifying the performance improvement of migration in load sharing systems. International Conference on Parallel and Distributed Processing Techniques and Applications, pp: 241-299. doi: 10. 1.1.41.5831 http://citeseerx.ist.psu.edu/viewdoc/ summary?doi=10.1.1.41.5831.
6. Hotovy, S., D. Schneider and T. O'Donnell, 1996. Analysis of the early workload on the cornell theory center ibm sp2. ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, pp: 272-273. http://doi.acm.org /10.1145/233008.233060.
7. Krueger, P. and R. Chawla, 1991. The stealth distributed scheduler. 11th IEEE International Conference on Distributed Computing Systems, pp: 336-343. doi: 10.1109/ICDCS.1991.148686
8. Milojicic, D., F. Douglis, Y. Paindaveine, R. Wheeler and S. Zhou, 2000. Process migration. ACMComputing Surveys, pages 241-299. http:// doi. acm.org/10.1145/367701.367728.
9. Pasquale, J., B. Bittel and D. Kraiman, 1991 A static and dynamic workload characterization study of the san diego supercomputer center cray x-mp. ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, pp: 218-219. http://doi.acm.org/10.1145/107972.107998

10. Shivaratri, N., P. Krueger and M. Singhal, 1992. Load distributing for locally distributed systems. IEEE Computer, pp: 33-44. DOI Book Mark: 10.1109/2.179115
11. Min Chi, Jung-Lok Yu, Ho-Joong Kim and Seung-Ryoul Maeng, 2003. Improving performance of a dynamic load balancing system by using number of effective tasks. IEEE International Conference on Cluster Computing Proceedings 2003. pp: 436-441 ISBN: 0-7695-2066-9 http://ieeexplore.ieee.org/xpl /freeabs_all.jsp?arnumber=1253
12. Shirazi, B.A., A.R. Hurson and K.M. Kavi, 1995. Scheduling and load balancing in parallel and distributed systems. IEEE Computer Society Press, Los Alamitos, California, http://portal.acm.org/citation.cfm?id=583069&dl GUIDE,ACM&coll=GUIDE&CFID=33653278&CFTOKEN=23960045#
13. Radha, S., S. MarySairaBhanu, N.P. Gopalan, Remote memory management and prefetching techniques for jobs in grid. Proceedings of the Second International Conference on Semantics, Knowledge, and Grid (SKG'06). DOI: 10.1109/ SKG.2006.73.