Volume 3, Issue 12, December 2013 ISSN: 2277 128X

International Journal of Advanced Research in Computer Science and Software Engineering

Research Paper

Available online at: www.ijarcsse.com

Process Division and Migration based Load Balancing for Distributed Operating System

[1]Jayna Donga, Assistant Professor, MBICT College, Gujarat, India

[2]Vatsal Shah,[3]Kanu Patel, Assistant Professor, B.V.M. Engg. College, Gujarat, India

[4]Rahul Goradia Assistant Professor, G.H. Patel Engg. College, Gujarat India

Abstract— We are observing that load is continuously increasing in present scenario in distributed operating system. That's why we need to balance the load in distributed operating system. There are several techniques to balance the load. But in this paper we have used process migration technique to balance the load. This paper includes comparison between some load balancing algorithms as well as proposed algorithm to balance the load in distributed operating system. In this paper we have tested for both type of process non preemptive as well as preemptive. Here, we have also considered cpu_bound as well as memory_bound process. To measure the load of the system we have taken two parameters in consideration cpu_utilization and memory_utilization. Here, homogeneous environment is taken in consideration.

Keywords— CPU utilization, Memory utilization, cpu_bound, memory_bound, preemptive, non preemptive

I. INTRODUCTION

To understand Load balancing, it is necessary to understand load. Load may be define as number of tasks are running in queue, CPU utilization, load average, I/O utilization, amount of free CPU time/memory, etc., or any combination of the above indicators. Load balancing can be done among interconnected workstations in a network or among individual processors in a parallel machine. Load balancing is nothing but the allocation of tasks or jobs to processors to increase overall processor utilization and throughput.

Actually load balancing is done by process migration. But to balance the load it is necessary to measure the load of individual node in network or in a distributed environment. For calculating node above mentioned factor in a definition of load are calculated. After calculating the node of individual, mark the underloaded/free and overloaded/busy node.[1,4]

Now to balancing load transfer the process from overloaded node to underloaded node. In this way load can be balance in network of work station or in a distributed environment.

Process migration is the transfer of process from one node to another node in network of workstations or nodes. It is very useful mechanism for balancing the load on distributed system. Load balancing in a distributed system can be done through transferring a process form heavily loaded node to lightly loaded node.

There are two types of process migration. (1) Preemptive Process Migration (2) Non-preemptive Process migration. Preemptive process transfers [1,4] involve the transfer of a process that is partially executed. This transfer is an expensive operation as the collection of a process's state (which can be quite large and complex) can be difficult. Typically, a process state consists of a virtual memory image, a process control block, unread I/O buffers and messages, file pointers, timers that have been set, etc. Non-preemptive process transfers [4], on the other hand, involve the transfer of processes that have not begun execution and hence do not require the transfer of the process's state. In both types of transfers, information about the environment in which the process will execute must be transferred to the receiving node.

II. LOAD BALANCING ALGORITHM

A. *Three types of Algorithm [1,4]*

1) Sender-Initiated Algorithm
2) Receiver-initiated Algorithm
3) Symmetrically Initiated Algorithm

B. *Components of Algorithm [1,4]:*

Typically, a scheduler has four components:

- A *transfer policy* [3] that determines whether a node is in a suitable state to participate in as process transfer
- A *selection policy* [3] that determines which process should be transferred
- A *location policy* [3] that determines to which node a process selected for transfer should be sent
- An *information policy* [3], which is responsible for triggering the collection of system state information.

III. WORK DONE

As it has been previously described that there are many algorithm for load balancing in distributed operating system. Each has its merits and demerits. We have surveyed a few traditional algorithms and also proposed a new algorithm. We have actually implemented all algorithms including our referred algorithm. We have checked it for non-preemptive and preemptive process migration.

Performance analysis of different algorithm in which process execution,

On distributed system according to

1) CPU-utilization with Non-preemptive migration
2) Memory-utilization with Non-preemptive migration
3) CPU-utilization with preemptive migration
4) Referred algorithm with Non-preemptive migration
5) Referred algorithm with preemptive migration
6) Propose algorithm with non-preemptive migration

In this research work actually we will compare referred algorithm with our proposed algorithm. In referred algorithm they have just taken decision of migration of the whole process on another node in the cluster system on the basis of CPU-utilization or Memory utilization. So in our referred algorithm, we have taken care of both parameter as well as the concept of process division and migration. In such a case where, the process is having independent tasks, in that situation we can divide process into independent tasks and then migrate these tasks on different lightly loaded nodes into the distributed system .So, in this way we can take benefit of many lightly loaded nodes in the distributed system and can decrease the execution time of the process. The goal of this research is to design an effective load balancing algorithm for a distributed system. A homogeneous environment is assumed. The individual computers are PCs having a single CPU and their own memory. They are interconnected through switched Ethernet. Each node runs the Linux operating system. Users can directly log into any node in the cluster. The use of cluster is totally distributed. The algorithm is centralized, but to avoid bottlenecks and a single point of failure we have kept its back up on another server. It considers CPU utilization, and memory usage. These data are exchanged periodically between the cluster nodes to server.

Ideally, the load information should reflect the current CPU utilization and memory utilization of a node. The system statistics such as CPU utilization and memory utilization 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. Therefore it is reasonable to average these statistics over several seconds. In the referred algorithm, 5 seconds is set for the averaging interval. CPU utilization (cpu u) and memory utilization (mem u) are considered as load information parameters to measure load of a node. The following equation is used to calculate each metric. [1,3]

$$l_n\,(par) = \frac{p_1 + p_2 + p_3 + \ldots + p_t}{t} \ldots\ldots\ldots\ldots\ldots\ldots\ldots (1)$$

where

- l_n is the average load metric of the specified parameter over the previous t seconds for a particular node.
- *par* is the information parameter of load. (*par* is either cpu u, mem u).
- $p_1 \cdots p_t$ is the value of a given parameter in a previous one second interval.
- *t* is the number of time intervals. *t* is set to 5 for this research.
- *n* is the number of a given node.

The first step in the process transfer determination is to classify the load at each of the nodes. The proposed algorithm uses four levels of load: **idle, low, normal** and **high**.

The first part of this step involves calculating two threshold values for the CPU utilization (cpu u) and memory utilization (mem u). Two thresholds are used because they give a more stable load balancing solution.

The calculation of the threshold for these two parameters is done as follow:

1. **Calculate load average of each parameter (nr, cpu u, and mem u) over all nodes. The equation is: [1,3]**

$$L_{avg}\,(par) = \frac{l_1 + l_2 + l_3 + \ldots + l_n}{n} \ldots\ldots\ldots\ldots\ldots\ldots\ldots (2)$$

where

- L_{avg} is the average load of a given parameter over all nodes.
- *par* is the parameter of load: nr, cpu u, and mem u.
- $l_1, \cdots, l_n$ are the current load of the parameter of each node derived by load estimation policy.
- *n* is the number of nodes.

Each of the l_i is the five second average of the desired parameter. See Equation 1.

2. **Calculate the threshold values**

The upper and lower threshold values of CPU utilization and memory utilization are calculated by multiplying the average load of each parameter and a constant value. [1,3]

$$t_H = H \times L_{avg}$$
$$t_L = L \times L_{avg}$$

where, t_H is the high threshold, t_L is the low threshold, H and L are constants. H is greater than one and L is less than one. In the referred algorithm, H and L are set to 1.3 and 0.7, respectively. That means when a certain load parameter is 30% above the average load, it is highly loaded. When a certain load parameter is 70% of the average load, it is lowly loaded; otherwise, it is normally loaded. This calculation and classification of parameters is done for the CPU utilization and memory utilization.

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 and memory utilization will be checked to decide whether it is in idle, high, low or normal level.

load =

- Idle: cpu u < 1%
- High: (mem u is high) or (cpu u is high)
- Low: (cpu u is low) or (mem u is low)
- Normal: otherwise

TABLE I
REFERRED ALGORITHM [1]

```
if ( host = = idle )
        {
                run = host ;
        }
else if (host == low || host == normal || host == heavy)
{
        if( search_idle() )
        {
        run = idle ;
        }
        else if ( search_low() )
        {
                if ( host != low )
                {
                run = low ;
                }
        else if ( search_normal() )
        {
                if ( host != normal )
                {
                run = normal ;
                }
        else
        {
        run = host ;
        }
}
```

TABLE II
PROPOSED ALGORITHM

```
If (host != idle)
{
        If(host == low)
          {       n = count_idle();
                  if ( n > 0 )
                  {
                          Divide process in "n" parts and run it on n idle node
                }
         }
          Else If (host == normal)
          {       n1 = count_idle( );
                  n2 = count_low( );
```



```
                    if ( n1 > 0 || n2 > 0 )
                    {
                            Divide process in "n1 + n2" parts and run it on n1 and n2
node
                    }
            }
            Else If (host == heavy )
            {       n1 = count_idle( );
                    n2 = count_low( );
                    n3 = count_normal( );
                    if ( n1 > 0 || n2 > 0 || n3 > 0)
                    {
                            Divide process in "n1 + n2+n3" parts and run it on n1, n2 and
            n3 node
                    }
            }
            Else
            {
                    Run on local host
            }
}
```

IV. EXPERIMENTAL RESULTS AND PERFORMANCE EVALUTION

In this research work actually we have compared our proposed algorithm with traditional parameter based algorithm as well as with referred algorithm. In traditional algorithm decision of migration of the process on the basis of CPU-utilization or Memory utilization, but it may happen that process can be of any type and in referred algorithm researchers have taken care of both parameter. As well as in referred algorithm basically used preemptive migration, so cost of running process will also decrease. But in our proposed algorithm actually we are dividing our process in independent tasks and then executing them on to different node of distributed system in parallel. Our evolution is based on execution time of a different CPU-bound process and memory-bound process on different loaded machine in distributed environment. For preemptive migration we have used BLCR checkpoint utility and for message passing we have used socket programming. In our proposed algorithm to divide a process we have used MPI programming (Message passing Interface).

Benefit of referred algorithm over the traditional algorithm is that it uses two threshold policy as well as the concept of idle node, so that process can automatically migrate on idle node and get the advantage. And one more benefit is that it always sort the node according to it's CPU utilization and memory utilization while in traditional algorithm it polls the node randomly and only used two-threshold policy.

Benefit of proposed algorithm over the referred algorithm is that it uses all available lightly loaded in surrounding and execute the task in parallel while referred algorithm only used single node from the distributed environment and migrate the process on it. So obviously, proposed algorithm works better than referred algorithm as it works in parallel.

A. System Configuration

Fadora 15 kernel 3.6.32-21 generic
Memory: 993.0 MB
Dual core: P0: 2.40 GHz, P1: 2.40 GHz

B. RESULTS

TABLE III
FOR CPU BOUND PROCESS

Node's Status	CPU-utilization
Idle	< 1%
Low	1-35 %
Normal	35-65%
Heavy	>65%

TABLE IV
FOR MEMORY BOUND PROCESS

Node's Status	Memory-utilization
Idle	<= 45 %
Low	45-60%
Normal	60-75%
Heavy	> 75%

TABLE V
PROCESS EXECUTION TIME FOR CPU BOUND PROCESS ON STANDALONE MACHINE WITHOUT ANY ALGORITHM

Node's status	Time(seconds)
Idle	27.423171
Low	33.762435
Normal	39.524131
Heavy	46.097726

TABLE VI
PROCESS EXECUTION TIME FOR MEMORY BOUND PROCESS ON STANDALONE MACHINE WITHOUT ANY ALGORITHM

Node's status	Time(μs)
Idle	449735
Low	748276
Normal	1452398
Heavy	Killed

TABLE VII
SENDER INITIATED ALGORITHM FOR CPU BOUND PROCESS BY NORMAL SENDER

Node's Status	CPU utilization	Memory utilization	Referred algorithm	Proposed algorithm
Idle	0	0	27.983456	14.983456
Low	35.182789	37.028367	32.382145	18.382145
Normal	39.371479	39.788001	39.818136	39.818136
Heavy	0	0	0	0

TABLE VIII
SENDER INITIATED ALGORITHM FOR CPU BOUND PROCESS BY HEAVY SENDER

Node's Status	CPU utilization	Memory utilization	Referred algorithm	Proposed algorithm
Idle	0	0	27.931409	14.234167
Low	35.178287	37.627311	32.624176	18.526271
Normal	39.371479	42.873541	35.937421	21.723456
Heavy	46.291997	46.725934	46.036871	46.972365

TABLE IX
SENDER INITIATED ALGORITHM FOR MEMORY BOUND PROCESS BY NORMAL SENDER

Node's Status	CPU utilization	Memory utilization	Referred algorithm	Proposed algorithm
Idle	0	0	458724	239465
Low	3131643	792321	733452	396676
Normal	3219452	1446716	1441769	1449834
Heavy	0	0	0	0

TABLE X
SENDER INITIATED ALGORITHM FOR MEMORY BOUND PROCESS BY HEAVY SENDER

Node's Status	CPU utilization	Memory utilization	Referred algorithm	Proposed algorithm
Idle	0	0	605976	322654
Low	3281929	898927	775423	408428
Normal	3399476	1754326	1618391	830982
Heavy	Killed	Killed	Killed	Killed

V. CONCLUSION

In this research we have tried to proposed algorithm against some previous algorithm for load-balancing in distributed operating system with preemptive and non-preemptive migration of process, and also it has been compared with available traditional algorithm based on parameter like CPU utilization and memory utilization of load-balancing in distributed operating system. It shows that proposed algorithm is efficient than traditional as well as referred algorithm for different type of application like CPU-bound process and memory bound process.

REFERENCES

[1] Vatsal Shah and Kanu Patel, "Load balancing algorithm by process migration in distributed operating system" in IRACST - International Journal of Computer Science and Information Technology & Security (IJCSITS), ISSN: 2249- 9555 Vol. 2, No.6, December 2012

[2] Vatsal Shah and Viral Kapadia, "Load balancing by process migration in distributed operating system" in International Journal of Soft Computing and Engineering (IJSCE), ISSN: 2231-2307, Volume-2, Issue-1, March 2012

[3] Paul Werstein, Hailing Situ and Zhiyi Huang, "Load balancing in a cluster computer", Proceedings of the Seventh International Conference on Parallel and Distributed Computing, Applications and Technologies, IEEE, 2006

[4] M. Kacer, P. Tvrdik, "Load Balancing by Remote Execution of Short Processes on Linux Clusters", Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID, 2002)

