IRACST - International Journal of Computer Science and Information Technology & Security (IJCSITS), ISSN: 2249-9555
Vol. 2, No.6, December 2012

Load Balancing algorithm by Process Migration in Distributed Operating System

Asst. Prof. Vatsal Shah
I.T. Dept. B.V.M. Engineering College
V.V. Nagar
Gujarat, India
vatsal.shah@bvmengineering.ac.in

Asst. Prof. Kanu Patel
I.T. Dept. B.V.M. Engineering College
V.V. Nagar
Gujarat, India
kanu.patel@bvmengineering.ac.in

***Abstract*—In present scenario load balancing in distributed operating system is challenging topic. There are various methods to balance the load. But in this paper we have focus on process migration technique. This paper focus different algorithm for load balancing in distributed operating system.**

Keywords- CPU utilization, Memory utilization

I. INTRODUCTION

A. *Load Balancing [1]*

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.

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.

B. *Process Migration [1]*

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 [3] 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 [3], 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) Sender-Initiated Algorithm
2) Receiver-initiated Algorithm
3) Symmetrically Initiated Algorithm

B. *Components of Algorithm:*

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 we all know there are many algorithms for load balancing in distributed operating system. Each has its merits and demerits. I have analyzed a few traditional algorithms and also proposed a new algorithm. We have actually implemented all algorithms including our proposed algorithm. We have checked it for both type of migration, preemptive and non-preemptive process migration.

Performance analysis of different algorithms in which process execution,

1) On standalone machine without any algorithm

On distributed system according to

2) CPU-utilization with Non-preemptive migration
3) Memory-utilization with Non-preemptive migration
4) CPU-utilization with preemptive migration
5) Proposed algorithm with Non-preemptive migration
6) Proposed algorithm with preemptive migration

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 processes can be CPU-bound or memory-bound. The type of a new process is unknown. The algorithm is centralized, but to avoid bottlenecks and a single point of failure I have keep its back up on another server. It considers CPU queue length, CPU utilization, and memory usage. These data are exchanged periodically between the cluster nodes to server.

A. Load estimation and information exchange policy [2]

Ideally, the load information should reflect the current CPU utilization and memory utilization 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 memory utilization. In the proposed algorithm, CPU utilization, CPU queue length, and memory utilization are used. The system statistics such as CPU utilization, CPU queue length 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 proposed algorithm, 5 seconds is set for the averaging interval. CPU utilization (cpu u), CPU queue length (nr) 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.

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

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 nr, cpu u, mem u).
- $p_1 \cdot\cdot\cdot p_t$ 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 averaged information including CPU utilization, CPU queue length and memory utilization are the load metrics used to describe the load on a node. The information exchange policy chosen for this research is a periodic policy with a time interval of one second.

B. Process transfer policy

In the proposed algorithm, this determination includes two steps. First the nodes are classified according to their loading metrics. Then a decision is made whether to start a process on another node.

1) Load classification [2]

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 queue length (nr), 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 three parameters is done as follow:

a) Calculate load average of each parameter (nr, cpu u, and mem u) over all nodes. The equation is:

$$L_{avg} = \frac{l_1 + l_2 + l_3 + \dots + l_n}{n} \quad (2)$$

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.
- $l_1, \cdot\cdot\cdot, l_n$ 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.

b) Calculate the thresold value

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

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

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

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 proposed algorithm, *H* and *L* are set to 1.3 and 0.7, respectively. That means when a certain load parmeter 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 queue length, 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, CPU queue length 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

2) *Transfer decision*

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 pseudocode defines how this decision is made.

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

This pseudocode 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. If no node can be found in the previous choices, the process is assigned to the local host.

C. Selection Policy [2]

The proposed algorithm does not attempt to determine the resource requirements of a newly arrived process. In addition, the proposed algorithm is preemptive and nonpreemptive.

D. Location policy [2]

When a process has been selected for transfer, the location policy determines the node to which the process should be transferred. For the cluster used in this research, the arrival of new processes is not centrally controlled and can occur at any time. In addition, there is no communication between nodes when they start new processes locally or start new processes on other nodes. The only communication for the load balancing algorithm is the periodic exchange of load data.

Since all nodes are grouped by the same algorithm, it is possible under worst case conditions for the same target node to be picked by several nodes at the same time. To help avoid this possibility, the proposed algorithm sorts all nodes in ascending order according to its CPU utilization, memory utilization or Queue length. Whenever it searches any kind of node then it returns the IP-address of that particular node easily.

IV. Experiment Results and Performance Evaluation

A. System Configuration

Ubuntu release 10.04 kernel 2.6.32-21 generic
Gnome 2.30.0

Memory:	993.0 MB
Dual core:	P0: 2.40 GHz
	P1: 2.40 GHz

B. Results

TABLE I. For CPU-bound process

Node's Status	CPU-utilization
Ideal	< 1%
Low1	1-20 %
Low2	20-35%
Normal	35-65%
Heavy	>65%

TABLE II. For memory-bound process

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

TABLE III. Process execution time for CPU-bound process on standalone machine

Node's status	Time(seconds)

Ideal	83.522427
Low1	93.258678
Low2	105.533306
Normal	167.817222
Heavy	402.291997

TABLE IV. FOR MEMORY-BOUND PROCESS ON STANDALONE MACHINE WITHOUT ANY ALGORITHM

Node' s status	Time(µs)
Ideal	667026
Low	803204
Normal	1555416
Heavy	Killed

TABLE V. SENNDER-INITIATED ALGORITHM FOR CPU-BOUND PROCESS BY HEAVY SENDER USING NON-PREEMPTIVE MIGRATION

Node's Status	Without algorithm (s)	CPU utilization (s)	Memory utilization (s)	Proposed algorithm (s)
Ideal	0	0	0	90.258656
Low1	0	114.731643	283.725934	100.734253
Low2	0	125.788001	307.337652	109.28367
Normal	0	188.298968	343.878954	163.828345
Heavy	402.291997	402.292865	403.166238	402.292734

TABLE VI. SENNDER-INITIATED ALGORITHM FOR CPU-BOUND PROCESS BY HEAVY SENDER USING PREEMPTIVE MIGRATION

Node's Status	Without algorithm (s)	CPU utilization (s)	Memory utilization (s)	Proposed algorithm (s)
Ideal	0	0	0	88.039754
Low1	0	111.731643	156.878954	97.166647
Low2	0	121.788001	198.371479	104.984916
Normal	0	167.819452	203.82789	142.677236
Heavy	402.291997	228.734569	397.622359	187.364926

TABLE VII. SENNDER-INITIATED ALGORITHM FOR MEMORY-BOUND PROCESS BY NORMAL SENDER USING NON-PREEMPTIVE MIGRATION

Node's Status	Without algorithm (µs)	CPU utilization (µs)	Memory utilization (µs)	Proposed algorithm (µs)
Ideal	0	0	0	556876
Low	0	2731643	803204	733452
Normal	1555416	2819452	1558916	1524769
Heavy	0	0	0	0

TABLE VIII. SENNDER-INITIATED ALGORITHM FOR MEMORY-BOUND PROCESS BY HEAVY SENDER USING NON-PREEMPTIVE MIGRATION

Node's Status	Without algorithm (µs)	CPU utilization (µs)	Memory utilization (µs)	Proposed algorithm (µs)
Ideal	0	0	0	635896
Low	0	2891823	912689	793693
Normal	0	2979496	1787816	1618391
Heavy	402.291997	228.734569	397.622359	187.364926

V. CONCLUSION

Here an algorithm has been present 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 algorithm for different type of application like CPU-bound process and memory bound process.

REFERENCES

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

[2] 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

[3] 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)