International Journal of Scientific Research in Computer Science, Engineering and Information Technology
© 2017 IJSRCSEIT | Volume 2 | Issue 1 | ISSN : 2456-3307

An Analytical Review : Static Load Balancing Algorithms

Navjot Jyoti
Assistant Professor, Department of Computer Science & Engineering, Northwest Group of Institutions, Dhudike, Moga, Punjab, India

ABSTRACT

Load balancing is the process of improving the performance of a parallel and distributed system by distributing load among different available processors. In literature we have multiple load balancing algorithms divided into two broad categories i.e. Static Load Balancing and Dynamic Load Balancing. This paper presents analytical review of three static load balancing algorithms and present the performance analysis of three static load balancing algorithms using execution time and waiting time as a benchmarks. Performance statistics of this behaviour evaluated by designing the simulator program in C++. On this simulator we have smaller number of jobs and larger number of job for different number of heterogeneous processors to get the execution time and waiting time.

Keywords : Round Robin, Randomized, Central Manager, Heterogeneous, Load Sharing, Load Balancing

I. INTRODUCTION

In parallel and distributed systems more than one processors processing parallel programs. The amount of processing time needed to execute all processes assigned to a processor is called workload of a processor. A distributed system provide the resource sharing as one of its major advantages, which provide the better performance and reliability than any other traditional system in the same conditions. One of the research issues in parallel and distributed systems is the development of effective techniques for distributing workload on multiple processors. The main goal is to distribute the jobs among processors to maximize throughput, maintain stability, resource utilization and should be fault tolerant in nature [1]. Some of the major benefits of parallel computing systems are information sharing among distributed users, resource sharing, better price/performance ratio, shorter response time, higher throughput, higher reliability, extensibility and incremental growth [5].

Load sharing policies may be either static or adaptive. Static policies use only information about the average behaviour of the system; transfer decisions are independent of the actual current system state. Static policies may be either deterministic (e.g., “transfer all compilations originating at node A to server B”) or probabilistic (e.g., “transfer half of the compilations originating at node A to server B, and process the other half locally”). Numerous static load sharing policies have been proposed. Adaptive policies, by contrast, are more complex, since they employ information on the current system state in making transfer decisions. This information makes possible significantly greater performance benefits than can be achieved under static policies [6].

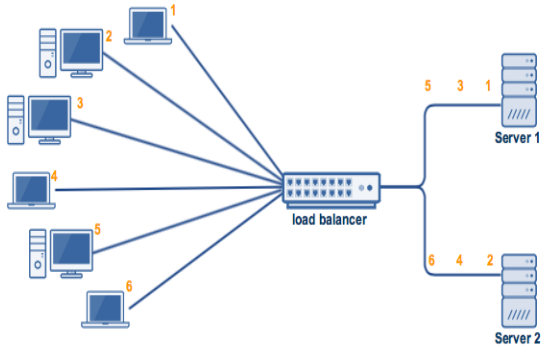
Load balancing is a technique applied in parallel system that is used to reach optimal system condition, which is workloads are evenly distributed amongst computers, and as its implication will decrease

programs execution time. One type of load balancing algorithm that may be used is static load balancing algorithm. This algorithm performs load balancing tasks before programs execution begin [1].

In static load balancing method the performance of the processors is determined at the beginning of execution. Then depending upon their performance the work load is distributed in the start by the master processor.

II. LOAD BALANCING STRATEGIES

a) *Round Robin Static Load Balancing Algorithm:* This algorithm distributes jobs evenly to all slave processors. All jobs are assigned to slave processors based on Round Robin order, meaning that processor choosing is performed in series and will be back to the first processor if the last processor has been reached. Processors choosing are performed locally on each processor, independent of allocations of other processors [1]. As shown below Figure 1.



Round Robin is undoubtedly the most widely used algorithm. It's easy to implement and easy to understand. Here's how it works. Let's say you have 2 servers waiting for requests behind your load balancer. Once the first request arrives, the load balancer will forward that request to the 1st server. When the 2nd request arrives (presumably from a different client), that request will then be forwarded to the 2nd server. Because the 2nd server is the last in this cluster, the next request (i.e., the 3rd) will be forwarded back to the 1st server, the 4th request back to the 2nd server, and so on, in a cyclical fashion.

Unfortunately, a load balancer running on a round robin algorithm won't be able to treat the two servers accordingly. In spite of the two servers' disproportionate capacities, the load balancer will still distribute requests equally. As a result, Server 2 can get overloaded faster and probably even go down. You wouldn't want that to happen.

b) *Randomized Static Load Balancing Algorithm:* This algorithm uses random numbers to choose slave processors [7]. The slave processors are chosen randomly following random numbers generated based on statistic distribution [1], as shown below figure 2.

c) *Central Manager Static Load Balancing Algorithm:* In this algorithm, in each step, central processor will choose a slave processor to be assigned a job. The chosen slave processor is the processor having the least load. Load is calculated here

CPU_{Load} = (No. Jobs in a queue/Total no. of jobs) * processor speed

The central processor is able to gather all slave processors load information, thereof the choosing based on this algorithm are possible to be performed [1]. As shown below in figure 3.

III. DESIGN OF SIMULATOR

In this simulator, we are providing some information to calculate the required results.

a) *information of jobs*

We are assuming a parallel program with small number (100) jobs and a parallel program with larger number (400) of jobs and assuming the information about each job i.e. average number of instructions and average CPI(Clock per Instruction) of each job.

b) Information about CPUs.

Also we are assuming different number of CPUs in this system and their speed in Hz. We are considering 5, 10, 15 and 20 heterogeneous CPUs.

c) Parameters used

We will calculate the execution time and waiting time for each job and then for a whole program. To calculate these parameters we need to find some intermediate results which are used to find final values.

To calculate the job execution time we need CPI and Number of instruction in job.

$$Job_{exe\_time} = Number_{instructions} * Instruction_{Exe\_Time} + Time_{qdelay}$$

Where $Instruction_{Exe\_time}$ is the time taken by single instruction to execute, $Number_{instructions}$ average number of instruction in a job and $Time_{qdelay}$ is the time spent in waiting queue.

To calculate instruction execution time we need to know CPI and time taken by clock tick.

$$Instruction_{Exe\_time} = Time_{clock\_tick} * CPI$$

To calculate the time taken by clock tick we use following equation.

$$Time_{clock\_tick} = 1/ Speed_{cpu}$$

This simulator will run on the load balancing computer (Master) and it will choose the CPU (Slaves) to assign the job. We are assuming we have a parallel program divided into number of job. Master will take a bunch of jobs in a parallel program and assign one by one to different CPUs according to a specific criteria. We have all jobs in hand to distribute, means no job is coming after starting the process of load balancing.

There may be executing other processes on the CPU then some processes have to wait, so there maximum jobs in the queue will be 20. When we have instruction execution time of each job then job total execution time can be calculated. Then it's easy to find the total execution time of the parallel program. To calculate the waiting time of each job in the program we need to know the waiting time and execution time of the preceding job.

To simplify analysis of simulation results, some assumptions are made to limit the simulated system. Following are the assumptions.

1. All jobs are static in nature i.e. no jobs are coming after starting the process of scheduling.
2. Computers used in parallel system are heterogeneous in nature.
3. The compared Static load balancing algorithms are Round Robin, Randomized, and Central Manager.
4. The load index used in Central Manager is CPU Load.
5. The algorithm comparison is performed based on two simulation results types, i.e. execution time and Waiting Time.
6. The simulated parallel system is a heterogeneous with total CPUs of 5, 10, 15, 20.
7. CPUs have taken with different speeds of 500, 1000, 1500 and 2000 in MHertz for heterogeneous.
8. Maximum Queue length at each CPU can be 20.
9. The jobs in the simulated parallel system are independent, and the simulated parallelism is on job or process level.
10. Other all factors are considered as constant like memory access time, input output access time, communication delay etc.

IV. SIMULATION PARAMETERS

In the simulator that will be run, there are three main components computer, program, and static load balancing algorithm. The three components have some parameters that their values should be set. The value for the simulation parameters is determined based on some assumptions. The parameters with their value are summarized in Tables below:

TABLE 1

Computer Parameters		
	Number of CPUs	**Description**
No_cpu	5,10,15,20	The amount of CPUs that is used in

		parallel system.
speed_cpu	500,1000,1500,2000, 2500 for heterogeneous in MHz	Processing unit clock speed.
cpi	5	The average of clock per instruction

TABLE 2

Program Parameters		
	No. of jobs	**Description**
no_jobs	100, 400	Total number of jobs in a program
no_ins	500	Mean total instruction in a job

TABLE 3

Algorithm type factor		
	Type of cpu	**Description**
round_robin	For heterogeneous	Round robin algorithm
randomized	For heterogeneous	Randomized algorithm
central_manager	For heterogeneous	Central manager algorithm that uses CPU load as a load index.
end_q		Queue length

TABLE 4

Amount of computer factor		
Level 1	5	5 CPUs are used
Level 2	10	10 CPUs are used
Level 3	15	15 CPUs are used
Level 4	20	20 CPUs are used

There are two things that will be measured from simulation result; they are execution time and total waiting time of jobs in a parallel program. To obtain these, running simulation and then the simulation result in form of execution time and waiting time are calculated. Simulation runs based on two factors combinations: they are algorithm and amount of computers. The algorithm factor has 3 levels, three algorithms on heterogeneous CPUs, while amount of computers factor has 4 levels i.e. 5, 10, 15 and 20 CPUs as given in Table 5. In order to indicate variations, then repetition is applied to the simulation experiment.

V. EXPERIMENTAL SETUP AND PERFORMANCE MEASUREMENTS

To get the measurements, this simulator run on the machine and provides the information about the jobs as we have given above factors. We run different number of jobs by providing different number of CPUs to the simulator.

Experiment 1. We run 100(smaller number of) jobs on the heterogeneous CPUs. It gives the values given in table 5. Here execution time and waiting calculated in the simulator as discussed above method. Here we use notations as RR represents Round Robin, CM represents Central Manger and RD represents Randomized, similarly ET and WT as Execution time and Waiting Time respectively. We can observe that if we have heterogeneous CPUs, then Central manager gives best results among these three methods. If we increasing the number of CPUs, then Round Robin reducing the execution time faster than others. The values given in table 5.

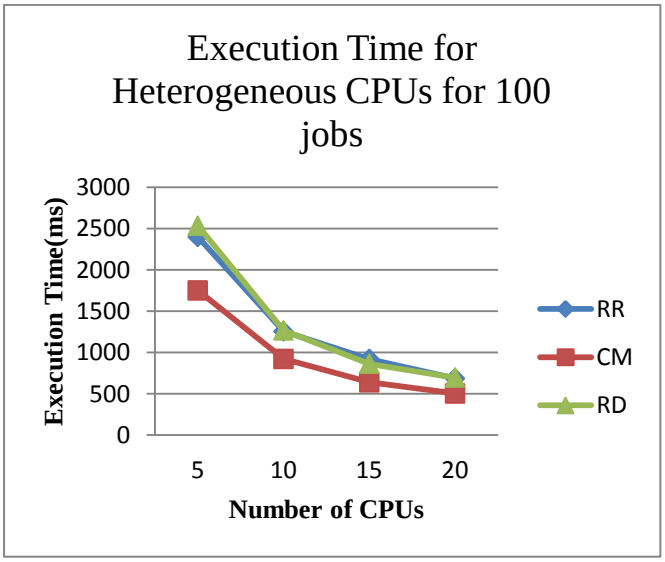
TABLE 5

Jobs ↓ CPUs →			5	10	15	20
100	RR	ET	2625.000	1375.000	962.500	750.000
		WT	2375.000	1125.000	712.500	500.000
	CM	ET	2625.0	1375.0	962.50	750.0

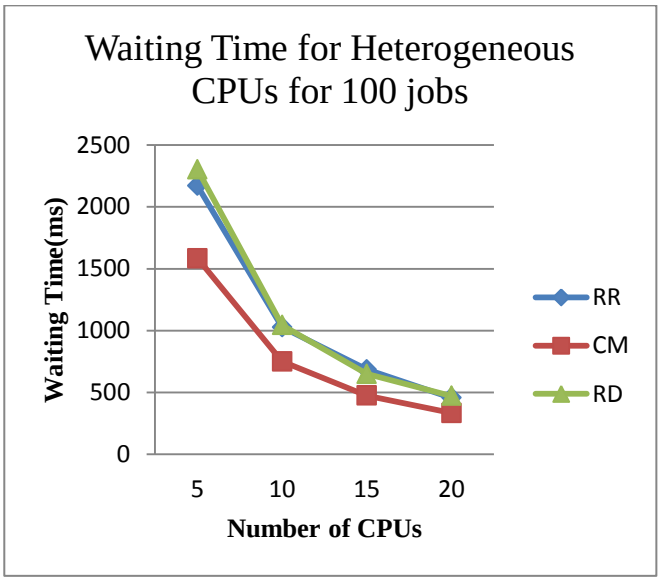
			00	00	0	00
		WT	2375.000	1125.000	712.500	500.00
	RD	ET	2705.500	1455.000	1062.500	792.500
		WT	2457.500	1205.001	812.500	542.500

Graphical representation of this result as follows.

Graph 1



Graph 2



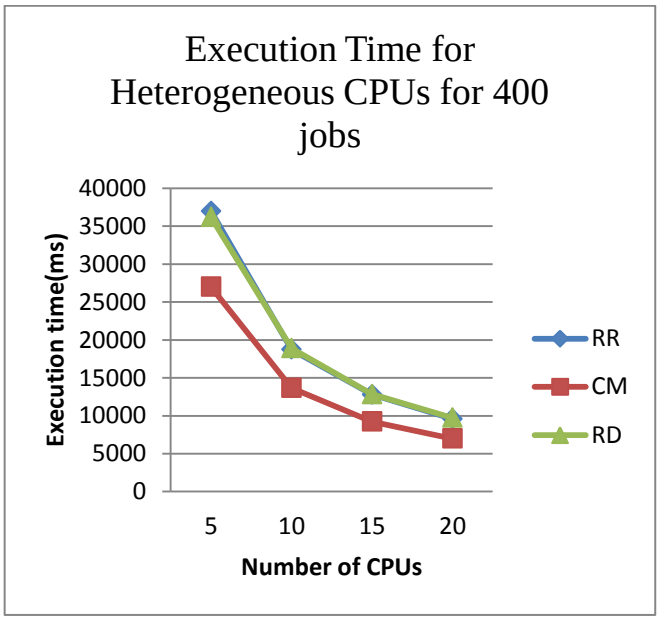
Experiment 2. Then we run 400 (larger number of) jobs on same kind of situations as we had for 100 jobs. If we have heterogeneous type of CPUs then we can see that Central Manager gives better results than randomized and round robin. The values are given in table 6.

TABLE 6

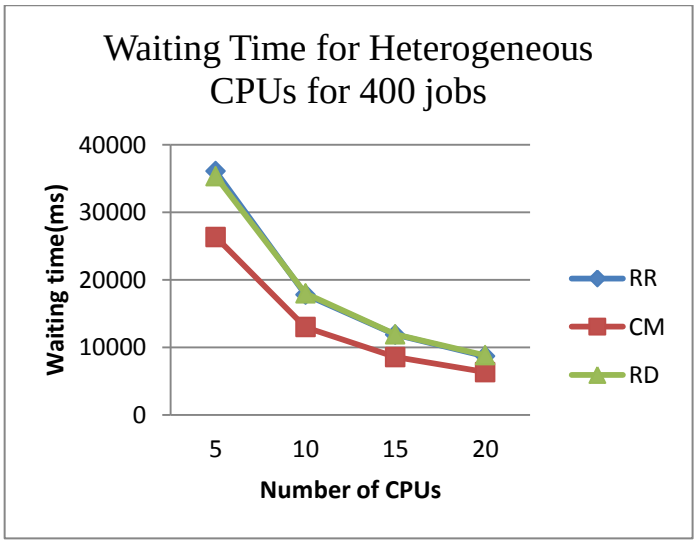
400	RR	ET	40500.004	20500.002	13837.500	10500.000
		WT	39500.004	19500.002	12837.501	9500.000
	CM	ET	40500.004	20500.002	13837.500	10500.000
		WT	39500.004	19500.502	12837.500	9500.000
	RD	ET	40597.504	20777.502	14482.501	10612.501
		WT	39597.504	19777.502	13482.501	9612.501

Graphical representations of these results as follows.

Graph 3



Graph 4



VI. INTERPRETATIONS OF RESULTS

These results have been taken for two parameters i.e. execution time and waiting time. As it can be seen from the graphs that central manger gives the best results among these three algorithms. It can observe that as number of CPUs increases, round robin gives the better results from randomized algorithms. On the other hand all algorithms behave very similar for waiting time as these were for execution time. One thing here to note that randomized algorithm might be good as comparison to Round robin, at some situations because of random number generated, but in average case round robin is better that randomized.

VII. CONCLUSION

According to my parameters of simulation I concluded that Central Manager Algorithm is best algorithm among these three algorithms and gives best results among others because it distributes the load fairly i.e. each time when it has to assign a job to a CPU, it chooses the CPU which has minimum load at that moment. On the other hand Round Robin Algorithm distribute the load evenly, it don't consider any kind of load of any CPU. It just divided the load evenly among all CPUs. And if we talk about Randomized Algorithm then its performance totally depends upon the random numbers generated, overall its performance is very similar to Round Robin Algorithm.

VIII. REFERENCES

[1]. Sharma S., Singh S., and Sharma M. , Performance Analysis of Load Balancing Algorithms, Proceedings of World Academy of Science, Engineering, and Technology, 28, 269-272, 2008.

[2]. S. Malik, "Dynamic Load Balancing in a Network of Workstation", 95.515 Research Report, 19 November, 2000.

[3]. Shirazi B. A., Hurson A. R., and Kavi K. M., Scheduling and Load Balancing in Parallel and Distributed Systems, IEEE Computer Society Press, California 1995.

[4]. Derek L. Eager, Edward D. Lazowska , John Zahorjan, "Adaptive load sharing in homogeneous distributed systems", IEEE Transactions on Software Engineering, v.12 n.5, p.662-675, May 1986.

[5]. Amit Chhabra, Gurvinder Singh, Sandeep Singh Waraich, Bhavneet Sidhu, and Gaurav Kumar, Qualitative Parametric Comparison of Load Balancing Algorithms in Parallel and Distributed Computing Environment, Proceedings of World Academy of Science, Engineering, and Technology, 2006.

[6]. Derek L. Eager, Edward D. Lazowska and John Zahorjan, "A Comparison of Receiver-Initiated and Sender-Initiated Adaptive Load Sharing". ACM 0-89791-169-5/85/007/0001,1985.

[7]. Motwani, R. and Raghavan, "Randomized Algorithms", ACM Computing Surveys, 28, 33-37, 1996