International Journal For Technological Research In Engineering
Volume 2, Issue 10, June-2015 *ISSN (Online): 2347 - 4718*

ADVANCE TECHNIQUE OF LOAD BALANCING THROUGH TASK MIGRATION IN DISTRIBUTED SYSTEM

Shashank Sharma[1], Mr. Ashutosh Kumar[2]
[1]PG Scholar, [2]Guide
M. Tech (CSE), Subharti Institute of Technology and Engineering, Meerut

ABSTRACT: Load Balancing in distributed system has an important impact on the performance. In previous system when server is idle, scheduler select random task which increased communication overhead. We illustrate this problem through assigning task iteratively which keep server busy. The purpose is to provide way to balance the load for a given task among the servers .Easy to maintain balance the load among server in distributed system by migrating process from one server to another server. To achieve load balancing and sharing resources Linear scheduling is used to assign task and resources to server. Scheduling the task to the number of servers in the distributed system done by dividing the task according to the severs in the distributed system. This idea is implemented in real time and it will reduce execution time.

Index Terms: Task Migration, Task Scheduling, Load Balancing

I. INTRODUCTION

A distributed system where set of a heterogeneous nodes are connected through underlying arbitrary communication networks and each with its own memory that work together towards a joint goal. Distributed systems are used for proper sharing and utilization of the available resources within the distributed environment .Users has all of the capabilities that are provided by their workstations, and at most times, this is sufficient. However, there are cases when the power of one workstation is not sufficient to complete all of the tasks at hand.It's created load imbalance in distributed systems. It is required for such workload imbalance to be minimized so as to make use of the computing power of idle or lightly loaded machine .One solution to this problem is to distribute some of the tasks to an idle workstation.

Distributed programming typically falls into one of several basic architectures or categories: Client-server, 3-tier architecture, N-tier architecture, Distributed objects, loose coupling, or tight coupling.

Client-server — A server is a process in a distributed system that provides a specific service to its clients. Typically, the name of server and a specification of its service are widely advertised in a system. Any process can send a message to a server and become its client .Smart client code contacts the server for data, then formats and displays it to the user. Input at the client is committed back to the server when it represents a permanent change. Client-server computing is a poor paradigm for distributed computing because methodologies for structuring a distributed computation in the form of a client-server configuration have not been evolved.

3-tier architecture — three tier systems move the client intelligence to a middle tier so that stateless clients can be used. This simplifies application deployment. Most web applications are 3-Tier.

N-tier architecture — N-Tier refers typically to web applications which further forward their requests to other enterprise services. This type of application is the one most responsible for the success of application servers.

Tightly coupled (clustered) — refers typically to a set of highly integrated machines that run the same process in parallel, subdividing the task in parts that are made individually by each one, and then put back together to make the final result.

Peer-to-peer — an architecture where there is no special machine or machines that provide a service or manage the network resources. Instead all responsibilities are uniformly divided among all machines, known as peers. Peers can serve both as clients and servers.

Space based — refers to an infrastructure that creates the illusion (virtualization) of one single address-space. Data are transparently replicated according to application needs. Decoupling in time, space and reference is achieved.

II. LOAD BALANCING VS. LOAD SHARING

Load distributing algorithms can further be classified as load balancing or load sharing algorithms, based on their load distributing principle. Both types of algorithms strive to reduce the likelihood of an unshared state (a state in which one computer lies idle while at the same time tasks contend for service to another computer) by transferring tasks to lightly loaded nodes. Load balancing algorithms, however, go a step further by attempting to equalize loads at all computers. Because a load balancing algorithm transfers tasks to a higher rate than a load-sharing algorithm, the higher overhead incurred by the load balancing algorithm may outweigh this potential performance improvement. Task transfers are not instantaneous because of communication delays and delays that occur during the collection of task state. Delays in transferring a task increase the duration of an unshared state, as an idle computer must wait for the arrival of the transferred task. To avoid lengthy unshared states, anticipatory task transfers from overloaded computers to computers that are likely to become idle shortly can be used. Anticipatory transfers increase that task transfer rate of a load-sharing algorithm, making it less distinguishable from load balancing algorithms. In this sense, load balancing can be considered a special case of load sharing, performing a

particular level of anticipatory task transfers.

III. REQUIREMENTS FOR LOAD DISTRIBUTING

While improving system performance is the main objective of a load distributing scheme, there are other important requirements it must satisfy.

Scalability: It should work well in large distributed systems. This requires the ability to make quick scheduling decision with minimum overhead.

Location transparency: A distributed system should hide the location of tasks, just as a network file system hides the location of files from the user. In addition, the remote execution of task must produce the same results it would produce if it were not transferred.

Determinism: A transferred task must produce the same results it would produce if it were not transferred.

Pre-emption: While utilizing idle workstations in the owner's absence improves the utilization of resources, a workstation's owner must not get a degraded performance on his return. Guaranteeing the availability of the workstation's owner must not get a degraded performance on his return. Guaranteeing the availability of the workstations' resources to its owner requires that remotely executed tasks be pre-empted and migrated elsewhere on demand. Alternatively, these tasks may be executed at a lower priority.

Heterogeneity: It should be able to distinguish among different architectures, processors of different processing capability, servers equipped with special hardware, etc.

IV. LOAD-BALANCING TAXONOMY

A. Load Balancing

In computer networking, load balancing is a technique (usually performed by load balancers) to spread work between two or more computers, network links, CPUs, hard drives, or other resources, in order to get optimal resource utilization, throughput, or response time. Using multiple components with load balancing, instead of a single component, may increase reliability through redundancy.

This division of system load can take place statically or dynamically:

1. Static load distribution assigns jobs to hosts probabilistically or deterministically, without consideration of runtime events. This approach is both simple and effective when the workload can be accurately characterized and where the scheduler is pervasive. Static load balancing can be classified
into two categories-optimal and sub-optimal.

2. Optimal SLB (Static Load Balancing): When all the information regarding the state of the system as well as the resource needs is known an optimal assignment can be made based on some criterion function. Examples of optimization measures are minimizing total process completion time, maximizing utilization of resources in the system, or maximizing system throughput. For example simulated Annealing (SA) and genetic algorithms (GA's) are optimization techniques.

3. Sub-optimal SLB: When for some of computations, optimal solution does not exist then sub-optimal methods can be applied. These methods rely on the rules-of-thumb and heuristics to guide a scheduling process. List scheduling is the most popular technique despite of poor performance in high communication delay situations. Lot of static algorithms, taking into account their optimal and sub-optimal nature, has been suggested by researchers so far

4 Dynamic load distribution: systems typically monitor the workload and hosts for any factors that may affect the choice of the most appropriate assignment and distribute jobs accordingly. The main problem with SLB algorithms was that they assume too much job information which may not be known in advance even if it is available, intensive computation may be involved in obtaining the optimal schedule. Because of this drawback much of the interest in load balancing research has shifted to DLB algorithms that consider the current load conditions (i.e. at execution time) in making job transfer decisions. So here the workload is not assigned statically to the processing hosts as was being done in SLB but instead of this workload can be redistributed among hosts at the runtime as the circumstances changes i.e. transferring the tasks from heavily loaded processors to the lightly loaded ones

V. LOAD BALANCING ALGORITHM

Paul werstien dynamic load balancing algorithm

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. The experimentation results with proposed algorithm had shown better results than traditional one that considers only queue length as metric.

Sammulal Algorithm

Sammulalet al. proposed an algorithm which assigns a cluster for an incoming job. 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 Algorithm

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. 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. 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.

Enhanced Module Migration Algorithm

Now, we are presenting a new algorithm which was found to be very good at experimental results. We name it as an Enhanced Module Migration Algorithm. In the later sections of the thesis, we will compare it with some well-known algorithms for module migration. The objective of the Enhanced Module Migration algorithm is to balance the load on nodes as efficiently as possible and thus minimize load imbalance. The balancing operation is performed by matching nodes with high loads with nodes with low load, and then letting migration take place within the pair. The algorithm is described below:

Enhanced Module Migration Algorithm

Step 1:	Sort the nodes in the decreasing order of their load imbalance.
Step 2:	For each node n repeat step 3
Step 3:	Sort the modules/tasks of node n in decreasing order of their load
Step 4:	Let i = 1
Step 5:	Let k be the total number of nodes
Step 6:	While i < k repeat steps 7 to 16
Step 7:	Let ns represents the i^{th} node
Step 8:	Let nd represents the k^{th} node
Step 9:	If load of ns >w_{ref} of ns **and** load of nd is <w_{ref} of nd then go to step 10 else go to step 15
Step 10:	Let j=1
Step 11:	While j <= number of modules allocated to ns repeat steps 12 to 14
Step 12:	Let m is the j^{th} module of ns
Step 13:	If nd can accommodate a node **and** sum of load of nd and m is less than w_{ref} of nd then migrate the module m from node ns to node nd.
Step 14:	increment j by 1
Step 15:	increment i by 1
Step 16:	decrement k by 1

In this way, each pair has one node with high load called the source node, ns and one node with low load called destination node, nd. In the pair (ns; nd), ns will be the source from which a module may be migrated to destination nd. For each (ns; nd) pair, the module to be migrated is selected in a greedy fashion: migrate the largest (in load) module that will fit on nd, since this will simultaneously decrease the load imbalance on both nodes the most. The can accommodate check handles non-performance related allocation policies, such as security or reliability.

VI. RESULT

In this approach, we change the way in which match-making algorithm sorts the nodes of the system. In general, the Match-Making algorithm sorts the nodes in decreasing order of their load. However, here we sort them in decreasing order of their load imbalance (i.e. the deviation of a node's load from its reference load (w-ref)). This gives a better pairing and a more optimized solution. In our test case, it reduced the overload of the entire system by 80%. Thus we found that in our test cases, the Enhanced Module Migration Algorithm provides the best solution and is most effective.

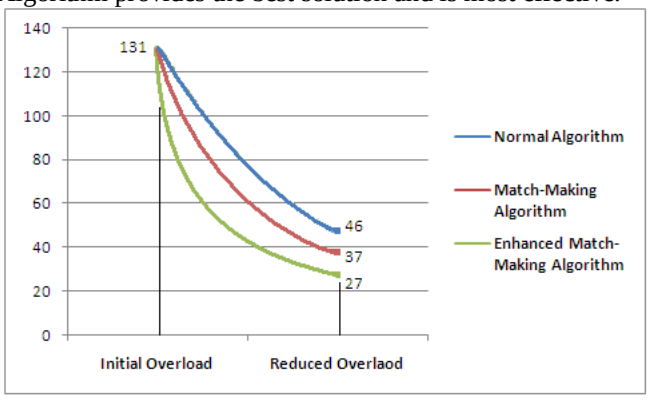

Figure 1: Results of the Experiment

VII. CONCLUSION

A task migration model has been formulated for dynamic load balancing of a distributed computing system in the context of load balancing. Enhanced task migration algorithm discuss in this thesis short the node and module according to load and group the node with a coordinator node and then migrate modules from overloaded nodes to nodes , algorithm first search light weighted node within group if it is not found then search nearby group node. As the main aim of Enhanced task migration algorithm is reducing the likelihood of nodes being idle while there are tasks in the system. The algorithm has low complexity and high scalability, and as such is well suited for performing load balancing in dynamic systems.

REFERENCES

[1] MorHarchol-Balter and Allen B. Downey. Exploiting Process Lifetime Distributions for Dynamic Load Balancing. Technical Report
[2] UCB/CSD-95-021, Computer Science Division, University of California, Berkeley, May 1995.
[3] Kristian Paul Bubendorfer "Resource Based Policies for Load Distribution", Victoria University of Wellington August 3, 1996
[4] William Osser, "Automatic Process Selection for Load Balancing", June 1992
[5] Jerrell Watts, "A Practical Approach to Dynamic Load Balancing", October 4, 1995
[6] Dynamic Migration Algorithms forDistributed Object Systems by V. Kalogeraki, P. M. Melliar-Smith and L. E. Moser, Department of Electrical and Computer Engineering, University of California
[7] Match Maker Algorithm from Migration Algorithms for Automated Load Balancing by N. Widell.
[8] Sammulal, P. and A. VinayaBabu, 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
[9] SagarDhakal, Majeed M. Hayat, Senior Member, IEEE, Jorge E. Pezoa, Cundong Yang, and David A. Bader, Dynamic Load Balancing in Distributed system in presence of delay IEEE Transactions On Parallel And Distributed Systems, Vol. 18, No. 4, April 2007
[10] Hung-Chang Hsiao,HaoLiao,Kuo-Chan Huang,Load Balance with Imperfect Information in Structured Peer-to-Peer Systemspp. 634-649April 2011
[11] T.T.Y. Suen, J.S.K. WongEfficient Task Migration Algorithm for Distributed Systemspp. 488-499July 1992 (vol. 3 no. 4)
[12] L. M. Ni, C. Xu, and T. B. Gendreau, "A distributed drafting algorithm for load balancing,"IEEE Trans. Software Eng., vol. SE-11, no. 10, pp. 1153-1161, Oct. 1985.
[13] J.-C. Ryou and J. S. K. Wong, "A task migration algorithm for load balancing in a distributed system," inProc. 22nd Annu. Hawaii Int. Conf. Syst. Sci., Vol. II, Jan. 1989, pp. 641-1048, Nominated as a candidate for best paper awards.
[14] C.-C. Shen and W.-H. Tsai, "A graph matching approach to optimal task assignment in distributed computing systems using a minimaxcriterion,"IEEE Trans. Comput., vol. C-34, no. 3, Mar. 1985.
[15] H. S. Stone, "Multiprocessor scheduling with the aid of network flow algorithms,"IEEE Trans. Software Eng., vol. SE-3, no. 1, Jan. 1977.
[16] Chyouhwa Chen, Kun-Cheng TsaiThe Server Reassignment Problem for Load Balancing in Structured P2P Systemspp. 234-246February 2008 (vol. 19 no. 2)
[17] Y. Amir, B. Awerbuch, A. Barak, R.S. Borgstrom, and A. Keren, "An Opportunity Cost Approach for Job Assignment in a Scalable Computing Cluster ," IEEE Trans. Parallel and Distributed Systems, vol. 11, no. 7, pp. 760-768, July 2000
[18] J. Watts and S. Taylor, "A Practical Approach to Dynamic Load Balancing," IEEE Trans. Parallel and Distributed Systems, vol. 9, no. 3, pp. 235-248 March 1998
[19] L. Fratta, M. Gerla, and L. Kleinrock "The Flow Deviation Network Design," Networks, vol. 3, pp. 97-133, 1973
[20] D. Grosu and A.T. Chronopoulos, "A Game-Theoretic Model and Algorithm for Load Balancing in Distributed Systems, "Proc. 16th Int'l Parallel & Distributed Symp., Apr. 2002.
[21] M. Mitzenmacher, "The Power of Two Choices in Randomized Load Balancing," IEEE Trans. Parallel and Distributed Systems, vol. 12, no. 10 pp. 1094-1104, Oct. 2001.
[22] M. Mitzenmacher, "How Useful Is Old Information?" IEEE Trans. Parallel and Distributed Systems, vol. 11, no. 1, pp. 6-20 Jan 2000
[23] J. Li and H. Kameda, "Load Balancing Problems for Multiclass Jobs in Distributed/Parallel Computer Systems," IEEE Trans. Computers, vol. 47, no. 3, pp. 322-332, Mar. 1998.
[24] A.N. Tantawi and D. Towsley, "Optimal Static Load Balancing in Distributed Computer Systems," J. ACM, vol. 32, no. 2, pp. 445- 465, Apr. 1985.
[25] M.H. McDougall, Simulating Computer Systems. MIT Press, 1987.
[26] Z. Zeng and V. Bharadwaj, "Design and Analysis of a Non- Preemptive Decentralized Load Balancing Algorithm for Multi- Class Jobs in Distributed Networks, " Computer Comm., vol. 27, pp. 679-694, 2004.
[27] H.C. Lin and C.S. Raghavendra, ªA Dynamic Load-Balancing Policy with a Central Job Dispatcher (LBC),º IEEE Trans. Software Eng., vol. 18, no. 2, pp. 148-158 , Feb. 1992.
[28] Z. Zeng and V. Bharadwaj, "Design and Analysis of a Non- Preemptive Decentralized Load Balancing Algorithm for Multi- Jobs in Distributed Networks," Computer Comm., vol. 27, pp. 679-694, , 2004.
[29] Krueger, P. and R. Chawla, 1991. The stealth distributed scheduler. 11th IEEE International Conference on Distributed Computing Systems, pp: 336-343.

