Load Balancing in Distributed System through Task Migration

Santosh Kumar Maurya[1]
Subharti Institute of Technology & Engineering Meerut India
Email- santoshranu@yahoo.com

Khaleel Ahmad[2]
Assistant Professor Department of CSE/IT
Subharti Institute of Technology & Engineering Meerut India
Email- khaleelamna@yahoo.co.in

***Abstract*— In distributed systems or clustered systems load imbalance is a major problem to improve system performance. Load balancing is process through which utilizing computing resources of lightly nodes by transferring some jobs on that node from highly overloaded nodes. The sharing and utilization of computer resources in a distributed system or clustered system is a more complicated task than in equivalent centralized systems, due to the distribution of computer resources over a set of autonomous and often physically separate nodes. For effective use of these fragmented resources load balancing are required to decrease response time and other CPU utilization constraints. Load balancing enables busy sites to offload some works to lightly loaded sites. But problem with load balancing algorithms is that if it is not implemented properly it increases network overhead and decreases the performance of systems. In this paper, we present an algorithm for load balancing which calculate dynamically the load of each node and migrates the task on the basis of predefined constraint which reduce network overhead.**

Keywords: Load Balancing, migration, Distributed Systems, Load Dispatcher, Load Reduction.

1. INTRODUCTION

A distributed system where set of a heterogeneous nodes are connected through underlying arbitrary communication networks that work together towards a joint goal. Distributed systems are used for proper sharing and utilization of the available resources within the distributed environment. But in this computing environment there are situation when the processing capacity of one node is not capable to execute all of the tasks at given time or rate of job arrival at one node have higher then others.

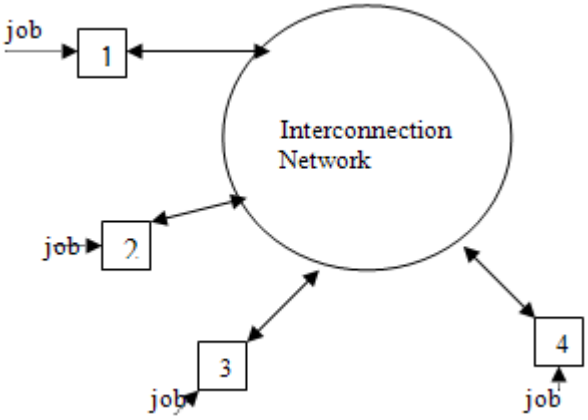

Fig1: Distributed computer system

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. Load balancing algorithms try to accomplish this works by transferring some task to other machines so that work load of each node is approximately same. This user-transparent mechanism increases the amount of processing capacity to all users of the systems. Task migration is a complex job in heterogeneous environment. Many different algorithms of load balancing for multiprocessing and distributed system have been presented. Load balancing algorithms may be classified as static or dynamic, depending upon the rule they follow for execution of jobs. Static load balancing algorithms assign tasks to a processor using priori task information (e.g. execution time ,arrival time, waiting time, amount of resources requirement and their interprocess communication requirements) and load once assigned does not change. A job is either assigned to the processor that received the job or transferred to another processor, but the decision regarding transfer of the task does not depend on the system state. Dynamic

load balancing algorithms uses current system load information to decide where in the network job should processed. If the workload of a node becomes heavy then the job received at the processor are transferred to another node which is not heavily loaded. Scheme of approach used for task migration in my paper is calculate the load of every module and every computing processor and calculation of computation power used in transfer of modules between nodes and choosing modules to transfer. This paper presents the advance task migration algorithm that performs load balancing by sorting node and module and pairing overloaded nodes with under loaded ones then task migration take place. Job migration in network environment can be implemented in a synchronous or asynchronous manner. This issue is primarily determined by maintenance of state and channel integrity. Certainly a synchronous implementation simplifies the need for handling messages in transit. In any case, a task must only be moved when its state is in a form consistent with any user-provided routines to transport data structures. In this paper we present a load balancing algorithms which improve the performance of the whole system. The algorithms take into account the load of module as well as the load of the node to which the transfer is made.

2. THE SYSTEM MODEL

In our model, we assume that there are N heterogeneous nodes Pi, i=1,2,3……N in the distributed system and nodes are belonging to a different domain connected through a high speed communication network. Due to varying computational facilities of each node, a module to be accounting different costs when executing on different computing node. In model we assume nodes are fully connected with each other. In the paper we use the term software load as the software task or software job generated by software modules when running on physical node or processor. High load on a node may increase delays in execution of a job. A module may come in as input to a node directly from outside or as a transfer from neighbors. Two types of module may thus be executed in a processor: locale module and remote module. A local module comes directly from outside the system. A remote module, on the other hand, is received at a particular node as a transfer from one of its neighbour's node. Programs in main memory are represented by M modules. Modules on a node are an atomic collection of programs, like a program object or program parts that can be migrated from one computing node to another. Matrix X (N X M) is used to represent execution load for module. Elements of X is represented by xij if module j contribute load on node i otherwise xij=0 if module is not allocated due to grouping constraints. In system communication model we assume actual communication cost is zero if two software modules are present on same node. Otherwise, cost of communication is given by a matrix C(M X M), where quantity ckl is the communication load if software modules k and software modules l are assigned to separate nodes. We assume ckk = 0 and ckl = clk. Messaging passing protocol is used for communication between nodes; A process is a sequence of module interactions, with given time frame. It is the arrival of process that makes the module works. The executing modules on a node generate loads on that node. Assignment of a module to a node is represented by binary matrix B. Elements of B is either 0 or 1 if Bij=1 then module j is assigned to node i otherwise Bij=0.

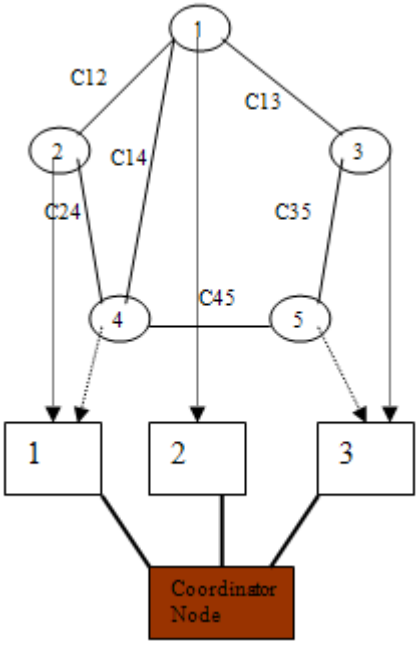

Fig:2. Model of System

Assigned the modules to processing nodes in such way that it's best meet the given condition and objectives. Formulation of load balancing depends on the nature of problem and communication load, several different conflicting objectives are existing. The main problem of objectives for performance is between grouping and migration of modules. By grouping software modules that passing message to each other during execution we can reduce cost of communication. Process of grouping modules decreases communication cost but it increase load imbalance on some nodes. Load imbalanced on a node increases waiting time of some module while some nodes are idle during this period. By distributing modules during execution higher parallelization is gained, but the more resources are wasted during migration processing. However, lot of consumptions of network resources in process of task distribution. Total load on particular node formulation depending on execution load, communication with other and computing parameter. Since the objectives of this works are to study load balancing using module migration, we chose an objective that minimizes the load imbalance in system. Workload Wi for selected node i and given allocation B be:

$$W_i(B)=\sum_{J=1}^{M}(X_j B_{ij}+\sum_{k=1} C_{jk}B_{ij}(i-B_{ik}))$$

Above formula is used to calculate load on a given node i and it should be close to the predefined reference load $W_{ref,i}$ constraints for node i given assignment B. We use notation L for load imbalance for allocation B is:

$$L=\sum_{i=1}\sum_{j=1} \left| Wi(B)-Wref,i(B) \right|$$

where Wref,i(B) is the reference load of the nodes weighted by node capacity. Each processing node may not be assigned more than it's capacity:

3. Normal Module Migration Algorithm

In the Normal algorithm, randomly select source node for transfer of unbalanced module and a destination node in pair(ns,nd). Pseudocode of algorithm is given below:

Line 1:	Count total number of processing node
Line 2:	For every node n repeat Line 4
Line 3:	i = 0
Line 4:	Let total number of nodes=t
Line 5:	While i < t do
Line 6:	Select random source node(ns) it is i^{th} node in system
Line 7:	Select random overloaded destination node(nd) it is k^{th} node in system
Line 8:	if load at node ns is (ns > w_{ref}) **&&** load at nd is(nd < w_{ref}) then execute Line 9 else i=i+1
Line 9:	j=1
Line 10:	While j <= total modules assigned to ns repeat Line 11 to 13
Line 11:	Let m is the j^{th} module of ns
Line 12:	If nd can accommodate a node **and** nd(load)+m(load)<nd(w_{ref}) then migrate m from ns to nd
Line 13:	j=j+1
Line 14:	i=i+1
Line 15:	t=t+1
Line 16:	repeat

Normal algorithm not properly migrate the task and select the node. Normal algorithm is suitable for small systems.

4. ADVANCE TASK MIGRATION ALGORITHM

Objective of this algorithm is to minimized load imbalance by transferring some task to other lightly loaded node. In algorithm we first short node and module then by comparing nodes with largest loads with nodes that have lowest load, and then starting migrating task. Pseudocode of algorithm is given below:

Line 1: node.sort() sort the imbalance node in increasing order of load.
Line 2: for all nodes in node list repeat Line 3
Line 3: Sort the modules load/tasks load of node n in increasing order
Line 4: t= total nodes in model
Line 5: Initialize i = 0
Line 6: While i < t do
Line 7: Select nd that represents the i^{th} node
Line 8: select ns that represents the $t\text{-}1^{th}$ node
Line 9: if load at node ns is (ns > w_{ref}) **&&** load at nd is(nd < w_{ref}) then execute Line 10 else i=i+1
Line 10: j=1
Line 11: While j <=total modules at node ns repeat Line 12 to 14
Line 12: Let m is the j^{th} module of ns
Line 13: if nd(load)+m< w_{ref} then migrate the migrate(ns,nd,m)
Line 14: j=j+1
Line 15: i=i+1
Line 16: t=t-1
Line 17: repeat

Above algorithm works in pair of node(ns,nd), ns is overloaded source node from which some load is migrated to underweight destination node nd. The algorithm works in some specified intervals. During the T interval load of each node is calculated. Matching operation is performed at a centrally located server node collects load information and then transferred according to imbalance. Server node is an ordinary node that acts as hub for transferring load information in the network. The matching process is made by load imbalance order: the first pair has the most overloaded node and the must under loaded node, second pair has the second most overloaded node and the second most under-loaded and so on.

The module is selected in a gluttonous fashion to be migrated: always select largest module for migration which are fit on destination node this process make load imbalance on both node and decrees network consumption.

5. COMMENTS

Load metrics: the individual nodes can estimate the fraction of its total load using analytic load calculation in this method; the load for a task is estimated based on knowledge of the time complexity of the algorithm(s) that task is executing along with the data structures on which it is operating This method has the advantage that it is potentially very responsive to a job for which the relative workload is changing quickly over time. In particular, knowing that some parameter which has a tremendous impact on a task's load has changed would allow the load balancing algorithm to anticipate that change ahead of time rather than responding to it after the fact. Load metrics are updated periodically. As the modules migrate to and from a node, the node's load matrices are updated with an estimate of the migrating module's load.

Coordinator node: a node with good connections to other nodes can be used as a coordinator node. But generally any node in the system can be used as a coordinator. Coordinator node contains information about load metrics of each node. The main function of a coordinator node is to maintain the current load state of each lightly loaded node in the community. Each lightly loaded node needs to send its current load state periodically to the coordinator. The coordinator itself maintains a table to hold incoming valuable information from all nodes.

Scalability: the advance task migration algorithm is scalable in the sense that in large system with many nodes the system can be portioned into group. Each group can have its own coordinator and nodes are paired within the group. First a light weighted node is checked in same group, if suitable node not found then after nearby group is searched and after getting a required node transfer takes place if protocol is satisfied for load transfer.

Overloaded: the overhead is associated with the advance task migration algorithm includes the load information sent to the coordinator node and the reply message sent to each source node. Due to network congestion the actual migration of module is likely to be expensive and difficult.

6. COMPARISON WITH OTHER ALGORITHM

Cooling [6] the cooling algorithm basic concept is similar to advance task migration algorithm on the surface; the difference is in which order module are chosen. The advance task migration algorithm first sorting node and module then finds (ns,nd) pairs, and then attempts migration within each pair In the cooling algorithm, source node is largest overloaded node and largest module on this node is considered for migration.

The destination node is the underloaded node that can lodge the source module. In advance algorithm many migrations are performing parallel, it is difficult in case of cooling.

Hot spot[6] the hot spot algorithm is similar to cooling, with addition of looking at estimated queuing latency of an activity when deciding what to migrate. The module that gives the largest latency to an activity is grated to a node where it's added estimated latency is less than on the source node.

Match making algorithm[7] is similar to the advance task migration algorithm. However there is a main difference between them. The Match-Making algorithm arranges the imbalance node in decreasing order of load while Advance task migration algorithm arrange the imbalance node in increasing order of their load (i.e. the difference of a node's load from its reference load (w-ref)). This gives better result then other algorithm.

7. CONCLUSION

An advance task migration model has been formulated for dynamic load balancing of a distributed computing system in the context of load balancing. Advance task migration algorithm discuss in this paper 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 advance 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. Some constraints on advance task migration algorithm are load balancing is currently implemented in a barrier. This is not a necessity. Introducing asynchrony into the load balancing process would allow its cost to be overlapped with idle time on under-loaded computers and would not disrupt applications that do not have algorithmic synchronization points. Task-based load balancing strategies fail whenever the load of a single job exceeds the average load over all computers. No matter where such a job is moved, the node to which it is mapped will be overloaded. By dividing the task through routines such as **node split()**, one can alleviate this problem by providing viable work movement options. In general, adoptions can be used to dynamically manage the granularity of a computation so as to maintain the best number of tasks, increasing or decreasing the available options as necessary.

8 . REFERENCES

[1] Mor Harchol-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 for Distributed 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] D. Grosu, A. T. Chronopoulos, and M. Y. Leung, "Load balancing in distributed systems: An approach using cooperative games," *Proc. 16th IEEE Int. Parallel Distributed Processing Symp.*, pp. 52–61, Apr. 2002.
[9] www.wikipedia.com
[10] Various sources over the Internet.