Abstract:

In this paper we formulate the Dynamic load balancing and static load balancing problem in distributed systems. Static load balancing method tasks are allocated to the processors on compile time and ones they are allocated, there can be no changes further at run time. Dynamic Load balancing (DLB) is method, under which allocation of tasks, to different processors, is done at run time. Dynamic policies take their decisions on the current state of the system. They are more complex than static policies.

Keywords—Load Balancing, Distributed System, Distributed computing, optimal scheduling, cluster, DLB, SLB.

I. INTRODUCTION

Distributed System is the computer network consists of heterogeneous processors and database which works together to accomplish some common goal. In distributed computing environment, each job is divided into numerous individual tasks assigned to various processors for accomplishment of a common goal or job. Each tasks allocated to the particular processor increases the load on the processor where it is to be executed. Every processor consists of queue where all coming tasks are queued and waits for its turn for execution.

In distributed computing various tasks are allocated and executed on different processors. There are many factors which are considered while execution of any job such as, load balancing, processor speed, network speed, memory allocation, starvation, etc. Memory sharing and load distribution is one of the crucial ones among them.

In distributed system and parallel system main difference is usage of memory. In distributed system each processor has its own memory whereas, in parallel system each processor uses the shared memory. This means in distributed computing, for every tasks all data is saved locally and then executed. Whereas, both are similar in another way as uses various processors for execution of some common job/goal and task is assigned to all processors, accordingly.

In distributed environment, one aspect is memory used which can be centralized or distributed database where memory allocation is considered as how data is assigned to multiple tasks. Another aspect of this is processing or execution of the tasks.

In distributed computing, which task will be allocated to which processor and its execution time and response time is considered. Tasks represent loads, allocated to numerous heterogeneous nodes or processors in a network. In distributed system, load balancing is referred for allocation of tasks to different processors. Tasks allocated are independent of each other and executed according to the order they are assigned to processor and stored in its queue. Distributed system consists of distributed load balancers. Distributed load balancer consists of various types of processor, memory, and speed of the network. It allocates and balances the load among various processors for optimum resource utilization and minimum response time.

Quicker job completion can be accomplished, either by increasing the size of computing nodes, i.e.; by increasing the computational power of each node, or
effectively by distributing the load among multiple existing nodes based on their functionality or capabilities. Each node in the system can have different order of service capabilities based on its characteristics such as; computing performance and their reliability.

Optimum load balancing algorithm leads to optimized resource utilization and throughput enhancement. It reduced response time, as well as evenly distributes the load among various nodes to avoid overloading problem. Keeping back-up, on another server, of all tasks and its data residing on different nodes increases the reliability, as in case of one machine crash, the whole system can still survive.

Load Balancing: Load Balancer is essentially based on two characteristics. First, assigning load to the best candidate node i.e., node capable of respective functionality. The second is to make heavy loaded nodes lightly loaded by reallocating their task to other lightly loaded nodes, which are idle or have lesser load. Load balancer’s main task is to determine the load on each node and to allocate the different tasks to different nodes, based on their current loads and other factors.

There are various benefits of load balancing such as: it improves overall performance of the system by reducing idle time of all nodes and by maximizing the utilization of resources. It provides higher throughput and reliability, at low cost. Small jobs do not suffers from starvation through various preemptive and non-preemptive techniques which are applied internally, within various nodes and externally, as between various nodes.

There are basically two types of load balancing methods as shown below in Figure1.

Static load balancing method tasks are allocated to the processors on compile time and ones they are allocated, there can be no changes further at run time. As names specifies, in SLB, processes or tasks are allocated statically instead of dynamic allocation. Here, allocation of tasks occurs based on its prior information and various factors such as; mean execution time, IPC (inter-process communications), incoming time and extent of resource needed by it.

Dynamic Load balancing (DLB) is method, under which allocation of tasks, to different processors, is done at run time. Dynamic policies take their decisions on the current state of the system. They are more complex than static policies. In DLB, at runtime, load is transferred from highly loaded nodes to lightly loaded nodes and likewise load is balanced among all available nodes and approximately at same time all processors get into idle state.

Dynamic load balancing consists of the following steps:-

1. Initiation
2. Load selection
3. Information Exchange
4. Load balancer location

In case of reallocation of tasks there is the constant need of load monitoring system. This increases the overhead and makes the system more complicated. In DLB, at run time allocation, distribution of tasks and
its reallocation increases and this results in increased overhead and less stability in comparison to static algorithm.

But DLB is more preferred than SLB due to some reasons such as; frequent reallocation of tasks which reduces idle time, minimizes response time and maximizes resource utilization. The idle time of different nodes reduces and more quickly job completion is accomplished. In DLB, in less time more jobs are accomplished, being more economical than SLB. Dynamic load balancing makes the decisions based on the current state of the system gymnast.

Essentially, in real-time DLB, evenly distribution of load across the system is a complex task. There can be different obstacles while allocating each and every individual tasks or processes to different processors or nodes, for this a smart process for the specific type of node is

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Information Exchange</th>
<th>Processor Matching</th>
<th>Load Matching</th>
<th>Communication</th>
<th>Applications</th>
<th>LB Locations</th>
<th>Initiation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SASH</td>
<td>Global</td>
<td>Processor will produce the fastest turn around</td>
<td>Cost Function</td>
<td>Randomized Global</td>
<td>Independent tasks</td>
<td>Centralized (dedicated)</td>
<td>Event driven (Shortest execution time)</td>
</tr>
<tr>
<td>DLB Local/Global</td>
<td>According to Load Balancer</td>
<td>Cost Function which uses past prediction to future</td>
<td>Randomized Global</td>
<td>Independent Loops</td>
<td>Centralized / Distributed</td>
<td>Receiver initiated</td>
<td></td>
</tr>
<tr>
<td>Automatic Heterogeneous Supercomputing</td>
<td>Global</td>
<td>According to Load Balancer</td>
<td>N/A</td>
<td>Randomized Global</td>
<td>Whole programs</td>
<td>Centralized</td>
<td>Event Driven (User)</td>
</tr>
<tr>
<td>Direct Neighborhood Repeated (DNR)</td>
<td>Local</td>
<td>Least loaded processor in the neighborhood of the receiver</td>
<td>If the load difference exceeds a threshold, a percentage of the load is sent.</td>
<td>Uniform Local</td>
<td>Independent Loops</td>
<td>Centralized / Distributed</td>
<td>Receiver initiated</td>
</tr>
<tr>
<td>Neighbor</td>
<td>Local</td>
<td>Adjacent processors</td>
<td>Load is distributed over the nodes of the island.</td>
<td>Uniform Local</td>
<td>Independent tasks</td>
<td>Distributed Asynchronous</td>
<td>Receiver initiated</td>
</tr>
</tbody>
</table>

Classification of various DLB Algorithms:
required to assign a task and DLB provides that functionality. 

Parameters classification: Centralized vs. distributed

In centralized strategy, all the nodes are connected to single centralized node which makes the decision whereas, in distributed load balancing strategy, the load balancer is replicated on all workstations where load information of each node is broadcasted and performs load balancing decision based on that shared information by themselves, as shown in Figure 2.

The issues regarding these strategies are: In centralized load balancing strategy, limited scalability and on failure of centralized load balancer bottleneck problem arises whereas, in distributed strategy the overhead and congestion across whole network increases, rapidly. Proper planning can help to solve these problems.

II. RELATED WORK

Load balancing is a challenging problem in distributed system. Load balancing strategies can be divided into two broad categories first one includes those applications where new tasks are developed at regular basis, no certain pattern is followed. Hence, dynamic allocation of tasks is done here. Second one includes iterative applications with permanent load patterns. This uses generally static allocation as all conditions are pre-known, but can also use dynamic approach.

2.1 A load balancing approach with centralized Load Balancer and two back-end servers.

The design illustrated in Figure 3 with one Load Balancer communicating with all the nodes and monitoring their load is proposed in [2]. This load balancer reports the load to two backend servers. The servers finally make the balancing decision and return the address of the suitable node to which the overload should be transmitted. Additional servers are used for the sake of reliability. If one server fails, the other can take its job and the system continues to work properly.

The central load balancer has three parts: Load monitoring server, Load reporting server and load balancing library, where the two back-end servers comprises of only Load monitoring server and Load reporting server.

The Load Reporting Server is used to collect the machine load information on which it is running. The collected data is sent to Load Monitoring Server which is located on same machine. Load Monitoring Server stores the collected data in data structure. Then an all-to-all broadcast of the load information is carried out. Then, at the central load balancer, when request comes, the Load Balancing Library finds least loaded machine and return the address of that machine.

The drawback of the design is the huge communication overhead involved as it follows a global strategy of load balancing. Secondly the cost

![Figure 2. The proposed strategy representing all-to-all broadcast load information.](image-url)
3.2 A load balancing approach using one supporting node with each primary node and using a priority scheme to schedule tasks at supporting nodes.

Let us now have a look on another approach proposed in [3], it uses one supporting node (denoted as SNi) with each primary node (denoted as Ni) as depicted in Fig. 3. In case of overload at node Ni, an interrupt service routine generates an interrupt and the overload is transferred to its supporting node and it also uses a priority scheme, if the priority of the incoming process at the supporting node is greater than that of the currently running process, then the current process is interrupted and assigned to a waiting queue and the incoming process is allowed to run at the supporting node. Otherwise, the current process continues and incoming process is in waiting state until the current process is completed.

![Fig.3: Proposed design for Load Balancing using multiple supporting nodes [3]](image)

This approach has a drawback of its complexity and the cost of such a huge infrastructure. The priority scheme makes it more dynamic and suitable for distributed systems as well as handling real time tasks.

Traditional load balancing techniques are not efficient enough to cope up with today’s requirements and to provide a healthy environment. Hence, various load balancing techniques are evolved and now also many modifications are going on. The paper presents and discusses the various techniques of load balancing algorithm and a new design, to perform dynamic load balancing and semi-distributed approach which uses a package offer. The proposed solution is generally based on local load conditions where, weights of all the neighbor computing nodes are calculated and the best one is selected.

**III. PROPOSED WORK**

The proposed design is a slight extension and modification to that proposed in [1]. It uses a clustered approach in which each cluster maintains three nodes and each cluster has a supporting node (as shown in Fig. 4). The load balancer maintains a queue for each cluster to store the load of its nodes. This reduces the cost of infrastructure used in the design proposed in [3], and improves the service offered by [2] by using clusters rather than individual nodes.

It uses a threshold approach to decide whether a node is heavily loaded or not. The Load Balancer has three parts: Load Monitoring Server (LMS), Load Reporting Server (LRS) and Decision Making Server (DMS). The Load Monitoring Server and the Load Reporting Server have similar tasks of calculating and collecting the system load information.

The decision making server runs only whenever some node is overloaded and finds the most appropriate node to which the overload is to be transferred.
In Centralized approach there is single node, so process the load at high speed by using switching but still a limitation is there. An approach is there to remove the limitation is to split the centralized node into small nodes called supporting nodes (SNs). But still here supporting node are not allotted load initially. Many times supporting nodes is idle or they are not properly loaded as only overload is assigned to supporting nodes. This is wastage of power of supporting nodes. We can also use the free time of SN by making them busy for this free time. So a further approach is developed here in which supporting nodes are given some load initially and SNs maintain a priority list of process or order in which the process at the SN will execute. Suppose a process Pi is currently executed by SN and a Primary node Ni is overloaded so that it finds a supporting node SNisuitable for transferring its overload, so Ni will interrupt the SN i, then SNi will assign Priority to the coming process and call the interrupt service routine to handle the interrupt.

Interrupt Service Routine actually compares the priority of each coming process with the currently executing process and perform the switching between the currently executing process and process coming from the primary nodes. Otherwise, each supporting node is maintaining a priority queue in which process to be executed are sorted according to the priority, in which coming process are stored in this queue with a priority.

IV. CONCLUSION

The presented design and algorithm works well for distributed systems and ensures that no process suffers starvation and no processor is overwhelmed. The presented design works for cluster with number of nodes=3. The proposed design has several advantages over the referred designs comprising reduced communication overhead, semi-distributed architecture, dynamic approach to load balancing and reduced cost and complexity.

For further research, it would be a significant task to design a modified approach to work for ‘n’ number of clusters with ‘n’ number of nodes. We can apply a dynamic priority scheme and also add some functionality to make it work for heterogeneous networks. The optimization of the algorithm is another task for future research.

Decentralized Load Balancing Method:

Decentralized resource allocation algorithms have been used to design polynomial time algorithms for intractable problems that provide solutions within the bounded proximity of the optimal solution. The load balancing problem in heterogeneous distributed system is modeled as a multi-player non-cooperative game with Nash equilibrium. The decisions to allocate the resources in an HDCS are based upon the pricing model
of computing resources using a bargaining game theory. In the process prior to executing a task, the heterogeneous computing nodes are participating in a non-cooperative game to reach equilibrium. The non-cooperative framework adopted to allocate tasks by m servers is modeled as m? Player game. We have evaluated the performance of two existing price-based job allocation schemes, namely the Global Optimal Scheme with Pricing (GOSP) and Nash Scheme with Pricing (NASHP). A modified version of each of these schemes has been introduced to analyze the performance by considering the effect of pricing on system utilization.

Distributed system provides support for powerful computing infrastructures to solve computationally demanding problems. These computing resources are spread over the globe with different independent administrative domains, where the ownership of computing nodes are with individuals or organizations. Grids and clouds are such systems with highly scalable computing infrastructures, which are regularly increasing with an increase in the user base. The ownership from distinct individuals or organizations requires a de-centralized resource management system. Problems with high computing requirements are most suitable to use a distributed computing infrastructure.

In the decentralized approach of load balancing, all of the computing nodes in the HDCS are involved in taking the load balancing decision. The load balancing decisions are based on the dynamic state of information of the whole system. Game theory has been used by the researchers to model a situation where at least two interactive decision makers with diverging interests are involved [172]. Interactive decision making involves players (nodes) as the decision makers. In particular, the node Mj has a goal to minimize its load Aj by transferring a part of the load to the other computing nodes in the HDCS. This load transfer process may lead to a situation where loads on the nodes are balanced. This results in equilibrium when the node has any incentive to transfer its load to the other nodes. The state at which load exchange between any two of the computing nodes stops and tasks are executed without interruption at arriving nodes is termed as the Nash equilibrium state. Classical game theory is a normative theory, in the sense that it expects players or agents to be perfectly rational and behave accordingly. In classical game theory, interactions between rational agents are modeled as games of two or more players that can choose from a set of strategies and the corresponding preferences. The game theory assumes that players will compute Nash equilibrium and choose to play one such strategy. Application of algorithmic game theory can be found in which lists several applications including networking and artificial intelligence along with basics of game theory. Resource allocation problems can be modeled as cooperative or non-cooperative games in heterogeneous environments as suggested in Cooperative game theory offers formal models to provide axiomatic solutions through a situation where an enforceable binding agreements between each pair of decision makers (players) are possible. Game theoretic algorithms are designed to achieve convergence to the Nash equilibrium by modeling the related information. They assume that the players.
can observe the actions of the other players. Moreover, the decision makers have complete freedom of preplay communication to make joint agreements about their operating points.

REFERENCES


