IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661,p-ISSN: 2278-8727, Volume 18, Issue 3, Ver. VI (May-Jun. 2016), PP 21-26
www.iosrjournals.org

Balancing Load in Computational Grids: A New Approach

[1]Debashreet Das, [2] C.R.Tripathy,
[1,2] Dept. of Computer science and Engineering, Veer Surendra Sai University of Technology, Burla, Odisha, India,

Abstract: *The emergence of grid computing over the internet needs a hybrid load balancing algorithm which can take into account the various characteristics of the grid computing environment. Hence, this paper proposes a load balancing strategy called Match-Maker Algorithm, which takes into account the grid architecture, resource availability and job characteristics. More focus has been given to module migration algorithm in an automated dynamic manner for the problem of allocating software modules to the processing nodes.*

Keywords: *Grid computing, Load Balancing, Reference optimization problem*

I. Introduction

The automatic management of distributive system leads to the efficient allocation of software to the processing elements. However, some of the important objectives which need to be achieved for the allocation procedure are the avoidance of unwanted communication, usage of the available resources[1], maintaining the security policies as well as the reliable policies, etc. In addition, Load-balancing can be better defined and explained using the following five policies[2]:

1. *Information Policy*: Indicates the type of workload information to be collected, when and where to be collected.
2. *Triggering Policy*: Determines the approximate starting time for a load-balancing operation.
3. *Resource-Type Policy*: Specifies whether a resource is a server or receiver of tasks, depending on the status of availability.
4. *Location Policy*: Utilizes the output provided by resource type policy for finding a suitable match either for a resource provider or a resource receiver.
5. *Selection Policy*: Specifies the tasks for migrating from overloaded resources to the under-loaded resources.

In addition to these policies for load-balancing, a distributed system also contains policies including scheduling the loads and fault-tolerant policies as well as for utilization of resources.

II. Related Work

The distribution and balancing the loads in parallel and distributed systems in a broad way has been studied by B.A. Shiraziet. al[4]. In addition to that, the conundrum of most favorable distribution of loads using queueing models with various parameters related to performance such as weighted mean response time, arithmetic average response time, etc. has been studied by H.Kameda[5] .

However, as compared to homogeneous systems, the distribution of load at an optimal level for heterogeneous and distributed systems having both consecrate and generic applications is studied by F.Bonomi and A.Kumar[6], K.W.Ross and D.D.Yao[7]. In the past, an attempt has been made for developing the load-balancing algorithms that are de-centralized with the objective of diminishing the needs for communication and is the outcome of de-centralization of these algorithms. In addition, Dandamudi[8] classified the load balancing mechanisms into static and dynamic. Some of the conventional static load-balancing mechanisms are the speed-weighted random splitting and the load-dependents static mechanisms. According to Dandamudi and Braun , the activities are dispensed mathematically to different nodes in a de-centralized system, so that each resource can complete the activities as desired. Moreover, the system has a unwavering degree of threshold on the performance of the system.

Dandamudi[8] also proposed the concept of balancing the load dynamically where the system time may be either present in the current state or in the most recent state. This mechanism is used for deciding the way in which the activities can be allocated to every resource in a de-centralized environment. In case the system is heavily loaded, the dynamic mechanism relocates the loaded activities to other nodes and gets them executed. When the overhanging caused due to relocation is controlled and gets bounded within a certain range, the dynamic load-balancing mechanism becomes preferable over static balancing.

According to Grosu and Chronopoulos[9] and Penmatsa and Chronopoulos[10], the loads among the systems are balanced by the servers using the mechanism of round-robin. This mechanism depends on gathering of information regarding the status of all systems by the server including the loads that are assigned to each system by other servers, in case of a single-user. The case of multi-users was considered as an extension by Penmatsa and Chronopoulos by taking communication delay as a parameter. However, Dhakal et.al.[11] thoroughed about dynamically balancinn the loads in distributed computing systems, thereby developing a stochastic model considering randomness in delay, but ignoring the presence of resources in different administrative domains.

J.Cao[12] and Junwei Cao et. al.[13] in their articles, suggested an ant-like self-organizing policy in order to achieve load balancing in a wide network of Grids by collecting local interactions among the nodes of the grids. In this miniature, a number of resource management agents collaborate in order to achieve automatic load-balancing of distributed job queues, where each ant requires two sets of "X" number of steps continuously as for deciding upon which is the least and which is heavily loaded nodes. These two nodes, in turn, again distribute the loads between themselves. After a sequence of contiguous re-distribution of loads among the networked nodes, a consistent load-balancing is obtained.

However, Yagoubi and Slimani[14] suggested an algorithm based on layered structure so that the dynamic load-balancing mechanism in a grid environment could be achieved. The suggested model is based on a tree model and includes the features related to scalability and heterogeneity, without taking into consideration, the physical architecture of the Grids. Along with, S.Ludwig and A. Mallem[15] presented two new load-balancing algorithms that was motivated from distributed swarm intelligence, where one algorithm finds its basis on ant colony optimization and the second one from particle swarm optimization. Yongsheng Haoet.al[16] suggested a load-balancing mechanism considering the deadline. They have proved that the mechanism, as suggested, reduces the completion time as well as the time for re-submission. However, Shah et. al[17] made a thorough study on dynamic load-balancing in grid systems by estimating the job arrival rate, processing rate and processor load, without taking into account the distributed architecture along with the method for optimal load balancing among distributed computing systems.

In addition, Arora et.al[18] also made an observation on dynamic load balancing in grid systems, ignoring the uncertainty of resources and also the method of optimal load balancing among distributed computing systems. Moreover, Anand, Ghosh and Mani[19] suggested a de-centralized, dynamic load-balancing algorithm(ELISA), thereby eliminating the problem of continual information interchange by assessing the load, depending on information about the state of the system that is obtained at huge intervals of time. The algorithm was drafted with the aim of diminishing communication delays by diminishing the requirement for interchange of status. However, Arora et. al[18] suggested an elevated, sender-initiated and scalable algorithm for scheduling the jobs and balancing the loads of the resources in a heterogeneous and distributed Grid environment.

The emergence of grid computing over the internet needs a hybrid load balancing algorithm which can take into account the various characteristics of the grid computing environment. Hence, this paper proposes a load balancing strategy called Match-Maker Algorithm, which takes into account the grid architecture, resource availability and job characteristics. More focus has been given to module migration algorithm in an automated dynamic manner for the problem of allocating software modules to the processing nodes

III. Problem Formulation and Analysis

In this paper, we have discussed module migration algorithm in an automated and dynamic manner for the problem of allocating software modules to the processing nodes. The module allocation algorithm is NP-Complete and belongs to a family of problems with a different formulation. The common idea is to find an optimal allocation of modules according to a given objective function. However, the module allocation algorithm is not free from overhead cost, wherein, the systems using static module allocation suffer from overheads. Besides this, running the algorithm is found to be time consuming. If the load situation in the network changes, then modules must migrate as they would in the case of the migration algorithm.

The following terms are defined in relation to our problem formulation:

1. ***Model:*** It is the distributed module based system to be used as a reference model in case of migration algorithms. Moreover, a system's performance depends upon the interaction between the hardware and software, where, models for both hardware and software are presented in an distributed system where communication software modules are to be allocated to physical processing nodes in a network.
   Physical resources in the model are represented by a collection of processing nodes, denoted by "N", where, the resources are considered to be heterogeneous. Due to varying computational facilities at each node, a particular module incurs different costs when executed on different processors.
2. ***Load:*** In the model, the term "load" is defined as the workload generated by the software modules running on physical processors with the assumption that the load of a module running on a processor can be measured. It is also claimed that delay becomes extremely large due to high load on a network.

3. ***Module*****:** The software in the system is represented by “M” modules. A module is an atomic entity such as object or component that can be migrated from node to node. The execution load for a module is denoted by “N *M” matrix “E”, where, the element “e_{ij}” is the execution load for module “j” allocated to node “I”. e_{ij} = if module “j” may not be allocated to node “I” due to security or domain membership reasons. Modules interact using an RPC –like protocol in this communication model, where the actual communication cost between two modules is zero if they are co-allocated on the same node. Otherwise, the communication load is given by an “M*M” matrix “C”[0][0].

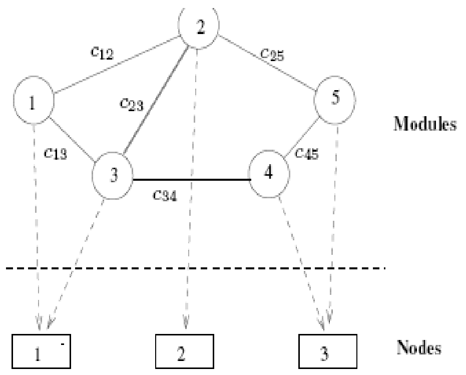

Fig : Model of the system

Reference Optimization Problem-

The main objective of Module allocation Problem is to allocate modules to physical Nodes. The main conflict of objectives for performance is between clustering and distribution of modules. As we have to study the load balancing using Module Migration , We choose an objective that minimizes the load imbalance in the system.

First, let the the workload wi for node i given allocation α be:

$$wi(\alpha) = \sum_{J=1}^{M}(ej\alpha ij + \sum_{k=1} cjk\ \alpha ij(1-\alpha ik)) \text{ -equ (1)}$$

Our objective is to minimize the system load imbalance. This means total load on node i should be as close as possible to the average load wref;i for node i given allocation α. In our notation, the load imbalance L given allocation α is

$$L=\sum_{i=1}^{N}\ \sum_{j=1}^{M} \left| wi(\alpha)- wref;i(\alpha) \right| \text{ -----------equ(2)}$$

wherewref;i(α) is the average load of the nodes weighted by node capacity. Each node may not be allocated more than it's capacity

$0 \leq wi< 1:0;\quad i = 1....N$ ---------------------equ(3)

The optimization problem is to find the α that minimizes L subject to equ (3).

The Match-maker algorithm:

The objective of the Match-maker algorithm is to balance the load on nodes and minimizes the load imbalance. The balancing operation is performed by matching nodes with high loads with nodes with low load and the migration will take place between the pairs.

```
Step-1:Node_ list .sort ( )
Step-2:for each n in node_ list:
 Step-3: n. module_ list .sort( )
Step-4: i = 1
Step-5:k=node_ list .length( ) (N)
Step-6:whilei< k:
Step-7:ns=node_ list(i)
Step-8:nd=node _list(k)
```

```
Step-9:if ns.load>ns.wref and nd.load<nd.wref :
Step-10:j = 1
Step-11: while j ≤ ns.module_ list.length( ):
Step-12:m=ns.module _list(j)
Step-13: if nd.can accommodate(c) and
Step-14: nd.load + m.load<nd.wref :
Step-15:migrate(ns,nd,m)
Step-16: else:
Step-17:j = j + 1
Step-18:i = i + 1
Step-19:k = k - 1
```

Each pair has one node .High load called the source node ns and low load called the destination node nd. Here module migrated from source node to destination node. The algorithm works in intervals of length T. The system is measured during the interval load. The matching operation is performed at a centrally located *coordinator* node collects load information and then redistributes it toother nodes.

The matching procedure is made by load imbalance order: The first pair has the most overloaded node and the must under-loaded node, the second pair has the second most overloaded node and the second most under-loaded and so on.

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(mod)* function handles non-performance related allocation policies, such as security or reliability.

Comparison with Other Algorithms:

Here we will discuss some load balancing algorithms that are published earlier and will compare them theoretically with our algorithm.

Cooling Algorithm

The Cooling Algorithm as suggested by V. Kalogeraki, P. M. Melliar-Smith and L. E. Moser[20] of Department of Electrical and Computer Engineering, University of California in their paper Dynamic Migration Algorithms for Distributed Object Systems has the following pseudo code:

```
Cooling algorithm ()
Find overloaded Processor p with loadp> HIGH
While loadp> HIGH
For all objects i of processor p in decreasing object_loadi,
Find processor q to host oject I using best-fit allocation
If (object_loadi + Loadq< Min (HIGH, loadp))
Move object i from processor p to q
Update processor_name of object i
Loadp =loadq - loadi
Loadq = loadq+object_loadi
```

Upon a thorough study, the following major differences were found between Cooling algorithm and Match Maker Algorithm:

SL NO	Match Making Algorithim	Cooling Algorithm
1	The Match Making Algorithm first finds (ns; nd) pairs (after sorting the nodes on the basis of Load Imbalance) and then attempts migration within each pair.	The most heavily loaded node is always the source node, and the module with highest load on that node is the primary candidate for migration.
2	Destination Node depends on the pair chosen.	The destination node is the least loaded node that can accommodate the candidate module.
3	Several migrations can be made in parallel.	More difficult in the case of the Cooling algorithm.
4	The source node knows what modules it has and is likely to be able to choose a good candidate for migration.	The Cooling algorithm requires all lists of module loads to be submitted by nodes at end of every interval.

Factors Restricting Cooling Algorithm's Software Simulation:
Following factors affect on the selection of overloaded processor (node) which makes it practically infeasible to simulate on software:
1. Actual computation and execution times of the methods of the objects
2. The usage on the processor's resources during the executions.
3. Parameters of the invocations, the current state of the invoked and invoking objects and the load of the processors.
4. Frequency of measurements.
5. Processing time of the method to execute locally on the processor.
6. Time the method spends on the Scheduler's queue waiting for the CPU to be released.
7. The order with which the methods will be executed depends on the other objects currently scheduled or queued on the processor.
8. The proportion of the processing load, the amount of memory and the disk bandwidth required for executing the method locally on the processor.
9. HIGH limit of a particular processor.
10. Any physical constraints on the objects. For example, a multimedia object that grabs live images from a camera is tight to a specific processor and cannot be reallocated.
11. Importance metric of the tasks to decide which task to remove from the system.

Normal Algorithm[21]:
This algorithm is another deviation of the Match Making Algorithm. The algorithm was developed in the process of developing the Match Making Algorithm.
In the Normal algorithm, we follow almost same steps as that in the Match Making Algorithm with the exception that we do not sort the nodes and modules before migration. Thus the pseudo-code reduces to:

Normal Algorithm
```
Step 1: for each node n repeat step 3.
Step 2: Let I =1
Step 3: Let k be the total number of nodes
Step 4: While i<k repeat step 5 to 14
Step 5: Let ns represents the ith node.
Step 6: Let nd represents the k th node.
Step 7: if Load of ns >wref of ns and load of nd is <wref of nd then go to step 8 else go to step13.
Step 8: Let j=1
Step 9: While j <= number of modules allocated to ns repeat steps 10 to 12.
Step 10: Let m is the jth module of ns.
Step 11: If nd can accommodate a node and sum of the load of nd and m is less than wef of nd then migrate the module m from node ns to node nd.
Step 12: increment j by 1.
Step 13 increment i by 1.
Step 14: increment k by 1.
```

IV. Conclusion

As with any thesis, several ideas have come to light that may improve the performance of our system but have not been implemented. Some of these did not seem to provide additional benefit. In this chapter, we describe some of these ideas and our intuition as to whether their implementation would be beneficial.

References

[1]. Yagoubi B., Slimani Y, "Load Balancing Strategy in Grid Environment", Journal of Information Technology and Applications, Vol.1, No. 4, 2007, pp.285-296.
[2]. Yagoubi et.al., "Load Balancing in Grid Computing", Asian Journal of Information Technology, 5(10), 2006, pp. 1095-1103.
[3]. Patel D., Das D., Tripathy C.R., "A new Approach for Grid Load Balancing among Heterogeneous Resources with Bandwidth Consideration",
[4]. Shirazi B.A. et.al., "Scheduling and Balancing in Parallel and Distributed Systems", IEEE Computer Society, 1995.
[5]. Kameda et.al., "An Algorithm for optimal static load balancing in distributed computing systems", IEEE Transactions on Computers, 1992.
[6]. Bonomi F, Kumar A., "Adaptive Optimal Load Balancing in a non-homogeneous multi-server system with a central job scheduler", IEEE Transactions on Computers,1990, pp. 1232-1250.
[7]. Ross K.W., Yao D.D., "Optimal Load Balancing and Scheduling in a Distributed Computer System", ACM Digital Library, 1991, pp.676-689.
[8]. Abawajy H.J, Dandamudi SP., "Parallel Job Scheduling on multicluster computing system", IEEE International Conference on Cluster Computing", pp. 11-18.

[9]. Grosu D., Chronopoulous T.A.,Leung Y.M, ”Cooperative Load Balancing in Distributed Systems”, Wiley Online Library, Vol. 20,2008,pp. 1953-1976.
[10]. Chronopoulous T.A., Penmatsa S., “Job Allocation Schemes in Computational Grids on Cost Optimization”, International Symposium on Parallel and Distributed Processing, 2005, pp. 180a.
[11]. Dhakal S. et.al., “A Regeneration-Based Approach for Resource Allocation in Co-operative Distributed Systems”, IEEE International Conference on Acoustics, Speech and Signal Processing, 2007, pp. 1261-1264.
[12]. Cao J., “Self-Organizing Agents for Grid Load Balancing”, IEEE/ACM International Workshop on Grid Computing, 2004, pp. 388-395.
[13]. Cao J. et.al., “Grid Load Balancing using Intelligent Agents”, Issue 1, Vol. 21, pp. 135-149.
[14]. Yagoubi B., Slimani S., “Dynamic Load Balancing Strategy for Grid Computing”, No. 7, Vol. 2, 2008.
[15]. Ludwig S., Maollem Azin, “Swarm Intelligence Approaches for Grid Load Balancing”, Journal of Grid Computing,2011, pp. 279-301.
[16]. Hao Y. et.al. “An enhanced load balancing mechanism based on deadline control on Gridsim”, Future Generation Computer Systems, Issue 4, Vol. 28, 2012, pp. 657-665.
[17]. Shah R. et.al. “On the Design of Adaptive and De-centralized Load Balancing Algorithms with Load Estimation for Computational Grid Environments”, IEEE Transactions on Parallel and Distributed Systems, Issue 12, Vol. 18, pp. 1675-1686.
[18]. Arora M. et.al. “ A De-centralized Scheduling and Load Balancing Algorithm for Heterogeneous Grid Environment”, International Conference on Parallel Processing, 2002, pp. 499-505.
[19]. Anand L. et.al. “ELISA: an estimated load information scheduling algorithm for distributed computing systems”, Issue 8, Vol. 37, 1999, pp. 57-85.
[20]. Kalogeraki V. et.al. “Dynamic Migration Algorithms for Distributed Object Systems”,
[21]. Diasse A, Kone F., “Dynamic-Distributed Load Balancing for High performance and Responsiveness Distributed-GIS”, JGIS, 2011.