A Novel Algorithm for Load Balancing In P2P System

Najim Sheikh [1], Mtech RGPV[1], sheikhnajim4@gmail.com,
Dr. Sachin Choudhari[2] Principal, SBITM Betul[2] choudhari.sachin1986@gmail.com

Abstract: In this paper, we propose two efficient algorithms referred to as Rate-based Load Balancing via Virtual Routing (RLBVR) and Queue-based Load Balancing via Virtual Routing (QLBVR), which belong to the above RAP and QRAP policies. We classify the dynamic distributed load balancing algorithms for heterogeneous distributed computer systems into three policies: Queue Adjustment Policy (QAP), Rate Adjustment Policy (RAP), and Queue and Rate Adjustment Policy (QRAP). We also consider algorithms Estimated Load Information Scheduling Algorithm (ELISA) and Perfect Information Algorithm, which were introduced in the literature, to implement QAP policy. Our focus is to analyze and understand the behaviors of these algorithms in terms of their load balancing abilities under varying load conditions (light, moderate, or high) and the minimization of the mean response time of jobs. We compare the above classes of algorithms by a number of rigorous simulation experiments to elicit their behaviors under some influencing parameters, such as load on the system and status exchange intervals. We also extend our experimental verification to large scale cluster systems such as a Mesh architecture which is widely used in real-life situations. From these experiments, recommendations are drawn to prescribe the suitability of the algorithms under various situations.

Key Words: Algorithm, Balancing, Load,

I. INTRODUCTION:

Today a growing number of companies have to process huge amounts of data in a cost-efficient manner. Classic representatives for these companies are operators of Internet search engines, like Google, Yahoo, or Microsoft. The vast amount of data they have to deal with every day has made traditional database solutions prohibitively expensive. Instead, these companies have popularized an architectural paradigm based on a large number of commodity servers. Problems like processing crawled documents or regenerating a web index are split into several independent subtasks, distributed among the available nodes, and computed in parallel. In order to simplify the development of distributed applications on top of such architectures, many of these companies have also built customized data processing frameworks. Examples are Google's Map Reduce, Microsoft's Dryad, or Yahoo!'s Map-Reduce-Merge. They can be classified by terms like high throughput computing (HTC) or many-task computing (MTC), depending on the amount of data and the number of tasks involved in the computation. Although these systems differ in design, their programming models share similar objectives, namely hiding the hassle of parallel programming, fault tolerance, and execution optimizations from the developer. Developers can typically continue to write sequential programs. The processing framework then takes care of distributing the program among the available nodes and executes each instance of the program on the appropriate fragment of data. Static. A dynamic algorithm [1], [2], [3], [6],makes its decision according to the status of the system, where the status could refer to a certain type of information, such as the number of jobs waiting in the queue, the current job arrival rate, the job processing rate, etc., at each processor. On the other hand, a static algorithm [8], [11], [12], [13], performs by a predetermined policy, without considering the status of the system. Dynamic load balancing algorithms offer the possibility of improving load distribution at the expense of additional communication and computation overheads. In [4], [20], it was pointed out that the overheads of dynamic load balancing may be large, especially for a large heterogeneous distributed system. Hence, most of the research works in the literature focused on centralized dynamic load balancing [20], in which a Management Station (M-Station)/Scheduler kept checking the system status and scheduled the arriving jobs among the processors by some strategies, such as Backfilling, Gang-Scheduling, Migration [20], etc. By centralization, the M-Station/Scheduler can handle most of the communication and computation overheads efficiently and improve the system performance. However, centralization limits the scalability of the parallel system and the M-Station/Scheduler has turned out to be the system bottleneck due to the trend that distributed computer systems are becoming larger and more complicated. Compared with the centralized strategies, distributed dynamic load balancing offers more advantages, such as scalability, flexibility, and reliability, and thus has received more and more attention recently [1]. To realize a distributed working style, each processor in the system will handle its own communication and computation overheads independently [11]. In order to minimize the communication overheads, in [1], [10], some methods were proposed to estimate the status information of the nodes in the system and, in [9], [14], the authors analyzed how randomization could be used in the load balancing problem. To obtain optimal solutions among the systems, the computation overheads still remained high. For example, in [11], the Li-Kameda algorithm needed more than 400 seconds (approximately) and even a well-known FD algorithm [7] needed more than 105 seconds to solve a generic case. Such high computation overheads make it impossible for the distributed systems to obtain optimal solutions dynamically proposed an algorithm named LBVR and proved that the convergence rate of LBVR was super-linear. A high convergence rate can reduce the computation overheads significantly. For instance, in most cases, the LBVR algorithm can obtain an optimal solution of distributed systems within 0.1 seconds. In this paper, according to the job assignment methods, we classify the distributed dynamic load balancing

As our focus is to analyze and understand the behaviors of the algorithms in terms of their load balancing ability, minimization of mean response time, in our rigorous simulation experiments gain intuition we consider a single class of jobs for processing One of our added considerations regarding the relative metrics of the different approaches under consideration.

II. PROBLEM DEFINATION:

The goal of a cloud-based architecture is to provide some form of elasticity, the ability to expand and contract capacity on-demand. The implication is that at some point additional instances of an application will be needed in order for the architecture to scale and meet demand. That means there needs to be some mechanism in place to balance requests between two or more instances of that application. The mechanism most likely to be successful in performing such a task is a load balancer. A load balancer provides the means by which instances of applications can be provisioned and de-provisioned automatically without requiring changes to the network or its configuration. It automatically handles the increases and decreases in capacity and adapts its distribution decisions based on the capacity available at the time a request is made.

Because the end-user is always directed to a virtual server, or IP address, on the load balancer the increase or decrease of capacity provided by the provisioning and de-provisioning of application instances is non-disruptive. As is required by even the most basic of cloud computing definitions, the end user is abstracted by the load balancer from the actual implementation and needs not care about the actual implementation. The load balancer makes one, two, or two-hundred resources - whether physical or virtual - appear to be one resource; this decouples the user from the physical implementation of the application and allows the internal implementation to grow, to shrink, and to change without any obvious effect on the user. The right load balancer at the beginning of such an initiative is imperative to the success of more complex implementations later. The right load balancer will be able to provide the basics required to lay the foundation for more advanced cloud computing architectures in the future, while supporting even the most basic architectures today. The right load balancer will be extensible.

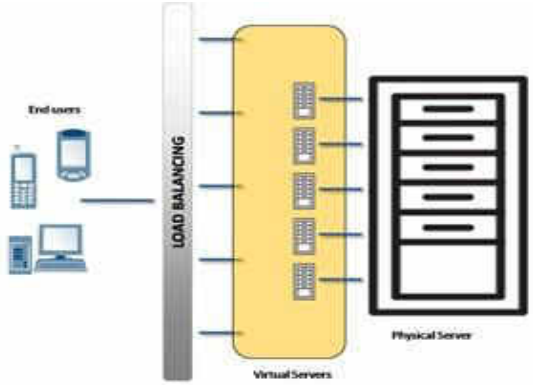

Figure 1.1: A Typical Load Balancer used in Cloud Computing

Load balancing ensures that all the processor in the system or every node in the network does approximately the equal amount of work at any instant of time. This technique can be sender initiated, receiver initiated or symmetric type (combination of sender initiated and receiver initiated types). Our objective is to develop an effective load balancing algorithm maximize or minimize different performance parameters (throughput, latency for example) for the clouds of different sizes (virtual topology depending on the application requirement). The system will take the service as input which is requested by the cloud customer and then the role of load balancer begins. It will communicate with the clients present in its network and assign the load to the least loaded one.

III. METHODOLOGY:

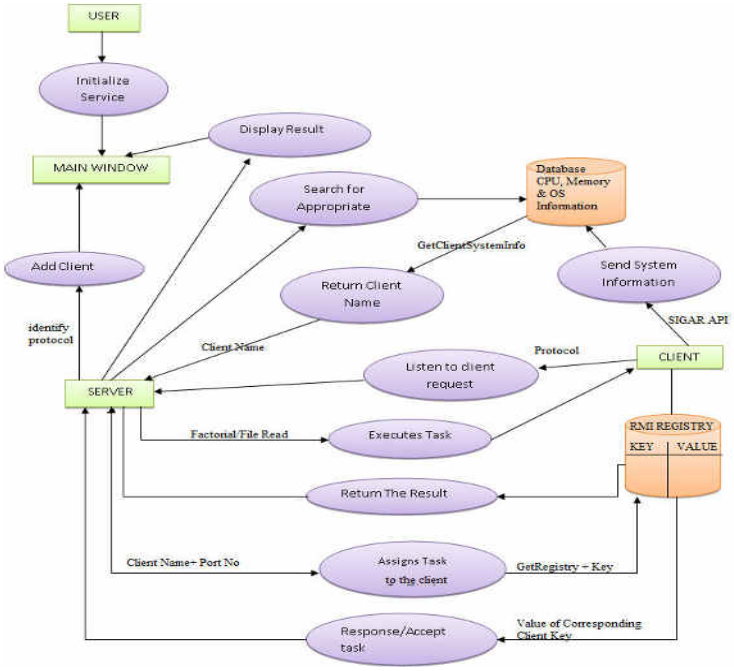

Figure: 1.2 the Data Flow Diagram of the System.

First present a general system model in the design of the algorithms. For convenience, we use “node” and “processor” interchangeably in the rest of this paper. We consider a generic parallel/distributed system shown in Fig. The system consists of n heterogeneous nodes, which represent host computers having different processing capabilities, interconnected by an underlying arbitrary communication network. Here, we use N to denote the set of nodes, i.e., n = [N], and E to denote a set whose elements are unordered pairs of distinct elements of N. Each unordered pair e = (i, j) in E is called an edge. For each edge (i, j), we define two ordered pairs, (i, j) and (j, i), which are called links, and we denote L as the set of links. A node i is said to be a neighboring node of j if i is directly connected to j by an edge. For a node j, let denote a set of neighboring nodes of node j. We assume that jobs arrive at node i (i N) according to an erotic process, such as inhomogeneous Poisson process with intensity function [15].

IV.THE SYSTEM MODEL AND CLASSIFICATION OF DYNAMIC LOAD BALANCING ALGORITHMS:

We first present a general system model in the design of the algorithms. For convenience, we use "node" and "processor" interchangeably in the rest of this paper. We consider a generic parallel/distributed system shown in Fig. 1. The system consists of n heterogeneous nodes, which represent host computers having different processing capabilities, interconnected by an underlying arbitrary communication network. Here, we use N to denote the set of nodes, i.e., n =[N], and E to denote a set whose elements are unordered pairs of distinct elements of N. Each unordered pair e =(i , j) in E is called an edge. For each edge (i , j), we define two ordered pairs, (i , j)and (j , i), which are called links, and we denote L as the set of links. A node i is said to be a neighboring node of j if i is directly connected to j by an edge. For a node j, let denote a set of neighboring nodes of node j. We assume that jobs arrive at node i (i N) according to an erotic process, such as inhomogeneous Poisson process with intensity function [15]. A job arriving at node i may either be processed locally or transferred through the network to another node j (j N) NÞ for remote processing. The service time of a job is a random variable that follows an exponential distribution with mean 1/μi, where μi denotes the average job service rate of node i and represents the rate (in jobs served per unit time) at which node i operates when busy. The queue discipline of the jobs in each node is FCFS and the buffer size is infinite. We denote βi(t) as the rate at which the jobs are processed at node i at time t. Once a job starts to undergo processing in a node, it is allowed to complete processing without interruption and cannot be transferred to another node in the meanwhile. In this model, we assume that there is a communication delay incurred when a job is transferred from one node to another before the job can be processed in the system and denote xij(t) as the job flow rate from node i to node j (j Vi) at time t. Further, we assume that each link (i , j) can transfer the load at its own transmission capability (otherwise referred to as transmission rate, commonly expressed as bytes/sec). We denote C as the set of transmission capacities of all the links and cij as the transmission capacity of a link (i , j) cij C . There are many communication delay models proposed for data networks in the literature. In our model, we assume that the communication delay functions can be any increasing, convex, and differential functions [11], and, for ease. Of simplicity, here we choose M/M/1 as the communication delay model [11],[16].

Fig.2. (a) Node model of queue adjustment policy. We first present a general system model in the design of the algorithms. For convenience, we use "node" and "processor" interchangeably in the rest of this paper. We consider a generic parallel/distributed system shown in Fig. 1. The system consists of n heterogeneous nodes, which represent host computers having different processing capabilities, interconnected by an underlying arbitrary communication network. Here, we use N to denote the set of nodes, i.e., n = [N], and E to denote a set whose elements are unordered pairs of distinct elements of N. Each unordered pair e = (i , j) in

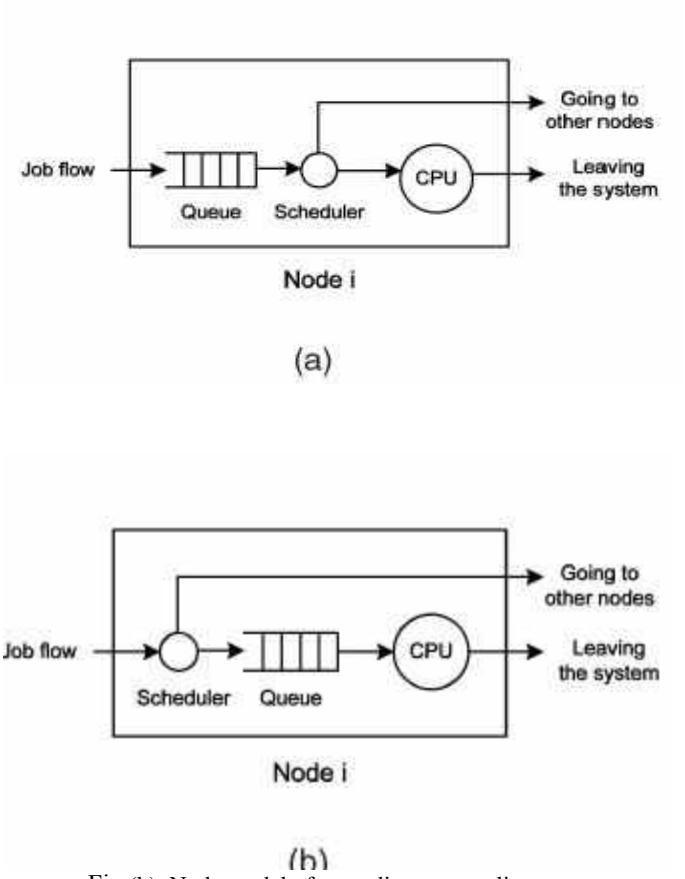

Fig (b). Node model of rate adjustment policy.

For load balancing algorithms, the model for a node is comprised of a scheduler, an infinite buffer to hold the jobs, and a processor. The scheduler is to schedule the jobs arriving at the node such that the mean response time of the jobs is a minimum. In the absence of a scheduler in a node, the job flow takes the following sequence of actions: A job enters the buffer, waits in the queue for processing, leaves the queue and gets processed in the processor, and then leaves the node (system). However, when a scheduler is present, depending on where a scheduler resides in a node to exercise its control on the job flow, we classify the distributed dynamic load balancing algorithms into three policies:

1. Queue Adjustment Policy (QAP): As shown in Fig. 2a, the scheduler is placed immediately after the queue. Algorithms of this policy [1], [5], [6] attempt to balance the jobs in the queues of the nodes. When a job arrives at node i, if the queue is empty, the job will be sent to the processor directly; otherwise, the job will have to wait in the queue. The scheduler of node i periodically detect the queue lengths of other nodes with which node i is concerned. When an imbalance exists, the scheduler will decide how many jobs in the queue should be transferred and where each of the jobs should be sent to. By queue adjustment, the algorithms could balance the load in the system.

2. Rate Adjustment Policy (RAP): As shown in Fig. 2b, the scheduler is immediately placed before the queue. When a job arrives at node i, the scheduler decides where the job should be sent and whether it is to be sent to the queue of node i or to other nodes under consideration. Once the job has entered the queue, it will be processed by the processor and will not be

transferred to other nodes. Using this policy, the static algorithms [9], [11] can attempt to control the job processing rate on each node in the system and eventually obtain an optimal (or near optimal) solution for load balancing. Because of the high computation overheads, until now no dynamic algorithm in the literature used this policy. In this paper, we will propose a dynamic algorithm which belongs to this policy.

3. Combination of Queue and Rate Adjustment Policy (QRAP): As shown in Fig. 2c, the scheduler is allowed to adjust the incoming job rate and also allowed to adjust the queue size of node i in some situations. Because we consider a dynamic situation, especially when we use RAP, in some cases, the queue size may exceed a predefined threshold and load.

V.STUDY OF ALGORITHMS:

In this section, we will introduce the algorithm named ELISA [1], which will be used as a benchmark algorithm and qualifies under the QAP category, and we will propose two algorithms based on LBVR [17], referred to as Rate-based Load Balancing via Virtual Routing (RLBVR), based on RAP, and Queue-based Load Balancing via Virtual Routing (QLBVR), based on QRAP, respectively.

1. ELISA: Estimated Load Information Scheduling Algorithm: We describe ELISA [1] in brief. In ELISA, the load scheduling decision is taken as follows: From the estimated queue lengths of the nodes in its neighboring nodes and the accurate knowledge of its own queue length, each node computes the average load on itself and its neighboring nodes. Nodes in the neighboring set whose estimated queue length is less than the estimated average queue length by more than a threshold θ form an active set The node under consideration transfers jobs to the nodes in the active set until its queue length is not greater than θ. and more than the estimated average queue length. The value of θ, which is predefined, is a sensitive parameter and it is of importance to the performance of ELISA. Here, the threshold θ is fixed in such a way that the average response time of the system is a minimum. Balance may result. Once this happens, QAP starts3.2 The Proposed Algorithm: RLBVR

2. The Algorithm: RLBVR

Although LBVR is a static load balancing algorithm, due to its super-linear convergence rate [17], LBVR can be tuned to handle dynamic situations. Thus, we attempt to design a dynamic load balancing algorithm RLBVR based on the working style of LBVR. The main structure of this algorithm is: First, we add a virtual node, which is referred to as the destination node (node d), into the network system. Connect node d with each node i (i N) by a virtual direct link (i ; d). Let the nodal delay of node i be treated as the communication delay on link (i ; d). After these modifications, the process of load balancing can be described in an alternative way. And more than the estimated average queue length. The value of θ, which is predefined, is a sensitive parameter and it is of importance to the performance of ELISA. Here, the threshold θ is fixed in such a way that the average response time of the system.

Three-node system has been transformed into a datagram network in which node i basically acts as a router. Referring to this figure, we observe that, for each node i, i = 1, 2 , 3, it can consider two paths to reach node d via its neighboring nodes and one path to reach node d directly. For example, from node 1 to node d, the paths. the way in which the loads are shared by the nodes can be described as follows:

3. PROCEDUR FOR NODE

Decision to use virtual servers as a fundamental unit of load balancing, and describe our earlier load balancing schemes on which the algorithm of this paper is based.

However, since items are queried by their IDs, changing the ID of an object would make it difficult to locate that object subsequently. Furthermore, some applications compute the ID of an object by hashing its content [7], thus rendering its ID static.

Just as the underlying DHT would do. In the case of Chord [7], each virtual server v of a node that leaves the system would be taken over by a node that is responsible for a virtual server v_ which immediately succeeds v in the identifier space. Similarly, when a node joins, it picks m random points in the ID space and splits the virtual servers there, thereby acquiring m virtual servers.

We assume that there are external methods to make sure that In particular, we assume that there is replication of data objects as proposed
In CFS [7], and departure of a node would result in the load being transferred to the neighbors in the identifier space.

In a previous paper, we introduced three simple load balancing schemes that use the concept of virtual servers for static systems [5]. Since the algorithm presented in this paper is a natural extension of those schemes, we briefly review them here. The schemes differ primarily in the number and type of nodes involved in the decision process of load balancing.

In the simplest scheme, called *one-to-one*, each lightly loaded node v periodically contacts a random node w. If w is heavily loaded, virtual servers are transferred from w to v such that w becomes light without making v heavy.

The second scheme, called *one-to-many*, allows a heavy node to consider more than one light node at a time. A heavy node h examines the loads of a set of light nodes by contacting a random *directory node* to which a random set of light nodes have sent their load information. Some of h's virtual servers are then moved to one or more of the lighter nodes registered in the directory.

Finally, in the *many-to-many* scheme each directory maintaints load information for a set of both light and heavy nodes. An algorithm run by each directory decides the reassignment of virtual servers from heavy nodes registered in that directory to light nodes registered in that directory. This knowledge of nodes' loads, which is more centralized than in the first two schemes, can be expected to provide a better load balance. Indeed, our results showed that the many-to-many technique performs the best. Our new algorithm presented in the next section combines elements of the many-to-many scheme (for periodic load bal-acing of all nodes) and of the one-to-many scheme (for emergency load balancing of one particularly overloaded node).

The basic idea of our load balancing algorithm is to store load information of the peer nodes in a number of *directories* which periodically schedule reassignments of virtual servers to

achieve better balance. Thus we essentially reduce the distributed load balancing problem to a centralized problem at each directory.

Each directory has an ID known to all nodes and is stored at the node responsible for that ID. Each node n initially chooses a random directory and uses the DHT lookup protocol to report to the directory (1) the loads $\_v_1, \ldots, \_v_m$ of the virtual servers for which n is responsible and (2) n's capacity c_n. Each directory collects load and capacity information from nodes which contact it. Every T seconds, it computes a schedule of virtual server transfers among those nodes with the goal of reducing their maximum utilization to a parameterized *periodic threshold* k_p. After completing a set of transfers scheduled by a directory, a node chooses a new random directory and the process repeats.

Than the estimated average queue length. The value of θ, which is predefined, is a sensitive parameter and it is of importance to the performance of ELISA. Here, the threshold θ is fixed in such a way that the average response time of the system is a minimum.

VI.EXPERIMENTAL RESULT AND DISCUSSION:

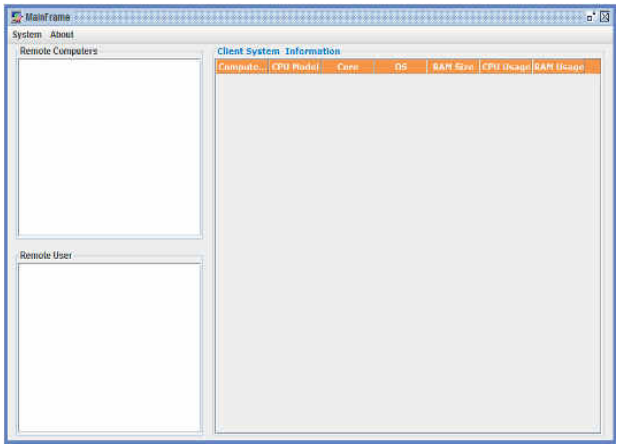

Figure 6.1: Main Window

The output screen shows a scenario where there is no client is connected to the server. The table on the screen shows the system details of the client connected to it.

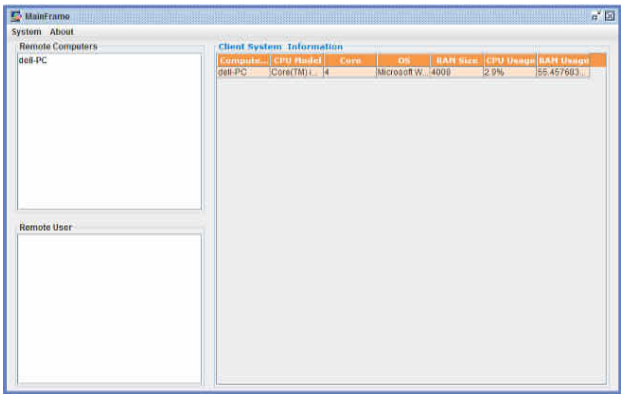

Figure 6.2: Main Window

The output screen shows a scenario where there are one client connected to the server. The table on the screen shows the system details of the clients connected to it.

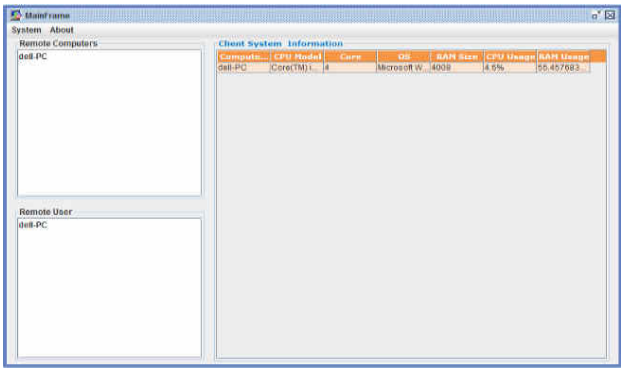

Figure 6.3: Main Window

The output screen shows a scenario where there is one client and on remote client is connected to the server. The table on the screen shows the system details of the clients connected to it.

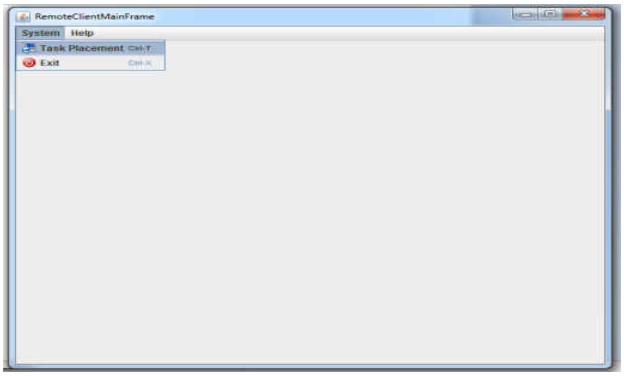

Figure 6.4: Remote Client Task Placement Frame

This frame window allows the user to choose a task or application that he/she desires to use for exploiting resources.

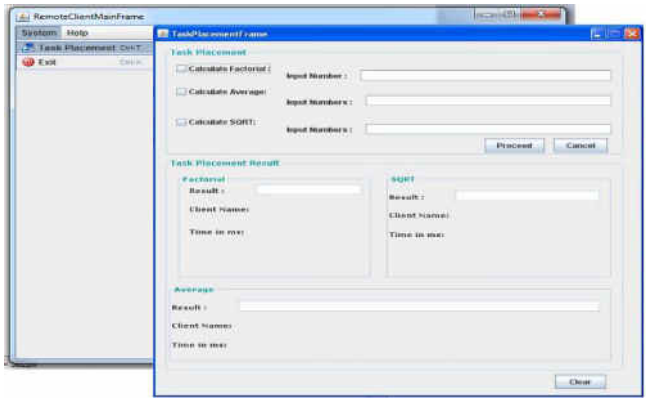

Figure 6.5: Task Placement Frame

The figure shows that the task is placed to the client which was least loaded and gives client-name and total time required for the execution along with the result

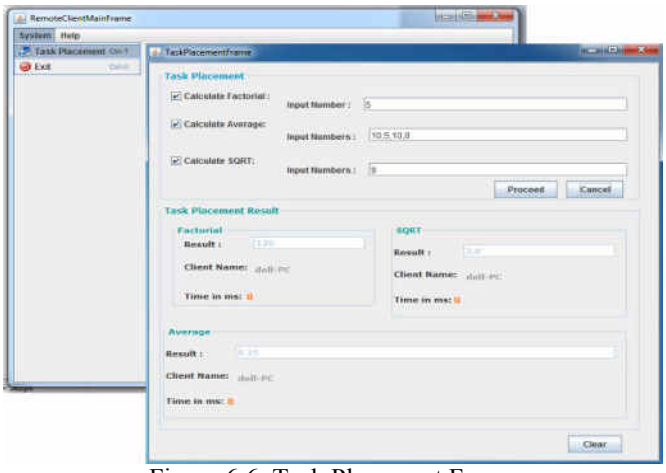

Figure 6.6: Task Placement Frame

The figure shows that the task is placed to the client who was least loaded and gives client-name and total time required fo the execution along with the result.

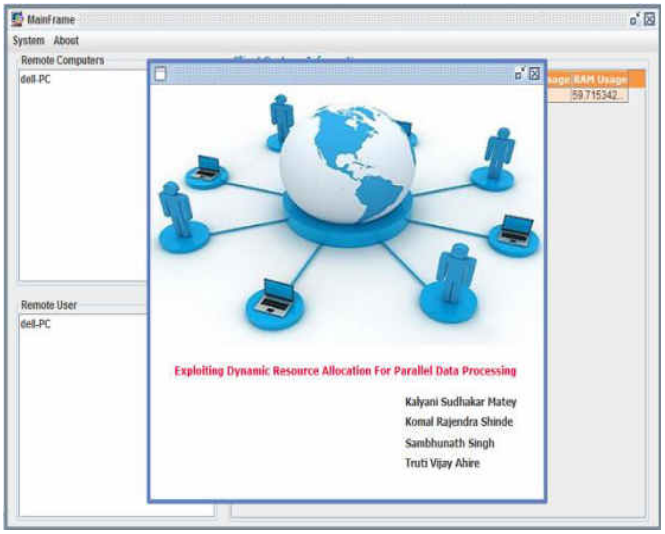

Figure 6.7: The About Window

The figure shows that the task is placed to the client which was least loaded and gives client-name and total time required fo the execution along with the result.

VII.Conclusions and Future Work

In this paper, we first classified the distributed dynamic load balancing algorithms into three policies:

QAP policy: queue adjustment policy, RAP policy: rate adjustment policy, and QRAP policy: combination of queue and rate adjustment policy. We proposed two algorithms, referred to as Rate-based Load Balancing via Virtual Routing (RLBVR) and Queue-based Load Balancing via Virtual Routing (QLBVR), which belong to RAP and QRAP, respectively. These two algorithms are based on LBVR and, hence, the computation overheads are small [17]. We have used Estimated Load Information Scheduling Algorithm (ELISA) [1] to present QAP policy, the main idea of which is to carry out estimation of load by reducing the frequency of status exchange, thereby reducing the communication overheads. Our policies are directly useful for performance evaluation of cluster/grid and distributed networks. The usefulness and applicability of our policies are demonstrated via rigorous simulation tests on a wide variety of system loading and other influencing parameters. We have also demonstrated the applicability of our policies to large scale.
We construct a dynamic job arrival rate pattern and carry out rigorous simulation experiments to compare the performances of the three algorithms under different system loads, with different status exchange intervals. With our rigorous experiments, we have shown that, when the system loads are light or moderate, algorithms of the RAP policy are preferable. Different loading situations. Our system model and experimental study can be directly extended to large size networks, such as multidimensional hyper cubes networks, to test their performances. Finally, in this paper, we have rigorously demonstrated the performances of the algorithms for a single class of jobs. In our near future work, we intend to divide the jobs in the system into several classes and assign each class of jobs its own priority. It would be interesting to consider multiclass jobs system as well and analyze the performances of these algorithms. From our experiments, we have clearly identified the relative metrics of the performances of the proposed algorithms and we are able to recommend the use of suitable algorithms for
Different loading situations. Our system model and experimental study can be directly extended to large size networks, such as multidimensional hyper cubes networks.

To test their performances a dynamic job arrival rate pattern and carry out rigorous simulation experiments to compare the performances of the three algorithms under different system loads, with different status exchange intervals. With our rigorous experiments, we have shown that, when the system loads are light or moderate, algorithms of the RAP policy are preferable the main idea of which is to carry out estimation of load by reducing the frequency of status exchange.

VII. REFERENCES:

1. L. Anand, D. Ghose, and V. Mani, "ELISA: An Estimated Load Information Scheduling Algorithm for Distributed Computing System," Computers and Math. with Applications, vol. 37, pp. 57-85, 1999.
2. D. Evans and W. Butt, "Dynamic Load Balancing Using Task- Transfer Probabilities," Parallel Computing, vol. 19, pp. 279-301,1993.
3. C. Walshaw and M. Berzins, "Dynamic Load-Blancing for PDE Solvers on Adaptive Unstructured Meshes," Concurrency: Practice and Experience, vol. 7, pp. 17-28, 1995.
4. Y. Zhang, K. Hakozaki, H. Kameda, and K. Shimizu, "A
5. Performance Comparison of Adaptive and Static Load
6. Balancing in Heterogeneous Distributed Systems," Proc.
7. IEEE 28th Ann. Simulation Symp., pp. 332-340, Apr. 1995.
8. Y. Amir, B. Awerbuch, A. Barak, R.S. Borgstrom, and A. Keren, "An Opportunity Cost Approach for Job Assignment in a Scalable Comsputing Cluster," IEEE Trans. Parallel and Distributed Systems, vol. 11, no. 7, pp. 760-768, July 2000.
9. 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, Mar. 1998.
10. L. Fratta, M. Gerla, and L. Kleinrock, "The Flow Deviation Network Design," Networks, vol. 3, pp. 97-133, 1973.
11. 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.
12. 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.
13. M. Mitzenmacher, "How Useful Is Old Information?" IEEE Trans.Parallel and Distributed Systems, vol. 11, no. 1, pp. 6-20, Jan. 2000.
14. J. Li and H. Kameda, "Load Balancing Problems for MulticlassJobs in Distributed/Parallel Computer Systems," IEEE Trans.Computers, vol. 47, no. 3, pp. 322-332, Mar. 1998.
15. 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.
16. J. Li and H. Kameda, "Optimal Static Load Balancing of Multi- Class Jobs in a Distributed Computer System," Proc. 10th Int'l Conf. Distributed Computing Systems, pp. 562-569, 1990.
17. A.E. Kostin, I. Aybay, and G. Oz, "A Randomized Contention-Based Load-Balancing Protocol for a Distributed Multi server Queuing System," IEEE Trans. Parallel and Distributed Systems, vol. 11, no. 12, Dec. 2000.
18. F.E. Beichelt and L.P. Fatti, Stochastic Processes and Their Application. Taylor & Francis, 2002.
19. D. Bertsekas and R. Gallager, Data Networks. Prentice-Hall, 1992.
20. Z. Zeng and V. Bharadwaj, "A Static Load Balancing Algorithm via Virtual Routing," Proc. Conf. Parallel and Distributed Computing and Systems (PDCS '03), Nov. 2003.
21. M. Avriel, Nonlinear Programming Analysis and Methods. Prentice- Hall, 1997.
22. N.U. Prabhu, Foundations of Queuing Theory. Kluwer Academic, 1997.
23. Y. Zhang, H. Kameda, and K. Shimizu, "Adaptive Bidding Load Balancing Algorithms in Heterogeneous Distributed Systems," Proc. IEEE Second Int'l Workshop Modeling, Analysis, and Simulation of Computer and Telecomm. Systems, pp. 250-254, Jan. 1994.