Dynamic Load Balancing Algorithms for Distributed Networks

M.Thejovathi M.Tech(CS&E),
Jawaharlal Nehru Technological University , Hyderabad, India

Summary
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 behaviours 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 behaviours 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.

Index Terms
Dynamic load balancing, cluster or distributed computer system, mean response time, queuing theory, status exchange interval.

1. Introduction:

A distributed computer system consists of many heterogeneous processors with different processing capabilities, connected by two-way communication links, and having their own resources/buffers. In such a system, if some hosts remain idle while others are extremely busy, system performance will be affected drastically. To prevent this, load balancing is often used to distribute the jobs and improve performance measures such as the mean response time (MRT), the time difference between the time instant at which a job arrives to the system and the time instant at which the

job gets processed, system utilization, etc. The design of such load balancing algorithms, in general, considers several influencing factors, for instance, the underlying network topology, communication network bandwidth, job arrival rates at each processor in the system. Load balancing algorithms can be classified as either dynamic or 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. However, in [17], the authors

Manuscript received February 5, 2014
Manuscript revised February 20, 2014

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 algorithms into three policies: the Queue Adjustment Policy (QAP), the Rate Adjustment Policy (RAP), and the Combination of Queue and Rate Adjustment Policy (QRAP). The details of the classification will be shown in Section 2. Based on LBVR, 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). We introduce an algorithm called Estimated Load Information Scheduling Algorithm (ELISA) and an algorithm named Perfect Information Algorithm (PIA), reported in the literature [1], for the purpose of continuity. The algorithms ELISA and PIA belong to QAP, whereas RLBVR and QLBVR belong to RAP and QRAP, respectively. We carry out a large number of rigorous simulation experiments to capture and analyze the effect of time-varying loads and different lengths of time intervals on the algorithms. As our focus is to analyze and understand the behaviours of the algorithms in terms of their load balancing ability, minimization of mean response time, in our rigorous simulation experiments, we consider a single class of jobs for processing One of our added considerations in this study is to gain intuition regarding the relative metrics of the different approaches under consideration. We extend our simulations to a large scale cluster system such as Mesh architecture that is of practical use in real-life applications. Based on the mesh topology, many prototype and commercial systems have been built. Our contribution elicits certain important behaviours of the distributed dynamic load balancing algorithms that serve to quantify the performances under different situations. From the simulations, we observe that, when system utilization is light or medium, RAP performs much better than QAP and QRAP with a relatively longer status exchange interval, which means less communication overhead. When system utilization is very high (Þ > 0.9), QAP performs the best among the three load balancing policies with high communication overheads. When the system utilization changes rapidly, QRAP is suitable and can achieve good performance with moderate communication overhead. This paper is organized as follows: Section 2 describes in detail the system model and the classification of the distributed dynamic load balancing algorithms. In Section 3, we propose RLBVR and QLBVR and give a brief introduction of ELISA. In Section 4, we present the results of simulations carried out in a 2-processor system to illustrate certain salient features of the algorithms under consideration. In Section 5, we extend our work to a large scale multiprocessor system, give a brief description of PIA, and compare the algorithms of the three policies. We highlight our contributions and discuss possible future extensions in Section 6

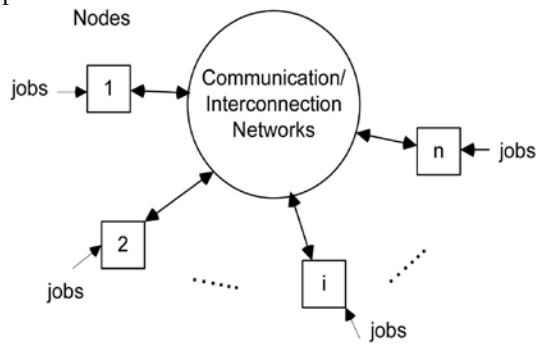


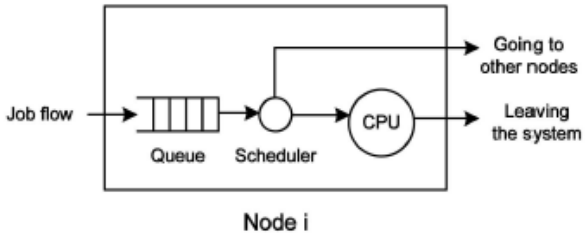
Fig. 1. A distributed/parallel computer system

2. Related work:

The System Model and Classification Of Dynamic Load Balancing Algorithms:

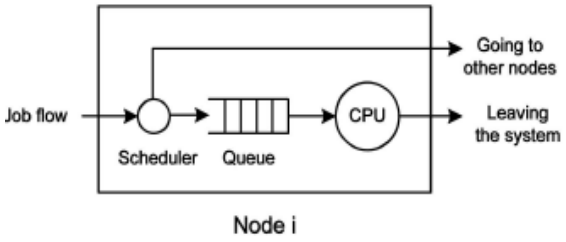
We first present a general system model in the design of the algorithms. For convenience, we use "node" and "processor" interchangeably in the rest of this paper. We consider a generic parallel/distributed system shown in Fig. 1. The system consists of n heterogeneous nodes, which represent host computers 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 neighbouring node of j if i is directly connected to j by an edge. For a node j, let $V_j = \{i, |(i,j) \in E\}$ denote a set of neighbouring nodes of node j. We assume that jobs arrive at node i (i $\in$ N) according to an ergodic process, such as inhomogeneous Poisson process with intensity function $\lambda_i(t)$ [15]. A job arriving at node i may either be processed locally or transferred through the network to another node j (j $\in$ 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 ($j \in$ 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 $\in$ 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].



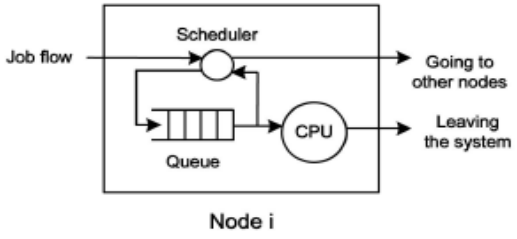
(a)

Fig.2 . (a) Node model of queue adjustment policy.



(b)

(b) Node model of rate adjustment policy.



(c)

(c) Node model of combination of queue and 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 detects 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 imbalance may result. Once this happens, QAP starts to work and guarantees that the jobs in the queues are balanced in the entire system. In this policy, we can consider the rate adjustment as a "coarse" adjustment and the queue adjustment as a "fine" adjustment. To the best of our surveys to date, there is no algorithm in the published literature that falls into this class of policies. In this paper, we will propose an algorithm which belongs to this policy that considers realizing a dynamic load balancing in distributed networks.

3. 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.

3.1. 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 neighbouring nodes and the accurate knowledge of its own queue length, each node computes the average load on itself and its neighbouring nodes. Nodes in the neighbouring set whose estimated queue length is less than the estimated average queue length by more than a threshold θ form an active set

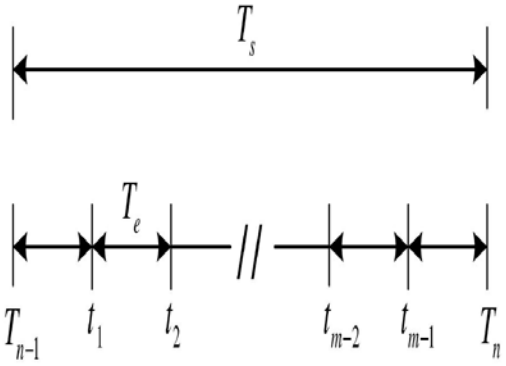


Fig. 3. Intervals of estimation and status exchange

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.

3.2 The Proposed 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 ($i \in N$) 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. As shown in Fig. 4, a 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 neighbouring nodes and one path to reach node d directly. For example, from node 1 to node d, the paths are: 1 → 2 → d, 1 → 3 → d, and 1 → d. The way in which the loads are shared by the nodes can be described as follows:

The jobs that arrive at node i according to a Poisson process with an average external job arrival rate of λ_i are routed to destination node d via every node $j \neq i$. The goal of load balancing now can be alternatively stated as a problem which attempts to minimize the mean link delay for each job in the system. Then, each node i in the system only considers the routing paths of (i , j)and (i , j) → (j , d) , $j \in$ Vi Use Newton's method [18] to obtain the optimal job routing rate of each path of node i. Eventually, obtain an optimal job transferring rate x ij and job processing rate βi, which are equal to job flow rates on paths (i , j)→ (j , d)and path (i , d), respectively. Each node i in the system monitors its external job arrival rate and obtains an estimate of the job arrival rate λ_i^n , which will be used as the job arrival rate of node i during the time interval between Tn_1 and Tn for computing. Before node i starts computing the optimal processing rate and transfer rates, node i broadcasts a request message to its neighbouring nodes.

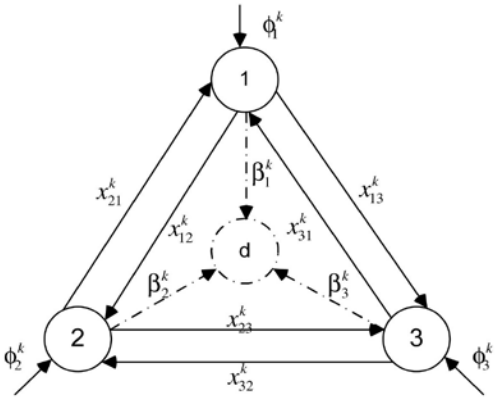


Fig. 4. Routing paths for each node

When node j (j ∈ Vi) receives this message, it will send x_{ji} n and βn j to node i immediately. If node i does not receive the response message from node j, it will consider node j has been shut down or the edge (i , j) is broken down and it will not send jobs to node j until it receives some message from node j some time later. Once node i obtains all local information, it can start computing to obtain an optimal solution, following which, node i sends some jobs to its neighbouring nodes and processes some jobs locally during this time interval. In the following, we will give more details on how to compute the optimal solution in each node.

3.2.1 Procedure for a Node:

For node i in the system, we denote Ui (βi)as the mean nodal delay function for a job and Gij (xij) as the mean communication delay function for a job transferred from node i to j. Some generic models Gij (xij) of include the delay of sending a job from node i to node j and the delay of sending the response back from node j to node i. In general, the path taken in each of the above-mentioned transfers (job and response transfers) may be different. Note that the profile of functions Ui (βi) and Gij (xij) may be very complicated. In practice, for analytical ease, it is often assumed that the functions Ui (βi) and Gij (xij) are differentiable, increasing, and convex functions [12], [16]. Here, we introduce another function, Fij, to unite the two different delay functions of Ui and Gij as follows:

$$F_{ij}(x_{ij}) = \begin{cases} U_i(x_{ij}), (i,j) \in L \text{ and } j = d; \\ G_{ij}(x_{ij}), (i,j) \in L \text{ and } j \neq d. \end{cases}$$

Then, node i obtains x_{ji}^n and β_j^n for the time interval between Tn_1 and Tn. When node i starts to compute, it calculates the total job arrival rate γ_i^n, which is:

$$\gamma_i^n = \hat{\lambda}_i^n + \sum_{j \in V_i} x_{ji}^n.$$

Referring to Fig. 4 again, we can see that node i has Pi, a set of routing paths. Hence, node i must determine the job flow rate xnp on each path p, (p ∈ Pi), and the objective function of node i is

$$Minimize: D(x) = \frac{1}{\gamma_i^n} \sum_{p \in P_i} \left[x_p n \sum_{\substack{all\ links\ (i,j) \\ on\ path\ p}} F_{ij}(x_{ij}^n) \right], \quad (1)$$

$$Subject\ to: \quad \sum_{p \in P_i} x_p^n = \gamma_i^n,$$

$$x_p^n \geq 0,\ p \in P_i.$$

We denote d_p as the first derivative length of path p with respect to x^n_p

$$d_p = \sum_{\substack{all\ links\ (i,j) \\ on\ path\ p}} \frac{\partial [x_p^n F_{ij}(x_{ij}^n)]}{\partial x_p^n}.$$

Step 1: ***Initialization***

$r = 0$ (r: iteration index).

Determine γ_i^n.

Find a feasible solution x to satisfy $\sum_{p \in P_i} x_p^{n(r)} = \gamma_i^n$.

Step 2: ***Solution Procedure***

Let $r = r + 1$.

For $j = 1$ to $v + 1$, $v = |V_i|$
Calculate the first derivative lengths of the paths p, that is $d_p^{(r)}$, $p \in P_i^k$.
End For

Find the minimum first derivative length (MFDL) paths $\bar{p}_i^{(r)}$ among P_i.

For $j = 1$ to v, $v = |V_i|$
To path p_j, $p_j \in P_i$, $p_j \neq \bar{p}_i^{(r)}$, calculate the second derivative length: H_{p_j}.
Let $x_{p_j}^{n(r)} = max\{0,\ x_{p_j}^{n(r-1)} - \alpha^{(r-1)} H_{p_j}^{-1}(d_{p_j}^{(r-1)} - d_{\bar{p}_i^{(r-1)}}^{(r-1)})\}$.
Let $x_{ij}^{n(r)} = x_{p_j}^{n(r)}$, $link\ (i,j) \in p_j$.
End For

Let $x_{\bar{p}_i^{(r)}}^{n(r)} = \gamma_i^n - \sum_{p \in P_i,\ p \neq \bar{p}_i^{(r)}} x_p^{n(r)}$.

Step 3: ***Stopping Rule***

If $|D(x^{n(r)}) - D(x^{n(r-1)})|/D(x^{n(r)}) < \varepsilon$, then **stop**, where ε is a desired acceptable tolerance for solution quality; otherwise, go to Step 2.

3.3 The Proposed Algorithm: QLBVR :

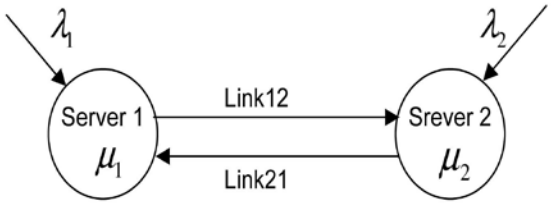
Now, we propose another dynamic load balancing algorithm, referred to as "Queue-based Load Balancing via Virtual Routing" (QLBVR), which belongs to the QRAP category, namely by combining queue and rate adjustment policies. As mentioned earlier, QLBVR carries out coarse adjustments on job transferring and processing rates and fine adjustments on queue lengths (number of jobs in the queue). These are as described below.

1. Coarse adjustment (on transfer and processing rates): The working style is similar to RLBVR. At every time instant Tn (n = 0 , 1, ...) , nodes in the system use the procedure for a single node to obtain an optimal solution. In the next time interval, Ts, each node adjusts its job transferring rates and job processing rate according to the computed optimal solution.

2. Fine adjustment (on queue lengths):The working style is similar to ELISA. Each status exchange interval Ts is divided into equal subintervals, denoted as estimation intervals Te. At time instant Te, node i estimates and adjusts the earlier estimate of the queue lengths of its neighbouring nodes and has an accurate knowledge of its own queue length.

From the analysis of QLBVR, we can observe that fine adjustment can affect the real transfer rates. However, xji, (j $\in$ Vi), the job transfer rate on link (j , i) that coarse adjustment of node i considers is an optimal solution of node j, instead of the real transfer rate on link(j , i). Hence, we can conclude that fine adjustment has no effect on the solution of coarse adjustment.

Remarks. Note that, in distributed dynamic load balancing algorithms, there may exist a nonzero probability that a job would shuttle between processors.



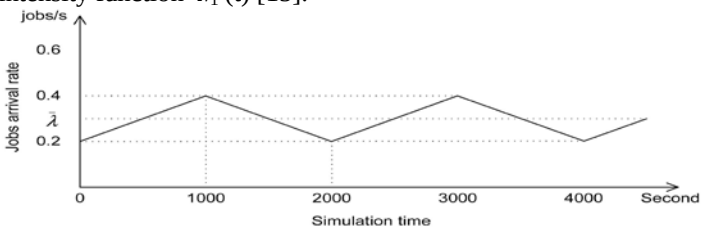
There are various ways to alleviate this problem. One of the ways is to allow the job to join at the position where it should have been if the job had arrived at this queue, instead of adding it at the end of the queue. This means that we keep track of the time at which it arrived at the system. This can considerably reduce the probability of the job being transferred once again and can guarantee minimizing the response time of that job.

4. Performance evaluation and discussions:

We have conducted extensive simulation tests to quantify the performance of the three types of load-balancing algorithms discussed above. As mentioned earlier, one of our added considerations in this study is to gain an intuition regarding the relative metrics of the different approaches under consideration. Our simulations focus on two main aspects of dynamic situation. One is the effect of the load on the nodes and the other is the effect of the status exchange interval lengths, which is indeed a crucial parameter that reflects the sensitivity of the algorithms. Here, we consider the mean response time (MRT) of a job as the main metric in our simulations. First, we give the introduction of our simulation model.

4.1 Two-Processor System Model and Some Important Issues:

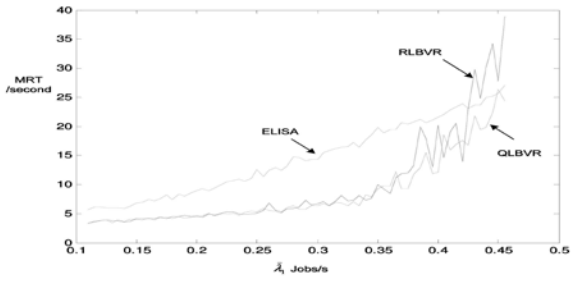
In order to illustrate the salient features of the algorithms in discussion, first we consider the simplest case when a multiprocessor system is limited to two processors, as shown in Fig. 5. It is possible that this model could also be considered equivalent to a situation in which an isolated processor communicates to a (single) processor that represents the rest of the network. Another reason why a 2-processor network is used as a test case is that an algorithm that does not work for two processors is unlikely to perform well for a multiprocessor case. In Fig. 5, we assume that jobs arrive at node 1 and node 2 according to Poisson processes with rates λ_1 and λ_2 respectively. We also assume that the average service times of a job at node 1 and node 2 are $1/\mu_1$ and $1/\mu_2$ respectively. From [19], we obtain that the average nodal delay of a job in node i (i = 1,2) is : $S_i = \frac{1}{\mu i - \lambda 1}$ where $\mu 1$ = 0:25 jobs/sec and $\mu 2$ = 0:5 jobs/sec in our simulation tests. In Fig. 5, jobs arrive at node 2 according to a homogeneous Poisson process with $\mu 2$ = 0:12 jobs/sec, which will be held constant throughout our simulations. Without load balancing, the system utilization of node 2 is P2= $\lambda_2/\mu 2$ = 0.12/0.5 = 0.24 and the system is at a low utilization level. In order to consider dynamic situations, the job arrival rate at a node must be a time-varying quantity. Thus, the job arrival rate at node 1 is an inhomogeneous Poisson process with intensity function λ_1 (t) [15].



We assume that the pattern of λ_1 (t) is as shown in Fig. 6. At time 0, λ_1 (t) starts from the lowest point l1. As time elapses, λ_1 (t) increases linearly. At time t = 1, 000 seconds, λ_1 (t) reaches the peak h_1 and, afterward, λ_1 (t) decreases linearly. Finally, at time t = 2, 000 seconds, it reaches the lowest point, l_1.After this point, λ_1 (t) increases again. For the sake of simplicity, we fix the difference between h_1 and l_1 to 0.2 jobs/sec. In order to describe the job arrival pattern quantitatively,
we denote $\lambda_1 = \frac{h1+l1}{2}$ as the load of the dynamic job arrival rate. For example, we use λ = 0.3 to consider the λ (t) shown in Fig. 6.

4.2 Effect of System Loading:

In this section, we will analyze the performances of the three policies with respect to the load on the system. First, in our simulation, we set λ_1 = 0.1 jobs/sec and then increase the value of λ_1 by 0.005 jobs per step. Also, we set Ts = 60 seconds and Te = 20 seconds in our experiments. For each value of λ_1, the simulation time is set to 20,000 seconds to obtain a statistical mean response time of the jobs arriving at the system. When the average system utilization p is greater than 0.95, we terminate the simulation, where p is defined as the ratio of average total arrival rate to aggregate processing rate of the system: $p = \frac{\lambda 1 n + \lambda 2 n}{\mu 1 + \mu 2}$



5 . Extension to large scale cluster System:

we shall consider a large scale networked cluster system to capture and understand the behaviour of our algorithms. This is an important study as it reflects a real life scenario. In our study, we specifically considered a mesh connected processor architecture. As mentioned earlier, based on the mesh topology, many prototype and commercial systems have been built and these architectures are able to handle data intensive computations. Hence, in this section, we extend our simulation experiments to a mesh multiprocessor system, as shown in Fig. 9. We will compare the performances of the three policies under different system workloads in a mesh-connected multiprocessor system.

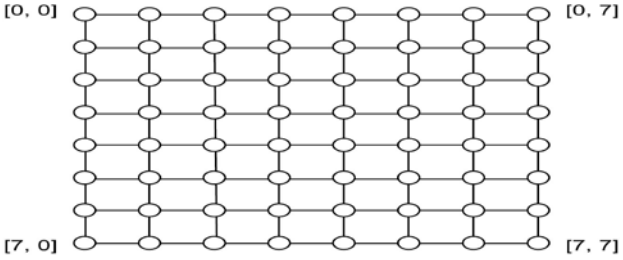


Fig. 9. The mesh-connected cluster system M=(8; 8) system

5.1 Static or Slowly Varying System Loading

Here, we assume that jobs arrive at node (x;,y) according to

a homogeneous Poisson process with ƛ(x , y) jobs/sec.

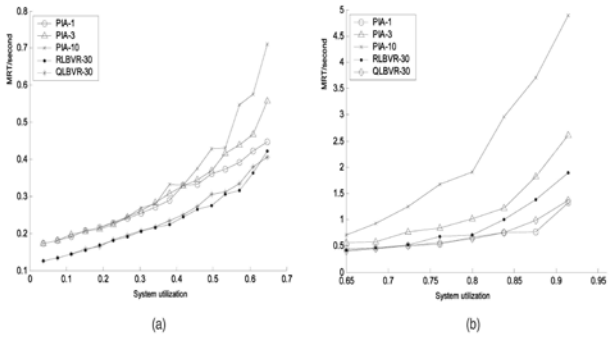
$$\rho = \frac{\sum_{i=0}^{7}\sum_{j=0}^{7}\lambda(x,y)}{\sum_{i=0}^{7}\sum_{j=0}^{7}\mu(x,y)}.$$

Under a given set of ƛ[X , Y], the system utilization is:

We carry out a series of simulations for the M [8, 8] mesh connected multiprocessor system with the five algorithms

described above, under different system utilization parameter

ƿ The simulation conditions are kept identical for all five algorithms for a given ƿ. For each simulation run, the Simulation time is set to 20,000 seconds, during which the first 5,000 seconds are considered “warm time.” After the warm time, we trace the jobs arriving at each node and record their arriving time, processing time, and leaving time. We average 10,000 jobs’ response time as the mean response time for this simulation. In the first simulation, the job arriving rates ƛ[X, Y]are randomly generated. Then, each node (x,y)increases its ƛ(x,y)by 0.25 jobs per second in each step. When the system utilization exceeds 0.9, we terminate the simulations. Below figure shows the simulation results for the five algorithms.

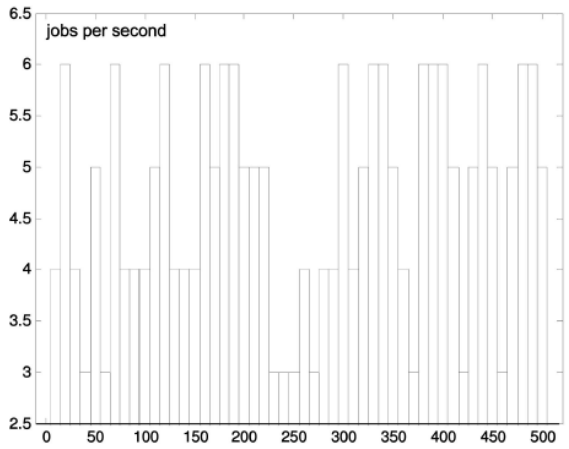


5.2 Experiments when the Arrival of Loads Is Varying Rapidly:

We construct another job pattern to simulate the situation when the job arriving rate of each node changes rapidly. We assume that, for each node, the time interval for a batch of jobs is 10 seconds, during which jobs arrive at the node according to a homogeneous Poisson process. In each 10 second time interval, the probability that the arrival rate will be same as in its previous interval is (0,1) otherwise, the job arriving rates are generated randomly with uniform distribution between ƛ min and ƛ max. We assume that, in each node, the maximum job arriving rate cannot exceed its maximum job processing rate.

For node (x, y),we use r_{min} to denote the ratio $\frac{\lambda min}{\mu(x,y)}$ and r_{max} to denote the ratio $\frac{\lambda max}{\mu(x,y)}$. Hence, we can use $[r_{min}, r_{max}]_{(x,y)}$ to denote this job pattern for node (x,y) in the system. For example, the job pattern shown in below Fig can be denoted as $(0.3,0.7)_{(7,7)}$for node (7, 7) whose job processing rate is nine jobs per sec. Again, let ƿ be the average system utilization for our simulations, which is:

$$\bar{\rho} = \frac{number\ of\ jobs\ arriving/simulation\ time}{\sum_{i=0}^{7}\sum_{j=0}^{7}\mu(x,y)}.$$



We conducted four kinds of simulation tests, indicated as

S.N. (simulation number), shown in the following:

A: System utilization is light. In this case, the job pattern of

each node (x,y) in the system is (0.1,0.4)(x,y)and

$$E(\bar{\rho}) = \frac{0.1+0.4}{2} = 0.25.$$

B: System utilization is moderate. In this case, the job pattern

is $(0.3,0.7)_{(x,y)}$for node (x,y)and

$$E(\bar{\rho}) = \frac{0.3+0.7}{2} = 0.5.$$

C: System utilization is high. In this case, the job pattern is

$(0.7,1)_{(x,y)}$for each node (x,y) in the system and

$$E(\bar{\rho}) = \frac{0.7+1}{2} = 0.85.$$

D: We randomly choose one-third of the nodes as lightly

loaded nodes ([0.1, 0.4]), one-third of the nodes as moderately loaded nodes ([0.3, 0.7]), and the remaining one-third of nodes as highly loaded nodes ([0.7, 1]) and

$$E(\bar{\rho}) = \frac{0.25+0.5+0.85}{3} = 0.533.$$

In each kind of simulations, for all five algorithms, we generate a set of job arriving rate sequences to ensure that the simulation conditions remain identical.

6. 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

cluster systems with mesh-connected topology.

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

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. 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

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 Performance Comparison of Adaptive and Static Load Balancing in Heterogeneous Distributed Systems," Proc. IEEE 28th Ann. Simulation Symp., pp. 332-340, Apr. 1995.
[5] 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.
[6] 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.
[7] L. Fratta, M. Gerla, and L. Kleinrock, "The Flow Deviation Network Design," Networks, vol. 3, pp. 97-133, 1973.
[8] 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.
[9] 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.
[10] M. Mitzenmacher, "How Useful Is Old Information?" IEEE Trans.Parallel and Distributed Systems, vol. 11, no. 1, pp. 6-20, Jan. 2000.
[11] 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.
[12] 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.
[13] 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.
[14] 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.
[15] F.E. Beichelt and L.P. Fatti, Stochastic Processes and Their Application. Taylor & Francis, 2002.
[16] D. Bertsekas and R. Gallager, Data Networks. Prentice-Hall, 1992.
[17] 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.
[18] M. Avriel, Nonlinear Programming Analysis and Methods. Prentice- Hall, 1997.
[19] N.U. Prabhu, Foundations of Queuing Theory. Kluwer Academic, 1997.
[20] 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.

Recieved B.Tech and M.Tech Degrees in Computer Science &Information Technology from Jawaharlal Nehru Technological University,Hyderabad ,AP, INDIA. 2004 & 2007. Currently working as an Assistant Professor of computer science Department in KING KHALID University K.S.A. Previously worked as Assistant Professor in JNTU Affliated Engineering Colleges, Hyderabad INDIA