Scalable Dynamic Load Balancing for P2P Irregular Network Topologies

Muhammad Waseem Akhtar and M-Tahar Kechadi
mwaseem@ieee.org, tahar.kechadi@ucd.ie

School of Computer Science and Informatics
University College Dublin, Belfield, Dublin 4, Ireland

Abstract. In this study we present a two-phase dynamic load balancing technique for P2P systems. In the first phase, given P2P network is mapped onto a hierarchical topology based on the tessellation of a 1-D space. This hierarchy is called TreeP (Tree based P2P architecture). In the second phase load balancing among the nodes is performed using PSLB algorithm. We also present an optimized version of our load balancing technique. This optimization makes the technique highly parallel and scalable. This technique is simple, efficient and does not introduce a considerable overhead as shown in the experimental results.

1 Introduction

Peer to peer (P2P) computing describes the current trend towards utilizing diverse resources available within a widely distributed network of nodes. A peer to peer system is formed by a large number of nodes that can join and leave the system anytime and have equal capabilities without any central control. Several P2P architectures have been developed, which include Chord, CAN, Pastry, Tapestry and P-Grid [5, 6, 21]. Although P2P systems have become an architecture of choice for file sharing applications, such systems are equally suitable for scientific computing, e-commerce and Grid applications [7, 22]. In fact peer to peer systems and Grids share the same focus on harnessing resources across multiple administrative domains. One of the most crucial aspects of these systems is the efficient utilization of resources and the distribution of the workload among the nodes [8–11, 20]. Thus load balancing is an important system function designed to distribute workload among available processors to improve the throughput and execution time of the distributed algorithms.

In this paper we propose a new dynamic load balancing technique based on parallel prefix, also known as scan operation. The proposed technique has two phases. During the first phase the network is mapped to a TreeP. The second phase deals with the redistribution of the workload among the nodes based on their processing power and their current load. The proposed technique is dynamic, non-preemptive, adaptive and fully distributed.

The paper is organized as follows: In the next two sections the mathematical models of network, nodes and tasks are presented along with the detailed description of the TreeP structure. In section three, we present our dynamic load

© A. Gelbukh, S. Torres, I. López (Eds.)
Advances in Computing Science and Engineering
Research in Computing Science 19, 2006, pp. 107-120

balancing technique. We illustrated our technique by an example. Creation of TreeP hierarchy is presented in section four. In section five an optimized version of our load balancing technique is presented. The section six discusses the performance of the technique. In section seven the experimental results are given. Finally concluding remarks and the future work are given in the section eight.

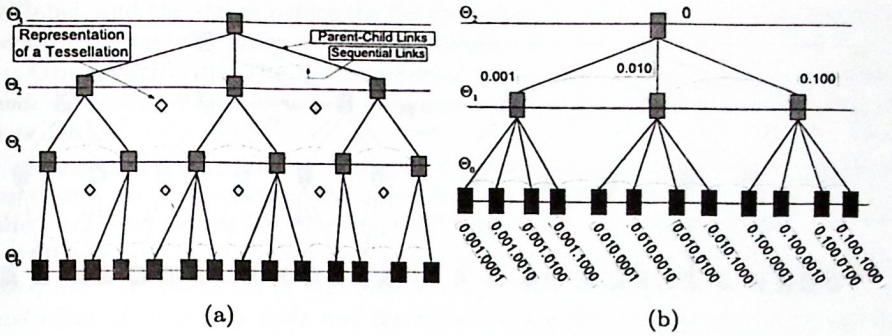
2 System Model and TreeP Architecture

A P2P system can be represented as a graph $G_{nm}(V, E)$ of n nodes and m edges. The set V represents the nodes of the system and E describes the interconnection links between the nodes. We consider that the following hold in system under consideration:

- Each node v_i is autonomous and is characterized by three attributes: its processing speed π_i representing the number of work units that can be executed per unit of time, its load n_i, and its neighborhood e_i.
- The network's flow b_{ij}, which is the effective data rate in bits per second on the link that connect the nodes v_i to vj.
- Each node v_i is equipped with a communication coprocessor that allows the communication and computation of loads to be carried out simultaneously.
- The tasks are independent and can be executed on any node regardless of their initial placement.
- Each task t_i is characterized by two parameters, which are the number of work units (in terms of computations) within the task (β_i), which dictates its computation cost(C_{comp}), and the number of packets required to transfer the task (μ_i), which dictates its communication cost (C_{comm}).

The TreeP topology [4], as shown in figure $1(a)$ consists of several layers of peers. The topology consists of connections that are actively maintained. These connections provide the skeleton of the hierarchy. New joining peers are assigned to the lowest layer, and are promoted to upper layers to fit the needs of the system. The system promotes the nodes in a distributed manner and the criteria used for promotion are based on the characteristics of the nodes such as: CPU, memory, bandwidth, network load, uptime and storage space. The network is assumed to be spatially distributed. The space is divided into tessellations and each tessellation is associated with one level of hierarchy. The ID of a node V_{ij} provides a spatial location in its tessellation. Level Θ_0 contains all active nodes of the network. The level Θ_j, where $j > 0$, consists of the nodes that have been selected from the level Θ_0, based on their uptime and their resource characteristics. These selected nodes assume the role of higher level nodes, continuing the existing role of level 0. Each node at level k is a parent of nodes covered by its tessellation at level $k - 1$. A parent is also responsible for promoting a child to its level of the hierarchy.

The TreeP structure is similar to a B^+Tree [14]. However unlike B^+Tree, the nodes of level Θ_0 can also be part of any other level. The higher level nodes act as a fabric of virtual interconnection network for TreeP topology, and are



(a) (b)

Fig. 1. (a) The TreeP Structure (b) Labeling Scheme for TreeP

called *virtual nodes*. Similarly to fully map the given system to a TreeP topology *virtual links* are introduced. These *virtual links* are considered as active links with zero bandwidth. Another main difference between TreeP and B^+Tree is that the nodes within the same level are connected by a bus topology. The height h of the hierarchy can be calculated in the same way as for a B^+Tree. The height of a TreeP network having n nodes and a minimum degree $t, t \geq 2$, is given by: $h \leq \log_t((n+1)/2)$. Usually, the height of a tree corresponds to the average number of children per parent c: $h = \log_c((n+1)/2)$.

3 Extension of Positional Scan Load Balancing to TreeP

The load balancing technique presented in this paper constitutes an extension of Positional Scan Load Balancing (PSLB) algorithm [1, 2]. The application of PSLB leads underlying system to a perfect load-balanced state at a very reasonable time. The PSLB algorithm preserves the locality decomposition. It can be applied at fine grain level to load balance a parallel application as well as at coarse grain level to schedule heterogeneous tasks.

The Extended PSLB presented in this paper is a two-step strategy. Firstly, the given P2P system is structured as a TreeP, and then PSLB is deployed. The PSLB technique is based on parallel prefix operator, or scan [10, 15–17], which can be defined as follows:

Definition: The prefix-sums (scan) operation takes a binary operator $\oplus$, and an ordered set of n elements $[a_0, a_1, \ldots, a_{n-1}]$, and returns the ordered set $[a_0, (a_0 \oplus a_1), \ldots, (a_0 \oplus a_1 \oplus \ldots \oplus a_{n-1})]$.

The overhead of executing PSLB on a system is directly proportional to the cost of performing scan operation. Scan operation can be implemented on TreeP in an efficient manner by exploiting the low height of the topology.

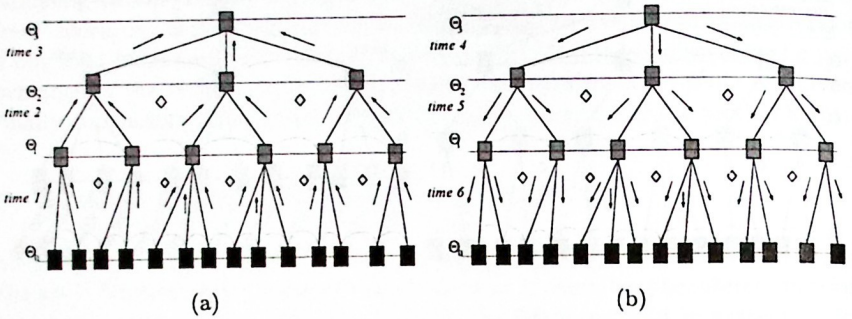


Fig. 2. The Scan Operation on TreeP Structure (a) Step I: Up-sweep phase (b) Step II: Down-sweep phase.

Proposition 1. *The time required for a scan operation performed on an n-node TreeP of height h is of complexity* $O \log n$, *that is* $O(h)$.

Proof. A scan can be performed on a TreeP in two phases, an up-sweep phase and a down-sweep phase. During the up-sweep phase, nodes at the level Θ_0 send the operand variable to their respective parent nodes. An intermediate node, will create a partial sum of the received values, and will pass it on to its parent node, and eventually this partial sums vector arrive at the root, Fig 2(a). At the root second phase starts, where this partial sums vector is made a full scan vector and is sent down to the lower levels. At the end of down-sweep phase, each node in the system has a complete scan vector, Fig 2(b). So assuming each communication channel an independent one, the complete scan operation can be performed in $2 \times h$, where h is the height of the TreeP.

3.1 TreeP Labeling

A simple yet efficient labeling scheme for TreeP nodes is presented in this section. This labeling scheme is based on prefix binary concatenated strings and holds the following properties: 1) a node v can determine its level in the tree, 2) two nodes v and u can determine their nearest common ancestor and, 3) a node v can determine its neighbors at the same level. The first two properties of the proposed labeling scheme are important for a scan operation and the third property is vital for the optimal routing during the task migration phase.

The proposed labeling scheme, shown in figure 1(b), assigns a label of 0 to the root. The label of a non root node is its parent's label (prefix) concatenates the delimiter and its own label calculated by its parent. Each parent node calculates the labels for its children, such that, the label of n^{th} child is a binary string of length d, where d is the total number of children of the parent node, where n^{th} bit is inverted to 1. In the prefix labeling schemes [12, 13] the string before the last delimiter is called a prefixlabel, the string after the last delimiter is called a

selflabel, and the string before the fist delimiter, between two neighbor delimiter or after the last delimiter is called a component.

Observation 1: *The maximum cost for the label storage on an individual node of a given TreeP structure of maximum degree d and total number of nodes n is* $d \log n$.

Observation 2: *Node v and node u having the same parent node are neighbors, and are connected at the same level of TreeP topology, if* x^{th} *bit is* 1 *in one's selflabel and* $(x+1)^{th}$ *or* $(x-1)^{th}$ *bit is* 1 *in other's selflabel.*

Observation 3: *Node v and node u having two different parents are neighbors, and are connected at the same level of TreeP topology, if one of them is the last child of its parent node and the second is the fist child of its parent and the parents are neighbors, that is, are connected at the same level of TreeP topology.*

Observation 4: *The nearest common ancestor of two nodes v and n is the node whose label is present in both node's label as a first common component.*

3.2 PSLB Algorithm

The PSLB algorithm can be summarized in the following five steps.

Algorithm 1: ***PSLB Algorithm - Brief Description***

1. Index the work units.
2. Use scan operator to collect information on the load in the system and on the processing power.
3. For each node (in parallel): Calculate normalized processing power vector
4. For each node (in parallel): Calculate locally the destination node of each work unit.
5. For each node (in parallel): Perform the migrations of the work units.

Consider a peer to peer system of total nodes q structured as a TreeP of height h. Each node v_i has a processing power π_i and a workload n_i expressed in terms of number of work units. The total work load and processing power of the system are denoted by $W = \sum_{i=1}^{q} n_i$ and $\Pi = \sum_{i=1}^{q} \pi_i$ respectively.

At the end of the two scan operations each node will know how much load and power is on its sub-hierarchy and in the system. Each node will then locally calculate the normalized relative processing power. In a perfect load balanced system, the load of each node is given by $W\gamma_i$, where γ_i is the normalized relative processing power.

The next step is to calculate the destination node for each work unit. Assume that a work unit u_x is currently on node n_i. Let node n_j be its destination node. Then, the problem consists of calculating the index of the node n_j. This is achieved by using the index number of work unit, the total load, and processing power on the left hand side of the node n_i. Each work unit is described in terms of two different index numbers. A local index number and a global index number. For example k^{th} work unit on node i can be described as I_k^i, that is work unit

number k on node i.

$$I_k^i = k + \sum_{a=1}^{a<i} n_a \quad (1)$$

The algorithm calculates the least index such that $\lambda_j \leq (k+S_i)/W$. This means that algorithm uses relative processing power of the nodes and the sum of the workload of the entire system to calculate the target node for each workload unit. The description of the algorithm to calculate the destination node is given in algorithm 2.

Algorithm 2: *Destination Node*

for all nodes i **in parallel**
  for $j = 1$ **to** n_i(load on node i)
    Find out last processor for which
    $\lambda_k \leq \frac{(j+S_i)}{W}$
    $l \leftarrow j + S_i + \lambda_k \times W$
    Migrate $(W_l^K \longleftarrow W_j^k)$
  end for
end for all

3.3 Example: PSLB on TreeP

Let us illustrate the PSLB technique by an example. Consider a TreeP of height 4 with 14 number of nodes, as presented in figure 1(a). The number of work units and the processing powers of each node are given in the first two rows of table 1.

Table 1. Initial load distribution, Processing Power, scan on Load and Processing Power, Normalized Processing Power and Final Load

v_i	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}	v_{13}
n_i	5000	3500	2000	2200	2500	1800	1400	1200	1500	3800	2800	1900	2400	2700
π_i	100	200	140	120	110	160	180	300	230	240	150	220	280	190
S_i	0	5000	8500	10500	12700	15200	17000	18400	19600	21100	24900	27700	29600	32000
λ_i	0	100	300	440	560	670	830	1010	1310	1540	1780	1930	2150	2430
γ_i	0.038	0.076	0.053	0.046	0.042	0.061	0.067	0.12	0.088	0.092	0.057	0.084	0.107	0.073
n_{bal}	1325	2649	1854	1589	1457	2119	2384	3974	3046	3178	1987	2914	3708	2516

Consider the node v_1. Its total initial load is 3500. The local index of its work units are labeled from 1 to 3500. The global index of its work units starts from 5000 + 1, since the total workload of v_0 is 5000. By executing the PSLB algorithm, we first calculate the exclusive scan for the processing power and the workload. Each node normalizes its processing power values.

For instance normalized processing power of node v_0 is 0.038168, so the workload that node v_0 should keep is 0.038168×34700, that is 1325. Quickly, each node determines which work unit should be kept and which work unit should be migrated and where. In addition, the under-loaded nodes know that they are receivers. In the table 1, node v_0 has to send 2649 tasks to node v_1 and 1026 tasks to node v_2. Node v_1 will send 828 work units to node v_2 and 1589 work units to node v_3 and 1083 work units to v_4 and eventually will have 2649 work units. That is the number of work units that node v_1 should have in a perfectly balanced system. Table 1 also shows the final distribution of the load among 14 nodes.

Table 2 shows the response time of the system before and after load balancing. Without load balancing, system will execute the tasks in 100 time units, but after the load balancing these tasks can be completed in 64 time units.

Table 2. Response Time of the system before and after load balancing

v_i	v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}	v_{13}
Initial Response Time	100	35	29	37	46	23	16	8	14	32	38	18	18	29
Final Response Time	64	44	41	45	50	38	35	31	30	33	41	32	27	28

4 TreeP Topology Creation

This constitutes the first step of EPSLB. The TreeP topology creation is presented in detail in [4], here we briefly describe it. The creation and maintenance of a TreeP structure is quite simple. When a node reaches a degree of 2 and does not have a parent, it will search for a parent by contacting its neighbors. The election of a parent is triggered when a node reaches a degree of 2. The election technique used in TreeP is described in [18, 19]. When the election is triggered, each participating node starts a countdown. The initial value of the countdown is calculated according to the node characteristics (CPU, bandwidth, average work-load, average network load, etc.). A node that has higher characteristics will have smaller countdown initial value. When the countdown of a node reaches 0 and if no other node was elected during this time, it will signal to its neighbors that it is their new parent. Similarly, if a parent has less than two children, it will start a countdown, but this time, the higher is the characteristic the longer is the countdown. At the end of the countdown, if it still has less than two children it will leave its current level and will become an ordinary node of the level 0.

Each peer maintains its routing table by exchanging data through its active connections. The exchange concerns only the routing table information that is out-of-date. When two nodes communicate for the first time they exchange information about their resources and state. Then, each node has to maintain this information with its direct neighbors. If the connection between two nodes

a and b is at level 0 and they have different maximum levels i_a and i_b, (with $i_a < i_b$), then a will send to b information about its parents at the level i_b.

Each active connection at level $i, (i > 0)$, allows the two end points to exchange their inner neighbors entry information and also information about their children. They also exchange their routing table entry about their immediate parent of level $i+1$. If the parent entry does not exist it will be added and then forwarded to its own parent. Such exchange prevents the network from having two roots of the tree that are not connected. Finally for any two neighbors, after the initial synchronization and the usual keep-alive message, they only exchange information concerning the out of dated data. Sometimes, the update can be delayed, waiting to be piggybacked during a keep-alive exchange. In the current implementation the update is exchanged immediately. This technique, for maintaining the routing tables, provides better connectivity and, therefore better performance and fault tolerance.

5 Optimized EPSLB on TreeP

An optimized version of EPSLB is presented in this section. This optimization not only exploits the tessellations and sequential links on the same level of a TreeP, but also makes the algorithm highly scalable.

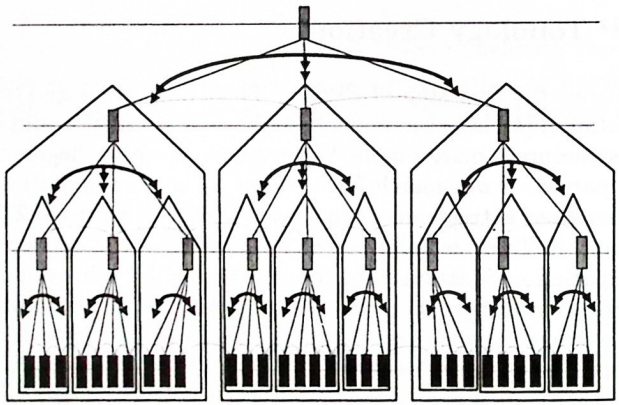


Fig. 3. A Tessellated TreeP Structure of height 4. Tessellations on the same level, contained in different parent tessellations, perform load balancing in parallel

A TreeP of height h can be modeled by a set of d_p tessellations of height $h-1$, where d_P is the degree of parent node shared by all the tessellations in the set.

$$TreeP^h = \{T_1^{h-1}, T_2^{h-1}, \ldots, T_{d_p}^{h-1}\}$$

Each tessellation can be described as T_y^x: where x represents the level of the tessellation and y represents the sequential order of the tessellation in its parent tessellation. The highest level node in a tessellation is called its root node. Root nodes of the tessellations on the same level can communicate through sequential links of TreeP topology. Parent node of the root node of a tessellation is called the parent node of the tessellation as well. Root nodes of the tessellations at level x with the common parent node are contained in the same $x+1$ level tessellation. Two tessellations on the same level with different parent nodes do not communicate directly. By using the TreeP labeling scheme presented in section 3.1, each node can easily determine all the above parameters and can spatially determine its location in a tessellated TreeP.

Similarly each tessellation at level $h-1$ in the above set can recursively be divided into d_p tessellations of height $h-2$:

$$T_1^{h-1} = \{T_1^{h-2}, T_2^{h-2}, \ldots, T_{d_p}^{h-2}\}$$

$$T_2^{h-1} = \{T_1^{h-2}, T_2^{h-2}, \ldots, T_{d_p}^{h-2}\}$$

At the same level each tessellation in the system can be considered as a single node participating in the PSLB algorithm, using the workload and processing power of the entire tessellation.Tessellations of height 1 are the leafs of the TreeP topology, that is, the nodes of the system.

The optimized EPSLB algorithm starts by an up-sweep phase of prefix operation, where each node will send its processing power and workload information to its parent node. Each parent/intermediate node will calculate the partial sum of these values and will pass on this information to its parent at the next level. As the down-sweep phase of the prefix operations starts, at each level, the root node of each tessellation T_y^x determines whether it is a sender or a receiver or both among the tessellations at the same level. A receiver (resp. sender) means that tessellation is underloaded (resp. over-loaded). If, for instance, a tessellation T_y^x has to receive additional tasks then its task consists of balancing its own workload according to the PSLB algorithm and just wait to receive more tasks which will go to the appropriate nodes. On the other hand, if it is over-loaded, then it uses PSLB to balance its own workload (only for the tasks that have to remain in the same tessellation), knowing that the tessellation of higher level will balance the tasks among themselves and therefore will migrate the extra tasks to the target tessellations. Each tessellation balances its load with other tessellations on the same level. This procedure guarantees that after its completion the entire system will be as close as possible to the perfect load balance state. The EPSLB on TreeP is highly parallel.

6 EPSLB Performance Model

In this section, firstly, a performance model for EPSLB is developed and then this model is extended to the optimized EPSLB. Let h and q denote the height and the number of nodes of a TreeP respectively. The number of communication

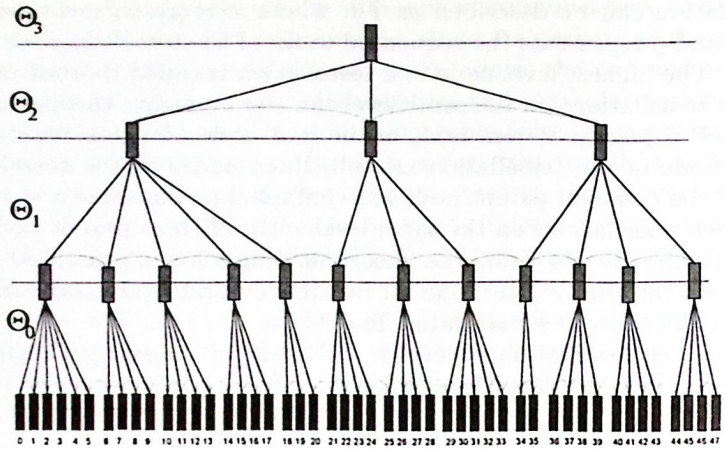


Fig. 4. A TreeP Structure of height 4 and 48 nodes

steps needed to perform a scan operation on the load and processing power ($(+, \gamma)$ and $(+, n)$) is $S_{comm} = 2(2 \times h)$ which is equal to $2 \times 2 \log q$. If we assume that both the load and processing power can be grouped in one message, then we only need one scan operation. The number of computations performed by each node is $S_{comp} = 2 \times (q - 1)$.

Let ψ and φ be the costs in time units of a communication and computation step, respectively, then the total cost of the algorithm can be expressed as follows:

$$S_q = S_{comc} + S_{comp} = 2(\log q)\psi + 2(q-1)\varphi \tag{2}$$

Equation 2 can be rewritten as:

$$S_q = S_{comc} + S_{comp} = 2(h)\psi + 2(q-1)\varphi \tag{3}$$

In optimized EPSLB load balancing is performed within each tessellation first and at every next level, each tessellation is considered as a single node participating in the load balancing. In optimized EPSLB algorithm, load balancing among highest level tessellations start only after $h + 1 = \log q + 1$ time steps. In a perfect load balanced system, the load of a tessellation t is $W^t = W\gamma^t$, where γ^t is the normalized relative processing power of tessellation t. So the root node of tessellation t will balance $W - W^t = W\gamma^t$ load with other tessellations on the same level within same parent tessellation. Since tessellations are created recursively, lower level tessellations perform the load balancing in parallel with in each parent tessellation making the optimized EPSLB algorithm highly scalable. Each sender node has to determine the target node on the workload units within the same parent tessellation.

Using the equation 3, cost of performing EPSLB on a tessellation t of height 1 with q^t nodes is:

$$S_q = S_{comc} + S_{comp} = 2\psi + 2(q^t - 1)\varphi \tag{4}$$

In optimized EPSLB, each higher level tessellation, can be described as a tessellation of height 1. As TreeP is a balanced tree stucture, we can safely assume that for all tessellations number of participating nodes q^t is approximately same. So the total cost of performing optimized EPSLB on a TreeP is:

$$S_q = (h+1)\psi + \sum_{i=1}^{i=h-1} \{2\psi + 2(q^t - 1)\varphi\} \tag{5}$$

For a large peer to peer system structured as a TreeP, the number of nodes in a tessellation q^t is very small compared to the total number of nodes q, so the execution time of the optimized EPSLB algorithm is smaller on TreeP nodes.

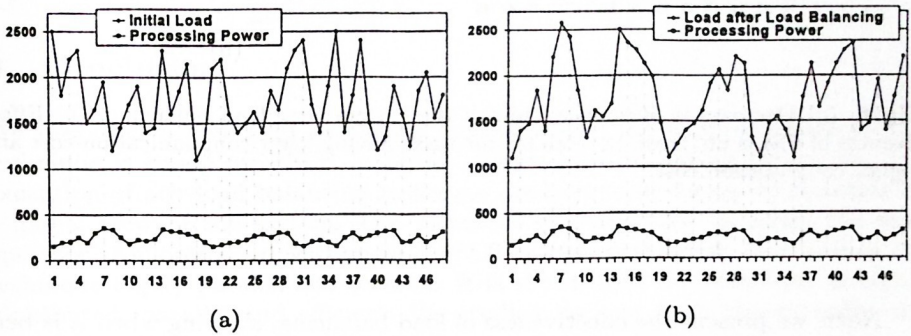
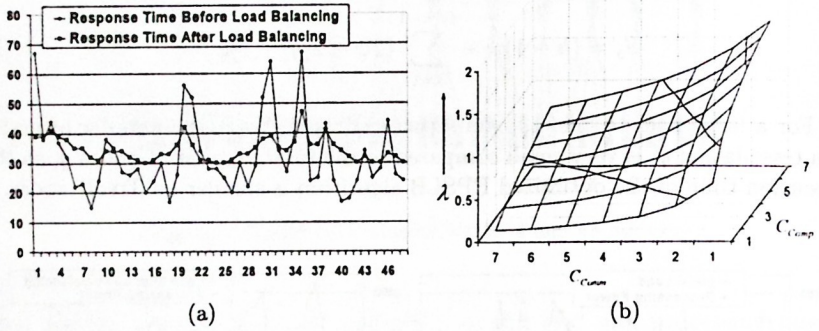


Fig. 5. (a) Load distribution before load balancing, (b) After load balancing, Load Distribution Curve perfectly corresponds to the Processing Power Curve. More powerfull processors have more work

7 Experimental Results

Experimental results presented in this section are based on the following parameters: the communication cost of each workload unit and the computational cost of a work load unit. The load is initially distributed among the nodes randomly. Figure 4 shows a TreeP of height 4 with 48 number of nodes. Figure 5(a) represents the processing powers of the individual nodes and the distribution of work units among the system before the load balancing. Figure 5(b) represents the distribution of work units after load balancing using the EPSLB algorithm. It can be clearly seen that powerful processors have more work. The

work is exactly balanced according to the relative normalized processing power of individual nodes. Figure 6(a) shows the response time of the individual nodes without load balancing and after load balancing. The graph clearly shows that the response time of the system after load balancing is better than the response time of the system without load balancing.



(a) (b)

Fig. 6. (a) Decrease in Response Time of the system after load balancing. (b) Effectiveness of PSLB on TreeP is more for the tasks having lower communication cost and higher computation cost

Next, we present the effectiveness of load balancing, showing when it is beneficial to perform load balancing. Let λ be the ratio of initial response time of the system to the balanced response time: $\lambda = \frac{RT_{init}}{RT_{bal}}$. Also let ϕ be the ratio of computation to communication cost of a workload unit: $\phi = \frac{C_{comp}}{C_{comm}}$. A value of λ more than one reflects that the response time of the system after load balancing is less than the response time of the system before load balancing, so the load balancing is effective. Figure 7 plots the crossover point showing clearly when it is beneficial to perform EPSLB. To calculate the crossover point, the EPSLB algorithm was executed on a network system of 48 nodes with an average workload of 85000 work units. The response time of the system with and without load balancing was calculated for different computation and communication costs, within a range of $\{1, 2, 3, 4, 5, 6, 7\}$. Figure 6(b) presents the effectiveness of EPSLB as the values of computation and communication costs change. The graph clearly shows that as computation cost of the system changes from 1 to 7, the effectiveness of the system increases. Figure 7 shows that when the computation to communication ratio ϕ is less than 1.75, the load balancing is not effective. But when this ratio is higher than 1.75, the EPSLB becomes very effective.

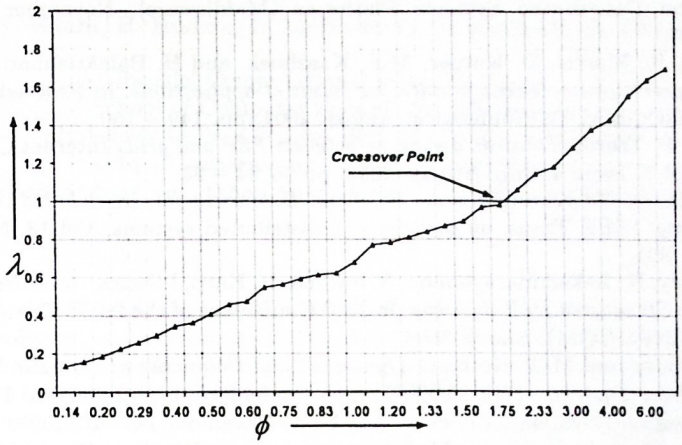


Fig. 7. Effectiveness of load balancing increases as the ratio of computation to communication cost of workload units increases

8 Conclusion

We proposed a new load balancing technique for peer to peer systems. The technique is based on PSLB, a pure dynamic load balancing technique. The execution of any load balancing technique requires some means of maintaining a global view of the system. The technique achieved this by using the scan operator. An optimized version of EPSLB is also developed, that makes the technique highly parallel and scalable. It is shown that the technique is highly distributed, parallel and efficient. We studied its cost both theoretically and experimentally.

References

1. M.W. Akhtar and M-T. Kechadi. *Dynamic Load Balancing on Irregular Networks Embedded in Hyper-Cubes.* 16th IASTED Intl. Conference on Parallel and Distributed Computing and Systems, MIT, Cambridge, MA, USA, November 9 – 11, 2004.
2. M.T Kechadi. David F. Hegarty. *Topology Preserving Dynamic Load Balancing for Parallel Molecular simulations.* In Proceedings of Supercomputing 97, 1997.
3. M.W. Akhtar and M-T. Kechadi. *Efficient Two-Pass Dynamic Load Balancing for Computational Clusters.* 23rd IASTED Int'l. Conference on Parallel and Distributed Computing and Neworks, Innusbruck, Austria, February, 15 – 16, 2005.
4. Benoit Hudzia, M-Tahar Kechadi, Adrian Ottewill. *TreeP: A tree based P2P network architecture.* IEEE International Conference on Cluster Computing,Boston, Massachusetts, USA, Setptember 27 – 30, 2005.
5. A. Rowstron and P. Druschel. *Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems.* IFIP/ACM International Con-

ference on Distributed Systems Platforms (Middleware), November 2001, pp. 329 – 350.
6. I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H. Balakrishnan. *Chord: A scalable peer-to-peer lookup service for Internet applications.* In Proceedings of the ACM SIGCOMM '01 Conference. August 2001. pp. 149 – 160.
7. D. Talia, P. Trunflo*Towards a synergy between P2P and grids* Internet Computing, IEEE. Vol 7, Issue 4, July-August 2003. pp.96, 94 – 95.
8. Ka-Po Chow, Yu-Kwong Kwok. *On Load Balancing for Distributed Multi-agent Computing.* IEEE Trans. on parallel and distributed systems, Vol 13, No 8, 2002. pp 787 – 801
9. B. Godfrey, K. Laksminarayanana, S. Surana, R. Karp, I.Stoica. *Load Balancing in Dynamic Structured P2P Systems.* In 23rd Conference of the IEEE Communication Society (INFOCOM), March 2004.
10. M.W. Akhtar and M-T. Kechadi. *Dynamic Load Balancing of Content Requests in Peer to Peer Systems.* 17th IASTED Intl. Conference on Parallel and Distributed Computing and Systems, Phoenix, AZ, USA, November 14 – 16, 2005.
11. P. Triantafillou, C. Xiruhaki, M. Koubarakis and N. Ntarmos. *Towards High Performance Peer to peer Content and Resource Sharing Systems.* In Proceedings of the Conference on Innovative Data Systems Research. CIDR, January 2003.
12. S. Kannan, M. Noar and S. Rudich. *Implicit representation of graphs.* STOC '88: Proceedings of the twentieth annual ACM symposium on Theory of computing, Chicago, Illinois, United States 1988, pp. 334 – 343.
13. C. Li and T.W. Ling. *An Improved Prefix Labeling Scheme: A Binary String Approach for Dynamic Ordered XML.* 10th International Conference on Database Systems for Advanced Applications, DASFAA 2005, Beijing. Vol 3453/2005 , April 2005, pp. 125 – –137.
14. D. Comer *Ubiquitous B-Tree.*ACM Computing Surveys (CSUR)Volume 11, Issue 2, June 1979, pp. 121 – 137.
15. Ka-Po Chow, Yu-Kwong Kwok. *On Load Balancing for Distributed Multi-agent Computing.* IEEE Trans. on Parallel and Distributed Systems, Volume 13, No 8, 2002. p 787 – 801
16. M.H. Willebeek-LeMair. A.P. Reeves. *Strategies for dynamic load balancing on highly parallel computers.* IEEE Trans. on parallel and distributed systems, Volume 4, No. 9, Sept. 1993.
17. Kuo-Liang Chung. *Prefix Computations on a Generalized Mesh-Connected Computer with Multiple Buses.* IEEE Transactions on Parallel and Distributed Systems. Volume 6, No 2, February 1995. pp 196 – 199.
18. J. Beal. *Paretless Distributed Hierarchy Formation.* Technical Report IA Lab MIT, 2003.
19. J. Beal. *A Robust Amorphous Hierarchy from Persistent Nodes.* In Proceedings of IASTED Conference on Communication Systems and Networks (CSN 2005). Benalmadena, Spain, September 8 – 10, 2003.
20. Mark Baker. *Cluster Computing White Paper.* Technical Paper University of Portsmouth,UK, December 2000.
21. K. Aberer. *P-Grid: A self-Organizing Access Structure for P2P information Systems.* Lecture Notes in Computer Science 2172, Springer-Verlag, Heidelberg, Germany, 2001. pp. 179 – 194.
22. I. Foster and C. Kesselman. *Globus: A Metacomputing Infrastructure Toolkit.* The International Journal of Supercomputer Applications and High Performance Computing. Volume 11. No. 2. 1997. pp 115 – 128.