World Academy of Science, Engineering and Technology
International Journal of Electrical and Computer Engineering
Vol:2, No:8, 2008

Load Balancing in Heterogeneous P2P Systems using Mobile Agents

Neeraj Nehra, R. B. Patel, and V. K. Bhat

***Abstract*—**Use of the Internet and the World-Wide-Web (WWW) has become widespread in recent years and mobile agent technology has proliferated at an equally rapid rate. In this scenario load balancing becomes important for P2P systems. Beside P2P systems can be highly heterogeneous, i.e., they may consists of peers that range from old desktops to powerful servers connected to internet through high-bandwidth lines. There are various loads balancing policies came into picture. Primitive one is Message Passing Interface (MPI). Its wide availability and portability make it an attractive choice; however the communication requirements are sometimes inefficient when implementing the primitives provided by MPI. In this scenario we use the concept of mobile agent because Mobile agent (MA) based approach have the merits of high flexibility, efficiency, low network traffic, less communication latency as well as highly asynchronous. In this study we present decentralized load balancing scheme using mobile agent technology in which when a node is overloaded, task migrates to less utilized nodes so as to share the workload. However, the decision of which nodes receive migrating task is made in real-time by defining certain load balancing policies. These policies are executed on PMADE (A Platform for Mobile Agent Distribution and Execution) in decentralized manner using *JuxtaNet* and various load balancing metrics are discussed.

***Keywords*—**Mobile Agents, Agent host, Agent Submitter, PMADE.

I. Introduction

LOAD balancing [1, 2] is a active technology that provides the art of shaping, transforming and filtering the network traffic then routing and load balancing it to the optimal node. By adding the concept of load balancer we can distribute the traffic for preventing from failure in any case by having capabilities such as scalability, availability, easy to use, fault tolerant, quick response time. Mobile agent technology offers a new computing paradigm in which an autonomous program can migrate under its own or host control from one node to another in a heterogeneous network. In other words, the program running at a host can suspend its execution at an arbitrary point, transfer itself to another host, or request the host to transfer it to its next destination and resume execution from the point of suspension is called mobile agent MA [3].

MA supports a variety of web based distributed applications namely: systems and distributed information Management [4] and information retrieval [5]. Other areas where MAs are seen as offering potential advantages are Wireless or mobile computing [6, 7] dynamic deployment of code, thin clients or resource limited devices, personal assistants, and MA-based parallel processing [8, 9]. Traditional load balancing approaches are implemented based on message passing paradigm [1, 10]. MA technology provides a new solution to support load balancing in heterogeneous network.

Moreover, a mobile agent based approach is flexible to incorporate new load balancing polices for various systems. MAs produce low network traffic. In message-passing based approaches, the nodes have to exchange messages of load information periodically in order to make decisions on load balancing. The mod_backhand [11] is such a load-balancing module for the Apache web server. The message exchanges result in high communication latency and thus deteriorate the performance of the system. Differently, a MA can migrate to a target server and interact to specified objects on the site.

The network traffic and communication latency can be largely reduced. MAs support asynchronous and autonomous operations. The nodes can dispatch MAs individually that travel independently between the nodes to perform various operations. A MA can encapsulate load balancing policies and travel to other node where it can make decision on load distribution according to the up-to-date state. Due to the merits of low network traffic and quick response time, MAs can strengthen the scalability of a system. In this paper we will execute the various load balancing policies for Peer-to-Peer system on PMADE (A Platform for Mobile Agent Distribution and Execution)[3,12]. These load balancing policies can achieve better performance than the message passing based approaches.

Rest of the paper is organized as follows. Overview of PMADE is provided in Section II, Section III discuses architecture for load balancing, policies decision, Section IV gives selection of policies, Section V gives selection of agents, Section VI presents performance study; Section VII gives related works, Section VIII concludes the article and future work.

II. Overview of PMADE

Fig. 1 shows the basic block diagram of PMADE. Each node of the network has an Agent Host (AH), which is responsible for accepting and executing incoming autonomous Java agents and an Agent Submitter (AS)[13], which submits the MA on behalf of the user to the AH. A user, who wants to perform a task, submits the MA designed to perform that task, to the AS on the user system. The AS then tries to establish a connection with the specified AH, where the user already

Neeraj Nehra is with School of Computer Science and Engineering, Shri Mata Vaishno Devi University, Katra (J&K), India (e-mail: nehra04@yahoo.co.in).

R. B. Patel is with Computer Engineering Department, M.M.Engineering College, Mullana (Ambala), Haryana, India (e-mail: patel_r_b@yahoo.com).

V. K. Bhat is with School of Applied Physics and Mathematics, Shri Mata Vaishno Devi University, Katra (J&K), India (e-mail: vijaykumarbhat2000@yahoo.com).

holds an account. If the connection is established, the AS submits the MA to it and then goes offline. The AH examines the nature of the received agent and executes it. The execution of the agent depends on its nature and state. The agent can be transferred from one AH to another whenever required. On completion of execution, the agent submits its results to the AH, which in turn stores the results until the remote AS retrieves them for the user. The AH is the key component of PMADE. It consists of the manager modules and the Host Driver. The Host Driver lies at the base of the PMADE architecture and the manager modules reside above it. It is the basic utility module responsible for driving the AH by ensuring proper co-ordination between various managers and making them work in tandem.

Details of the managers and their functions are provided in [12]. PMADE provides weak mobility to its agents and allows one-hop, two-hop and multi-hop agents [14]. PMADE has focused on Flexibility, Persistence, Security, Collaboration, and Reliability [3].

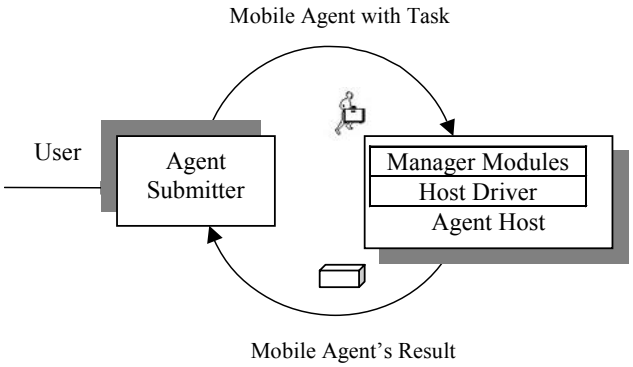

Fig. 1 Block Architecture of PMADE

III. Architecture for Load Balancing

The architecture for load balancing consisting of following components namely Interface, policy, and agents. Interface is used to communicate with the external world using PMADE. Policies are to be executed by the corresponding agent. We use the concept of virtual servers [15] for load balancing. A virtual server looks like a single peer, but each physical node can be responsible for more than one virtual server. The key advantage of splitting load into virtual servers is that we can move a virtual server from any node to any other node in the system. This operation looks like a leave followed by a join. Even though splitting load into virtual servers will increase the path length on the overlay, we believe that the flexibility to move load from any node to any other node is crucial to any load-balancing scheme. We have identified certain policies and agents for the architecture, which will be discussed next.

IV. Policy

In this section, we present three simple load-balancing schemes. All these schemes try to balance the load by transferring virtual servers from heavily loaded nodes to lightly loaded nodes. The key difference between these three schemes is the amount of information required to make the transfer decision. In the simplest scheme, the transfer decision involves only two nodes, while in the most complex scheme, the transfer decision involves a set consisting of both heavy and light nodes. We first define the notion of heavy and light nodes.

(a) Heavy and Light Nodes: Let L_i denote the load of node i , where L_i represents the sum of the loads of all virtual servers of node i. We assume that every node also has a target load (T_i). A node is considered to be heavy if $L_i > T_i$, and is light otherwise. The goal is to decrease the total number of heavy nodes in the system by moving load from heavy nodes to light nodes.

(b) Virtual Server Transfer: The fundamental operation performed for balancing the loads is transferring a virtual server from a heavy node to a light node. Given a heavy node h and a light node l, we define the best virtual server to be transferred from h to l as the virtual server v, the transfer of which satisfies the following constraints:

- Transferring v from h to l will not make l heavy.
- v is the lightest virtual server that makes h light.
- If there is no virtual server whose transfer can make h light, transfer the heaviest virtual server v from h to l.

(c) Splitting of Virtual Servers: If no virtual server in a heavy node can be transferred in its entirety to another node, then a possibility is to split it into smaller virtual servers and transfer a smaller virtual server to a light node. While this would improve the time taken to achieve balance and possibly reduce the total load transferred, there is a risk of excessively fragmenting the identifier space. Hence, a scheme to periodically merge virtual servers would be needed to counteract the increase in the number of virtual servers caused by splitting. Further policies are classified into

One-to-One: The first policy is based on a one-to-one mechanism, where two nodes are picked at random. A virtual server transfer is initiated if one of the nodes is heavy and the other is light. This policy is easy to implement in a distributed fashion. Each light node can periodically pick a random task and then perform a lookup operation to find the node that is responsible for that task. If that node is a heavy node, then a transfer may take place between the two nodes. In this scheme only light nodes perform probing; heavy nodes do not perform any probing. There are three advantages of this design choice. First, heavy nodes are relieved of the burden of doing the probing second, when the system load is very high and most of the nodes are heavy, there is no danger of either overloading the network or thrashing. Third, if the load of a node is correlated with the length of the space owned by that node, a random probe performed by a light node is more likely to find a heavy node.

One-to-Many: Unlike the first scheme, this scheme allows a heavy node to consider more than one light node at a time. Let h denote the heavy node and let $l_1,l_2,l_3, \ldots,l_k$ be the set of

light nodes considered by h. For each pair (h_i,l_i) we pick a virtual server v_i . Among the virtual servers that this procedure gives, we choose the lightest one that makes heavy node h light. If there is no such a virtual server, we pick the heaviest virtual server among the virtual server v_i $(1 \leq i \leq k)$ to transfer. We implement these policies by defining the role of each mobile agent, which is to be discussed in coming sections.

Many-to-Many: This scheme is a logical extension of the first two schemes. While in the first scheme we match one heavy node to a light node and in the second scheme we match one heavy node to many light nodes, in this scheme we match many heavy nodes to many light nodes. Our goal is to bring the loads on each node to a value less than the corresponding target. To allow many heavy nodes and many light nodes to interact together, we use the concept of a global pool of virtual servers, an intermediate step in moving a virtual server from a heavy node to a light node. The pool is only a local data structure used to compute the final allocation.

The policy consists of three phases:

1. Unload: In this phase, each heavy node h transfers its virtual servers greedily into a global pool till it becomes light. At the end of this phase, all the nodes are light, but the virtual servers that are in the pool must be transferred to nodes that can accommodate them.
2. Insert: This phase aims to transfer all virtual servers from the pool to light nodes without creating any new heavy nodes. This phase is executed in stages. In each stage, we choose the heaviest virtual server v from the pool, and then transfer it to the light node k determined using a best-fit heuristic, i.e., we pick the node that minimizes $T_k - L_k$. This phase continues until the pool becomes empty, or until no more virtual servers can be transferred.
3. Dislodge: This phase swaps the largest virtual server v from the pool with another virtual server v' of light node i such that $L_i + \text{load}(v) - \text{load}(v') \leq T_i$

V. Agents

We require three agents, out of which two are mobile agents and one is stationary. A brief look of these agents is as follows:

- The routing agent (RA) is stationary agent responsible of updating the routing table that resides at each node. It carries a routes vector table containing the communication cost from the assigned node to each other node in the network. This table has a lifetime measured by the number of the hops the routing agent is allowed to perform before updating the vector table. The routing agent plays an important role in informing each node in the network about the addresses of other nodes and if a failure of a node or a link is detected, it is the role of the routing agent to spread it over the network by copying the new table and migrating.
- Load Information Agent (LIA) is a MA responsible for information gathering. It travels around the nodes (virtual server), collects the load information, and meanwhile propagates the load information to each node.
- Location agent (LA) is a mobile agent and is activated whenever an overload situation arises on nodes (virtual server). Its job is to find the suitable receiver partner for the overloaded node that launched it.

VI. Performance Evaluation

Mobile agent based load balancing schemes are executed on network consisting of 100 nodes operating over Fast Ethernet (100M bit/sec.) and employing a decentralized *JuxtaNet [16, 17]* which is an open and decentralized peer-to-peer network model. The JuxtaNet is significant in the sense that it is an open, general-purpose P2P network model. JXTA's *JuxtaNet* is abstracted into multiple layers namely core, service and application with the intention that multiple services will be built on the core. The core and services will support multiple applications. In fact, there is no constraint against the simultaneous existence on the JuxtaNet of multiple services or applications designed for a similar purpose. As an example, just as a PC's operating system can simultaneously support multiple word processors, the JuxtaNet can simultaneously support multiple file-sharing systems. Category 5 (twisted-pair) copper wire runs among the PCs and an Ethernet hub, enabling users of those networked PCs to access each others resources in a decentralized manner.

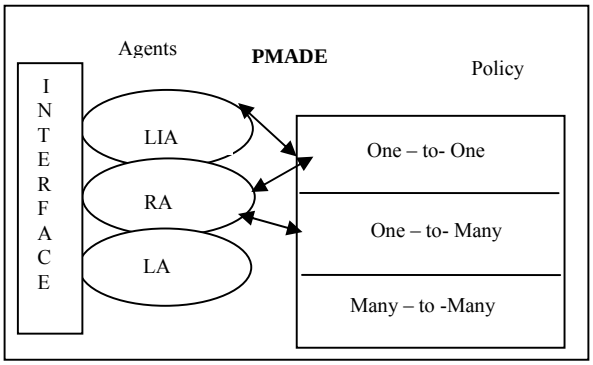

Fig. 2 Architecture for Load Balancing

PMADE and J2SDK 1.5 are used for result analysis by comparing its performance with mod_backhand [11] (A load balancing approach based upon message passing paradigm). The load on different peers is measured at different time interval. The length of job queue denotes the load on peer.

First, while we do not restrict ourselves to a particular type of resource (storage, bandwidth or CPU), we assume that there is only one bottleneck resource we are trying to optimize for. Second, we consider only schemes that achieve load balancing by moving virtual servers from heavily loaded nodes to lightly loaded nodes. Such schemes are appropriate for balancing storage in distributed file systems, bandwidth in systems with a web-server like load, and processing time when serving dynamic HTML content.

The AS is used to generate request to peers. Performance of various load balancing policies and their impact on load is measured by the following metrics.

- System Throughput: The overall throughput is measured in number of requests processed per second.
- Network Traffic: The overall communication overhead is measured in the total number of bytes transferred in the communication and number of probes for each individual policy.

Fig. 3 shows the system throughput of two approaches along x axis number of clients requests is a factor of 100 while along y axis request satisfied /sec is shown. Clearly mobile agent approach is better than traditional message passing paradigm in metric of System throughput.

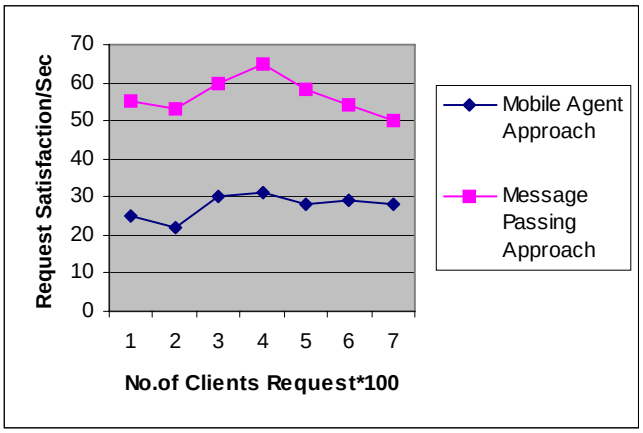

Fig. 3 System Throughput

Fig. 4 compares the network traffic using mobile agent approach and message passing approach. It clearly shows that MA approach generates low communication delay compared to the message passing approach.

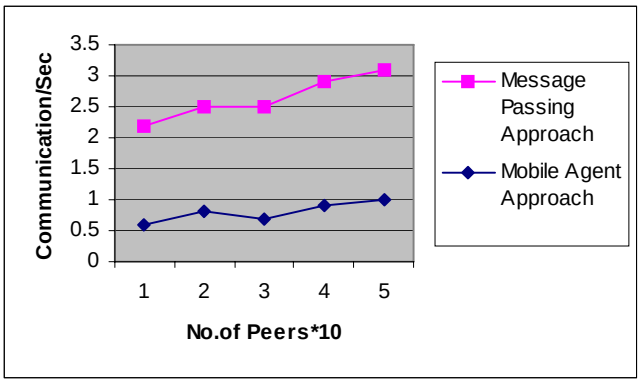

Fig. 4 Network Traffic

In Fig. 5 we plot the total number of probes performed by the heavy nodes before they completely transfer their excess load to light node. This graph shows that one to one scheme is sufficient if load remains stable over long period of time and if the control traffic overheads do not affect the system too much.

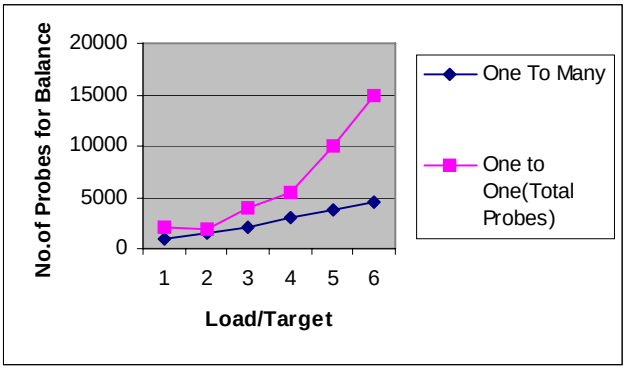

Fig. 5 The number of probes required for all nodes to become light

Fig. 6 compares system throughput of the mobile agent approach and the case without load balancing. This result shows that MA approach improves the system throughput while increasing the number of peers, but there is no improvement in system throughput without load balancing.

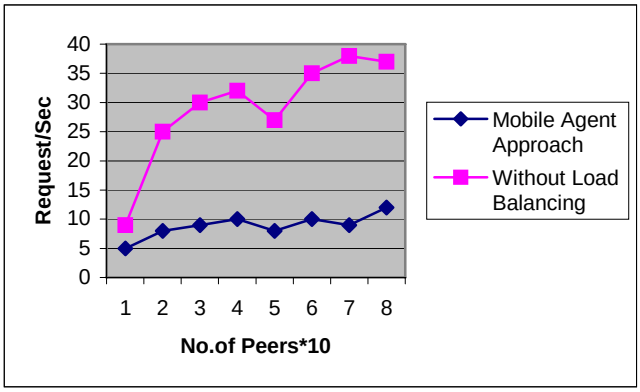

Fig. 6 System throughput using mobile agent and the case without load balancing

VII. Related Work

Load balancing is indispensable for a P2P system to assure even distribution of workload on each peer. But one of the most difficult problems that arise on P2P system is the selection of an efficient load balancing policy. The load balancing policy should aim for evenly utilized Peers and a minimum response time for the processed requests. Under standard methodology load selection is done randomly. The random selection cannot guarantee load balancing. Round robin is widely used because it is easy to implement and implies only a minimum overhead. A variation of round robin policy is the weighted round robin policy [18]. With weighted round robin the incoming requests are distributed among the peers on a round robin fashion, weighted by some measure of the load on each of the peers.

Another techniques, which is called dispatching techniques which when implemented by network address translation or other methods (such as HTTP redirection), introduce higher overhead than does network load balancing. This limits throughput and restricts performance. SUNSCALAR [19] provides load balancing by using both approaches, i.e., Dispatcher and Round Robin.

But Today's most well known peer-to-peer applications are Napster, Gnutella [20], and Freenet [21], and various research projects have been initiated in the past few years, such as Pastry [22] and Chord [23]. Although the different peer to-peer applications share the same notion of peer-to-peer networking, the intended usage and approach varies from application to application. Napster and Gnutella are primarily file-sharing applications: exchange of files between peers.

Napster's approach to information search is traditionally client-server, while Gnutella adheres more to the peer-to-peer philosophy and forwards information search requests to its neighboring peers in the network. (Although Gnutella recently introduced super nodes and client nodes for more scalable information retrieval.) Freenet is more like a distributed information storage system. It pools unused disk space across potentially hundreds of thousands of desktop computers to create a collaborative virtual file system. Pastry provides a scalable, distributed object location and routing infrastructure for wide-area pee r-to-peer applications. It can be used to support a variety of peer-to-peer applications, including global data storage, data sharing, and group communication and naming. Chord, on the other hand, focuses on a scalable peer-to-peer lookup service to efficiently locate the node that stores a particular data item. Chord provides support for just one operation: given a key, it maps the key onto a node. The JXTA project from Sun Microsystems [16, 17] works on core network computing technology to provide a set of simple, small, and flexible mechanisms that can support peer-to-peer computing. The focus is on creating basic mechanisms and leaving policy choices to application developers.

The self-organizing behavior of peer-to-peer networks has also been studied. In particular scalability, fault tolerance, and security have been subject of study. It has been observed that peer-to-peer networks organize themselves into a "small-world" networks [24, 25], which are typically characterized by a power-law distribution of the edge degree. In such a distribution, the majority of nodes have relatively few local connections to other nodes, but a significant small number of nodes have large wide-ranging sets of connections. Even in very large networks, the small-world topology enables short paths because these well-connected nodes provide shortcuts. Small-world networks are surprisingly resistant to random errors, because random failures are most likely to eliminate nodes from the poorly connected majority of nodes. But the feature that makes it immune to accidents also makes it vulnerable to attacks if the well-connected nodes are targeted.
A framework for load balancing using MA named EALBMA (Efficient and Adaptive Load Balancing based on MA)[26] has been made in which a novel algorithm for updating load information partially based on MA which is called ULIMA.

MA support load balancing in parallel and distributed computing [8,14], e.g., Traveller [27] using resource broker. It implements parallel application such as L. U. Factorization and sorting. MESSENGERS [6] is a system for general-purpose distributed computing based on MAs. It supports load balancing and dynamic resource utilization. Flash [28] is a framework for the creation of load balanced distributed application in heterogeneous cluster system.

VIII. Conclusions and Future Work

In this paper we have implemented different load balancing policies on P2P systems and studied the various metrics affecting these policies using mobile agent approach. The performance evaluation shows that Mobile agent approach is far better than the traditional load balancing approach in heterogeneous P2P network. In the future work we would like to implement this approach to Cluster of PCs and Grid Computing and try to measure different metrics regarding these systems for load balancing. Also fault tolerance would be studied and its impact on load balancing policies.

References

[1] Dias, D., Kish, W., Mukherjee, R. and Tewari, R., A Scalable and Highly Available Web-Server, in Proc.41st International Computer Conference (COMPCON'96), IEEE Computer Society, SanJose, CA, 1996, pp. 85-92.
[2] W. Tang, M. Mutka, Load Distribution via Static Scheduling and Client Redirection for Replicated Web Servers, in Proc. 1st International Workshop on Scalable Web Services (in conjunction ICPP 2000), Toronto, Canada, 2000, pp. 127-133.
[3] Patel, R. B., Design and Implementation of a Secure Mobile Agent Platform for Distributed Computing, PhD Thesis Department of Electronics and Computer Engineering, IIT Roorkee, India, Aug. 2004.
[4] Jonathan Dale, A Mobile Agent Architecture for Distributed Information Management, Ph.D. thesis,Univ. of Southampton, Sept. 1997
[5] Haverkamp, D. S. and Gauch, S., Intelligent Information Agents: Review and Challenges for Distributed Information Sources, in Journal of the American Society for Information Science, 49(4): 304-311, 1998.
[6] Chess, D., B. Grosof, Harrison, C., Levine, D.,Parris, C. and Tsudik, G., Itinerant agents or mobile computing, IEEE Personal Communications Magazine, 2, pp. 34-49, Oct. 1995.
[7] Imielinsky, T. and Badrinath, B. R., Wireless Computing: Challenges in Data Management,Communication of the ACM, 37(10): 18-28, 1994.
[8] Al-Jaroodi, J., Mohamed, N., Jiang Hong and Swanson, D., A Middleware Infrastructure for Parallel and Distributed Programming Models on Heterogeneous Systems, IEEE Transactions on Parallel and Distributed Systems, Special Issue on Middleware, 14(11): 1100-1111, Nov. 2003.
[9] Al-Jaroodi, J., Mohamed, N., Jiang Hong and Swanson, D., An Agent-Based Infrastructure for Parallel Java on Heterogeneous Clusters, in Proceedings of the IEEE International Conference on Cluster Computing, IEEE, Nov. 2002.
[10] Cardellini, V. and Colajanni, M., Dynamic Load Balanc ing on Web-server Systems, IEEE Internet Computing, 3, pp. 28-39, 1999.
[11] Schlossnagle, T., The Backhand Project: Load balancing and Monitoring Apache Web Clusters, in Proceedings Apache Con Europe 2000, London, Britain, mod_backhand, http://www.backhand.org/mod_backhand
[12] Patel, R.B. and Garg, K., PMADE – A Platform for mobile agent Distribution & Execution, in Proceedings of 5th World MultiConference on Systemics, Cybernetics and Informatics (SCI2001) and 7th International Conference on Information System Analysis and Synthesis (ISAS 2001),Orlando, Florida, USA, July 22-25, 2001, Vol. IV, pp. 287-293.
[13] Patel, R. B. and Garg, K., A New Paradigm for Mobile Agent Computing, WSEAS Transaction on Computers, Issue 1, Vol. 3, pp. 57-64, Jan. 2004.
[14] Patel, R.B. and Garg, K, A Flexible Security Framework for Mobile Agent Systems Control and Intelligent Systems, 33(3): 175-183, 2005.
[15] F. Dabek and M. F. Kaashoek and D. Karger and R. Morris and I. Stoica. "Wide-area Cooperative Storage with CFS", Proc. ACM SOSP 2001.
[16] L. Gong. JXTA: A network programming environment.IEEE Internet Computing, 5(3):88–95, May/June 2001
[17] Sun Microsystems, Inc. Project JXTA: An open, innovative collaboration. White Paper, http://www.jxta.org/project/-www/docs/OpenInnovative.pdf>, Apr. 2001.
[18] CiscoSystemsInc.LocalDirector. http://www.cisco.com

[19] Singhai, A., Lim, S. B. and Radia S. R., The SunSCALR Framework for Internet Servers, IEEE FaultTolerant Computing Systems, Jun 1998.
[20] D. Clark. Face-to-face with peer-to-peer networking. Computer, 34(1):18–21, Jan. 2001.
[21] I. Clarke, S. G. Miller, T.W. Hong, O. Sandberg, and B.Wiley.Protecting free expression online with Freenet. IEEE Internet Computing, 6(1):40–49, Jan./Feb. 2002
[22] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In Middleware 2001, volume 2218 of Lecture Notes in Computer Science, pages 329–350, Berlin, Germany, 2001. Springer-Verlag.
[23] 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 2001Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM'01), pages 149–160, San Diego, CA, Aug. 2001.
[24] M. A. Jovanovic. Modeling peer-to-peer network topologies through "small-world" models and power laws. In Proceedings of the IX Telecommunications Forum (TELFOR 2001), Belgrade, Yugoslavia, Nov. 2001.
[25] D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. Nature, 393(6684):440–442, June 1998.
[26] Server Iron Chassis L4-7 Software Configuration Guide. <http://www.foundrynet.com/services/documentati on /sichassis/management.html>
[27] C. -Z. Xu and Wims, B., Mobile Agent Based Push Methodology for Global Parallel Computing, Concurrency and Computation: Practice and Experience, 14 (2000), pp. 705-726.
[28] Obeloer, W., Grewe, C. and Pals, H., Load anagement with Mobile Agents, in Proc. 24th EUROMICRO Conference (EUROMICRO98), vol.2, Vasteras, Sweden, 1998, pp. 1005-1012.

Nehra Neeraj is Lecturer (School of Computer Science and Engineering), Shri Mata Vaishno Devi University, Katra(India). His research is focused on use of agents, mobile computing, parallel/distributed computing, multiagent system and fault tolerance. Prior to joining SMVDU, Katra he has worked with HEC Jagadhri and MMEC Mullana, Ambala, Haryana, India.

Dr. R. B. Patel received PhD from IIT Roorkee in Computer Science & Engineering, PDF from Highest Institute of Education, Science & Technology (HIEST), Athens, Greece, MS (Software Systems) from BITS Pilani and B. E. in Computer Engineering from M. M. M. Engineering College, Gorakhpur, UP. Dr. Patel is in teaching and Research & Development since 1991. He has published about 50 research papers in International/National Journals and Refereed International Conferences. He has been awarded for Best Research paper by Technology Transfer, Colorado, Springs, USA, for his security concept provided for mobile agents on open network in 2003. He has written 5 books for engineering courses. He is member of various International Technical Societies such as IEEE-USA, Elsevier-USA, Technology, Knowledge & Society-Australia, WSEAS, Athens, etc for reviewing the research paper. His current research interests are in Mobile & Distributed Computing, Mobile Agent Security and Fault Tolerance, development infrastructure for mobile & peer-to-peer computing, Device and Computation Management, Cluster Computing, etc.

Dr. V. K. Bhat is a Assistant Professor, SMVDU Katra, India. His research is focused on Ring theory, Graph theory, and discrete structure. He has 11 years of teaching and research experience. He has published 19 papers in international/national journals and 18 papers in international/national conference proceedings. He is a recipient of UGC (SRF and JRF) fellowship.