Grid Computing integrated with Mobile computing wireless devices

ShipraGoel

(Lecturer in Dr.KedarNathModi Institute of Engineering and Technology)
Department of MCA
shipra_goelred@yahoo.co.in

Abstract

One application domain the mobile computing has not yet entered is that of grid and cluster computing: the aggregation of network-connected computers to form a large scale, distributed system that can be used for resource intensive scientific or commercial applications in a scalable and cost effective manner. This research paper views grid and cluster computing nodes as small scale devices comprises laptops, tablets, PDA's etc that can be connected through an internet, potentially through wireless links. The integration of these devices leads to many difficulties because small scale devices are heterogeneous and lack the computational, bandwidth, storage and availability characteristics commonly required for high performance in distributed computing. So this research paper tries to resolve these difficulties

Keywords

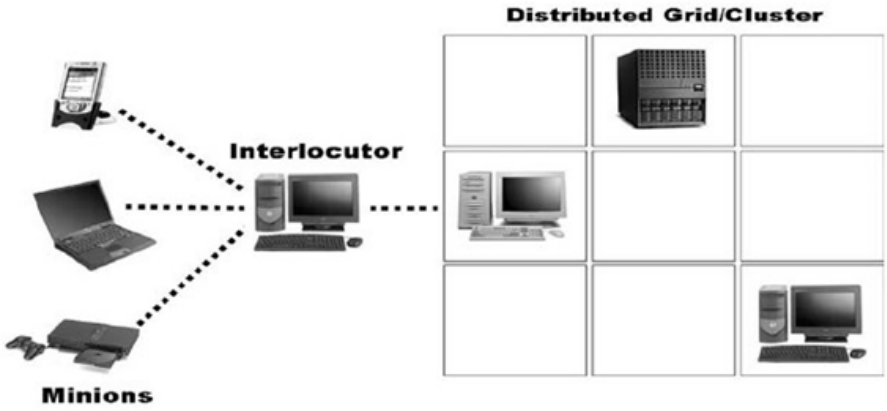
Mobile computing,GridComputing,WirelessDevices,LEECH.

1. Introduction

Mobile computing deals with mobility and grid computing deals with heterogeneity. This research paper tries to integrate grid and cluster small scale nodes in a mobile computing environment. However many technological, commercial and consumer trends supports the inclusion of such devices in a computational grid. First Moore's Law of increasing transistor density will continue to drive CPU performance in small scale devices. Second availability of wide area wireless communications will be more prevalent. Third consumer use of small scale intelligent electronics continues to grow yearly. Fourth trends in pervasive computing suggest a future where small devices will be predominant form factor of choice. With these observations, This research paper devised LEECH(Leveraging Every existing computer out there).To mitigate the effects of wide heterogeneity and unpredictable availability commonly associated with small scale devices a LEECH architecture is designed using a hierarchical designing to abstract away the underlying devices. Specifically we used proxy based cluster infrastructure to provide small scale devices with favorable deployment, interoperability, scalibility, adaptivity resiliency characteristics as shown in figure1. In this figure the proxy additionally serves important roles of service negotiator and resource request partitioned for abstract classes of devices in its group.

DOI : 10.5121/ijmnct.2011.1203

Fig1 A broad view of LEECH Proxy based Clustered architecture



2. LEECH Architecture

Give that use of small scale devices in a grid is compelling and potentially useful; a system architecture must be constructed to facilitate the integration.

2.1 Key challenges:

A naïve approach to architecture would be simply to run grid cluster software on the small scale devices, connect the devices together and allow the devices to behave and assume the same responsibility as typical desktop PC nodes. Although this approach may work in a practice a number of significant obstacles will be encountered:

a) Grid/cluster overhead:-There is a memory and CPU use overhead incurred for running grid or cluster software. It is unlikely that small scale devices particularly PDA devices with limited memory and CPU performance, will be able to operate as full-fledged nodes.

b) Device heterogeneity: - The variety of devices is potentially large. Workload spreads across such machines cannot be generalized.

c) Scalability and management: - even for existing distributed systems ,scalability is a major issue. When one considers the inclusion of hundreds of small scale devices, scalibility and handling of these machines become a larger issue.

d) Service discovery: - Small scale devices must be able to find grid nodes in order to participate.

e) Dynamic, unpredictable availability: - device owners are privileged to turn off their devices at times of their own choosing.

f) Power consumption:- of course the power management requirements of devices is a major factor.

g) Multi-hop wireless network participation:- If the devices participate in a wireless network, what are the ramifications with regards to routing and discovery?

2.2 Overview of architecture:

To address these issues, we chose a proxy-based clustered architectural approach to integrating small-scale, heterogeneous devices to computational grids and clusters. The LEECH architecture enables communication between small-scale, heterogeneous devices and a computational grid or distributed system via a proxy middleware. We term the proxy node an interlocutor and we call the small-scale, heterogeneous devices to which it is connected its minions. In our system, minions are closely associated with an interlocutor, which in turn is responsible for hiding the heterogeneity of its minions from the rest of the grid system. We suggest that interlocutors can support a large number of minion devices and that the aggregation of many minions' resources can be presented to the distributed system as an interlocutor's own resources. This hierarchical organization, similarly seen on the Internet with hierarchical routing and domain name system (DNS), improves scalability by intentionally limiting the number of devices that is globally visible. On each interlocutor and minion, we instantiate a daemon process that facilitates communication and interactivity within the LEECH system. The interlocutor daemon provides functionality for service discovery, session management, adaptive control, and job scheduling. In a similar fashion, the minion daemon handles service discovery beaconing, application fragment management, and session control.

2.3 Grid/Cluster and LEECH

In Fig2 we show the interaction between nodes using MPI and LEECH. In the context of MPI alone, programmers develop their parallel applications while using MPI library calls (such as MPI_Send and MPI_Recv) to send and receive buffers of data. An interconnectivity fabric, whether a communication bus within a multiprocessor, an Ethernet network within a LAN, or the Internet for distributed grids, facilitates message delivery. We also note that distributed shared memory can be used as a programming model for distributed applications, but in this chapter we focus on message-passing applications. Figure 2 also shows how the LEECH architecture fits in with existing systems. The existing grid or cluster remains unchanged, save for the execution of LEECH components on chosen nodes, which act as both a grid node as well as an interlocutor. Communication between the interlocutors with other grid nodes still uses MPI, although communication between the interlocutor and the minions is through our LEECH API.

other grid nodes still uses MPI, although communication between the interlocutor and the Minions is through our LEECH API.

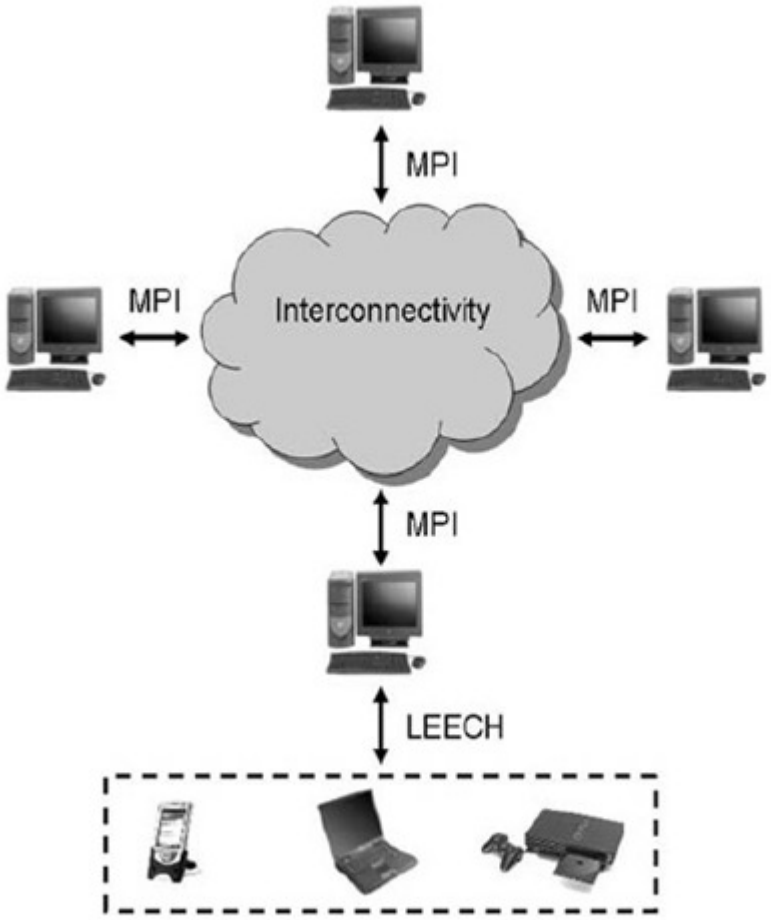


Fig2 :- Communication Architecture of a Distributed Application Using LEECH as Part of a Grid/Cluster System

2.4 Application Major Component and Minor Component

An application that runs atop the LEECH architecture is divided into two parts — the major component and the minor component. The major components contain the application logic and MPI communication calls as well as LEECH calls. Note that in the absence of LEECH, the major component is executed on the grid/cluster like any other MPI program. On the other hand, using the application within the LEECH architecture requires that the major component be written to contain LEECH communication calls as well as programming logic to partition tasks and data among the minions. Additionally, if the major component runs within LEECH but without minions, then the programming logic should appropriately handle normal computational activities within the cluster or grid. The minor component runs on minion devices and should be written to contain the application logic to handle an apportioned segment of computation and data. This minor component communicateswith a major component through the LEECH API. This

configuration allows the interlocutor running the major component to interact with other cluster nodes (via MPI) on behalf of the minions running the minor components. The logical division of a distributed application into our major component and minor component allows code executed on the minion nodes to be smaller than the entire distributed application and thus more easily executed on small-scale devices with limited memory. Furthermore, the grid or cluster MPI runtime overhead may be too large for such devices, so MPI communication does not reach the minor component in our architecture. This application division also introduces a difference between LEECH applications and typical MPI applications: MPI applications are typically data parallel, but LEECH applications can be task parallel or data parallel.

2.5 Interlocutor

A LEECH interlocutor node is itself a grid/cluster node that has been deemed an interlocutor by its administrators based on a machine's hardware and network capabilities. Interlocutor nodes should be placed strategically such that they are geographically close to a focal point where a large number of minions often come online to reduce network latency between the minion and the interlocutor. In a wireless LAN environment, interlocutors should be collocated with wireless base stations where large numbers of wireless minions regularly connect. Current distributed/grid architectures require nodes to have fairly powerful processors, more than moderate amounts of primary and secondary storage, and reliable network connections. The small-scale devices we consider do not meet these requirements, so interlocutor nodes are typically mid-range desktop PC or workstation systems or greater Furthermore, an interlocutor may itself be a work node and perform a part of an application's computation, but we leave that software design choice to the application programmer. Although an interlocutor is designed to be coupled with a grid or cluster system through the distributed application itself, it is important to note that the LEECH system, in particular the daemons and the communication library, can run standalone (i.e., without MPI or Globus) outside of a grid or cluster. In this mode, a parallel message-passing application can still be written using only our APIs and daemons to run within an isolated group of minions and an interlocutor. It is also possible for a LEECH system to be organized in a hierarchical manner, such that an interlocutor can be a minion of another interlocutor. Because the focus of this chapter is on LEECH itself, we will leave a closer analysis of LEECH's interaction with other grid infrastructures like Globus for future work. The interlocutor software is a daemon process logically divided into four main services or components. The service discovery server (SDS) allows minions to discover and register with interlocutors in the vicinity. The major component handles connections and communication with the major component running on the same machine as the interlocutor daemon. This session manager allows major components to post new work to be computed and pick up results, if any, from completed computations. The interlocutor- to-minion session manager acts as a server waiting for connection requests from minions and handles communication sessions with its minions. The interlocutor-to-minion session manager communicates directly with the minion-to-interlocutor session managers in its minions.Finally, the job manager and availabilityadaptation scheme handles the scheduling and assignment of jobs to minions, working closely with the interlocutor-to-minion session manager.

2.6 Minion

A minion executes a LEECH minion daemon process responsible for:

• Receiving a job from the interlocutor with which it has registered

• Making some computation based on the job's data
• Sending results back to the interlocutor
• Repeating the three previous steps, until going offline

The minion daemon process is the middleman between the minor component and the interlocutor. The minion, much like the interlocutor, has three main components:
1. Service discovery agent (SDA)
2. Minion component session manager
3. Minion-to-interlocutor session manager

However, unlike the interlocutor, the minion's services run serially to allow operation on small-scale devices whose operating systems may not support multi-threading. The SDA allows a minion to discover local interlocutors. The SDA advertises minion services to interlocutors it has discovered. The minion component session manager handles connection setup and teardown and communication with minor components running on the same small-scale machine as the minion daemon process. Finally, the minion- to- interlocutor session manager handles connections and communication with interlocutor daemons that have been discovered by the minion's SDA. The minion-to-interlocutor session manager works directly with the interlocutor's interlocutor-to-minion session manager.

2.7 Availability Adaptation and Job Management

On the small-scale class of devices we are considering, we cannot expect results returned from a node that has gone offline and will not be coming back online. The grid community has identified this unreliability problem but has not yet addressed it . Typical MPI programming idioms deal with node failures with fail-stop semantics. Instead, resiliency can be facilitated by the parallel programming library or by the application itself. In LEECH, we provide support to handle dynamic minion availability within the library because low availability (and even periods of complete disconnectivity) is common. We have developed an availability adaptation scheme to gracefully accommodate mobile systems that come and go in and out of the network before completing the computation of some job whose results must eventually be submitted to the interlocutor. The two main features that facilitate our availability adaptation scheme are the LEECH job and replicated job assignments. A LEECH job is a partition of the computation and data of an application. More specifically, it comprises a unique set of initial data, the computation of this data by one or more minion nodes, and its results. A job is created at the interlocutor by calling a LEECH send function and its results are collected by calling a LEECH receive function. We chose to keep minions anonymous from the major component rather than to provide named communication. If we supplied the application programmer with a means of establishing and maintaining named communication between the interlocutor and specific minion,Providing resiliency would mean adding checkpoints for each set of communications back and forth between each interlocutor and minion pair and restoring checkpoints when failures occur. We can easily see how adapting to dynamic availability, but providing direct, named communication would cause LEECH to grow in complexity, shrink in scalability, and lose significant potential performance gains. The LEECH job supports the communication model above by allowing interlocutors to submit and collect jobs from minion classes rather than from individual minions. Each job is individualized with a unique job ID (JID). The LEECH interlocutor generates JIDs and manages the mappings between jobs and the minions; however, this mapping is invisible to the application programmer. A job uses one round trip communication exchange between the

interlocutor and a minion. If a minion fails or goes offline unexpectedly, the interlocutor's SDS component's lease manager notifies the job manager, which simply reassigns the job to a new minion. The subsequent computation of this reassignment entails only one job's amount of additional work and only one repeated transmission.

3.Experiments and analysis

Here we analyze the performance of the LEECH library against MPI on a test bed of small-scale devices running parallel applications. Our results will show that MPI is a poor choice for communication within a small-scale device environment. In particular, we will show that LEECH has lower communication overhead (particularly relevant for communication-intensive programs), exhibits better resiliency in the face of minion failure, and provides flexible tools for more convenient data and computation partitioning. Due to chapter space constraints, we omit results from our applications that use the interlocutor and minions within a larger grid cluster (where the interlocutor acts as a grid node). This larger scenario already encompasses the interactivity between the interlocutor and the minions, which is the critical path we are studying. The experiments in this section were chosen to represent classes of communication-intensive or computation-intensive applications. For the latter, although the interlocutor's major component could have been written to perform application computation, we chose to have the interlocutor execute only partitioning and daemon functionality; this allowed us to focus on the performance of the minions in computationally intensive programs. We wanted a broad range of small-scale devices that we could easily program. We thus chose a set of low-end laptops (Dell™ 3800 and 4000 models over 2 Mbps 802.11b running Red Hat Linux), PDAs (Compaq iPAQ 3650 and 3670 models over 2 Mbps 802.11b running Familiar Linux), and a PlayStation2 (connected with Fast Ethernet) with varying CPU, storage, and communication characteristics. All of the machines have some form of Linux installed on them. The PlayStation 2 in particular used the Linux Kit from Sony, which provided a special Red Hat distribution along with a hard drive and Ethernet adapter. Except for the synthetic communication experiment, we used a Sun Blade 100 workstation running Solaris™ operating system as the interlocutor. All test bed devices used alternating current (AC) power during the course of the experiments. In future work, as our test bed is expanded, we will look at how power constraints affect the system, particularly when users are working on the devices at the same time. We will look into relevant power metrics and heuristics to react to them.

3.1 Synthetic Application for Measuring Communication Overhead

In our first experimental set, we wanted to show the overhead of the LEECH communication library against that of MPI. We used MPICH, a popular implementation of MPI used in current Beowulf cluster computing. We installed MPICH version 1.25 on all our machines except for the iPAQ PDAs because our cross-compiler could not compile the MPICH code. Another notable fact is that the memory footprint of MPICH's MPD messaging daemons is over 800 kilobyte (KB), whereas our LEECH minion daemon required only 400 KB. Although the PDAs could have run the MPICH daemons in theory, other more memory-constrained devices, particularly. In fig .3, we show the execution of a synthetic communication benchmark, written as both LEECH and MPI versions, to reveal the round trip latencies between two laptops connected over an 802.11 network. The times measured for both versions include other 802.11 traffic. This application simply passes message buffers back and forth between one machine and another without any further computation. We varied the buffer size from 5 KB to 1000 KB, as shown on

the x-axis. It can be seen in this graph that the LEECH version incurs a much lower communication overhead than does MPICH/MPD. This difference increases as the size of the buffer increases. It has been previously noted that MPI has a high degree of communication overhead proportional to the message size [25]. Contributing factors include message headers, management of large data structures to handle unfulfilled function calls, and group organization. LEECH is much more lightweight with minimal message headers and simple internal management schemes.

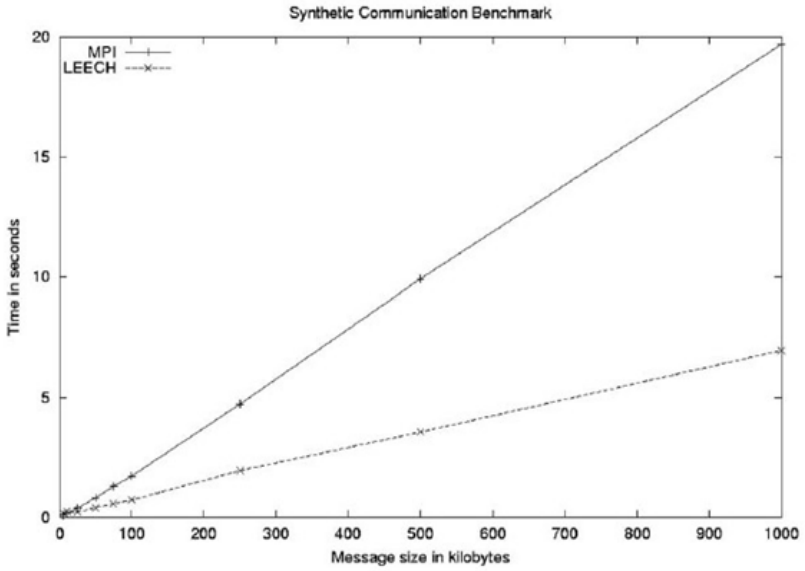


Figure 3. Average Round-Trip Latencies Measured from Execution of Synthetic Communication Benchmark Experiments were measured across two laptop

References:

[1]. T. Anderson, D. Culler, D. Patterson, and the NOW Team. A Case for NOW (Networks of Workstations), IEEE Micro=, February 1995.

[2] M. Baker, R. Buyya, and D. Laforenza. The Grid: International Efforts in Global Computing, in Proceedings of the International Conference on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet, July 31– August 6, 2000.

[3]. D. Becker, T. Sterling, D. Savarese, J. Dorband, U. Ranawak, and C. Packer. Beowulf: A Parallel Workstation for Scientific Computation, in Proceedings of the International Conference on Parallel Processing, 1995.

[4]. J. Broch, D. Maltz, D. Johnson, Y.-C. Hu, and J. Jetcheva. A Performance Comparison of Multi-Hop Wireless Ad-Hoc Network Routing Protocols, Mobile Computing and Networking , pp. 85–97, 1998.

[5]. R. Buyya, K. Branson, J. Giddy, and D. Abramson. The Virtual Laboratory: Enabling On-Demand Drug Design with the World Wide Grid, in Proceedings of the IEEE International Symposium on Cluster Computing and the Grid,May 21–24, 2002.

[6]. Cactus. From Supercomputers to PDAs: Cactus on an iPAQ, August 27, 2002. www.cactuscode.org/News/Ipaq.html.

[7]. T. Cai, P. Leach, Y. Gu, Y. Goland, and S. Albright. Simple Service Discovery Protocol, IETF Internet Draft, October 1999.

[8]. H. Casanova, G. Obertelli, F. Berman, and R. Wolski. The AppLeS Parameter Sweep Template: User-Level Middleware for the Grid, in Proceedings of Supercomputing,2000.

[9]. K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid Information Services for Distributed Resource Sharing, in Proceedings of the 10th IEEE International Symposium on High-Performance Distributed Computing (HPDC), 2001.

[10]. H. Dail, H. Casanova, and F. Berman. A Decoupled Scheduling Approach for the GrADS Program Development Environment, in Proceedings of Supercomputing, 2002.

[11]. T. DeFanti, I. Foster, M. Papka, R. Stevens, and T. Kuhfuss. Overview of the I-WAY: Wide-Area Visual Supercomputing, The International Journal of Supercomputing Applications and High Performance Computing, vol. 10, no. 2, Summer–Fall 1996.

[12]. Hooked on Lithium, The Economist, Technology Quarterly, June 22, 2002.

[13]. J. Flinn, S. Park, and M. Satyanarayanan. Balancing Performance, Energy

Conservation, and Application Quality in Pervasive Computing, in Proceedings of

ICDCS, July 2002.

[14]. G. Forman and J. Zahorjan. The Challenges of Mobile Computing, IEEE Computer, vol. 27, no. 4, April 1994.

[15]. I. Foster and C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit, International Journal of Supercomputer Applications, vol. 11, no. 2, 1997.

[16]. Gartner, Gartner Dataquest Says Worldwide PDA Market Suffers through a Dismal Year in 2002, Gartner Press Release, January 27, 2003. www3.gartner.com/ 5_about/press_releases/pr27jan2003a.jsp.

[17]. Gartner, Gartner Says Worldwide PC Shipments Experienced Third Consecutive Quarter of Positive Growth, Gartner Press Release, April 17, 2003. www3.gartner. com/5_ about/press_releases/prapr172003a.jsp.

[18]. Globus, www.globus.org.

[19]. F. Gonzalez-Castano, J. Vales-Alonso, M. Livny, E. Costa-Montenegro, and L. Anido-Rifon. Condor Grid Computing from Mobile Handheld Devices, ACM Mobile Computing and Communications Review, vol. 7, no. 1, January 2003.

[20]. A. Grimshaw, W. Wulf, J. French, A. Weaver, and P. Reynolds, Jr. Legion: The Next Logical Step toward a Nationwide Virtual Computer, University of Virginia Technical Report no. CS-94-21, 1994.

[21]. M. Haahr, R. Cunningham, and V. Cahill. Supporting CORBA Applications in a Mobile Environment, in Proceedings of the 5th International Conference on Mobile Computing and Networking, August 1999.

[22]. J. Haartsen. BLUETOOTH — the Universal Radio Interface for Ad-Hoc Wireless Connectivity, Ericsson Review, no. 3, 1998.

[23]. K. Kennedy, M. Mazina, J. Mellor-Crummey, K. Cooper, L.Torczon, F. Berman, A. Chien, H. Dail, and O. Sievert. Toward a Framework for Preparing and Executing Adaptive Grid Programs, in Proceedings of NSF Next Generation Systems Program Workshop, April 2002.

[24]. T. Kimura and H. Takemiya. Local Area Metacomputing for Multidisciplinary Problems: A Case Study for Fluid/Structure Coupled Simulation, in Proceedings of the International Conference on Supercomputing, 1998.

[25]. M. Kobler, J. Kim, and D. Lilja. Communication Overhead of MPI, PVM, and Sckt Library, University of Minnesota Tech Report HPPC-98-06, 1998.

[26]. C. Lee, C. DeMatteis, J. Stepanek, and J. Wang. Cluster Performance and the Implications for Distributed, Heterogeneous Grid Performance, in Proceedings of 9th Heterogeneous Computing Workshop, May 2000.

[27]. C. Lee, S. Matsuoka, D. Talia, A. Sussman, M. Mueller, G. Allen, and J. Saltz. A Grid Programming Primer, Technical report, Advanced Programming Models Research Group, August 2001.

[28]. M. Litzkow, M. Livny, and M.W. Mutka. Condor — A Hunter of Idle Workstations, in Proceedings of the 8th International Conference of Distributed Computing Systems, June 1988.

[29]. S. Lyer, L. Luo, R. Mayo, and P. Ranganathan. Energy-Adaptive Display System Designs for Future Mobile Environments, in Proceedings of ACM Mobisys, May 2003.

[30]. J. Markoff. From PlayStation to Supercomputer, The New York Times, May 27, 2003.

[31]. G. Miklos, A. Racz, Z. Turanyi, A. Valko, and P. Johansson. Performance Aspects of Bluetooth Scatternet Formation, in Proceedings of the 1st Annual Workshop on Mobile Ad Hoc Networking and Computing, 2000.

[32]. M. Migliardi, M. Maheswarn, B. Maniymaran, P. Card, and F. Azzedin. Mobile Interfaces to Computational, Data, and Service Grid Systems, ACM Mobile Computing and Communications Review, vol. 6, no. 4, October 2002.

[33]. MPI: A Message Passing Interface Standard, June 1995. www.mpi-forum.org.

[34]. MPICH — A Portable MPI Implementation, www-unix.mcs.anl.gov/mpi/mpich/.

[35]. MPICH-G2 homepage, www3.niu.edu/mpi/.

[36]. M. Robshaw. Security Estimates for 512-bit RSA, Technical note, RSA Laboratories, 1995.

[37]. G. Pei, M. Gerla, and X. Hong. LANMAR: Landmark Routing for Large Scale Wireless Ad Hoc Networks with Group Mobility, in Proceedings of IEEE/ACM MobiHOC, August 2000.

[38]. C. Perkins and D. Johnson. Mobility Support in IPv6, in Proceedings of the 2nd Annual International Conference on Mobile Computing and Networking, November 1996.

Authors

ShipraGoel She is a lecturer in Dr.KedarNathModi Institute of Engineering and Technology(Department of MCA)She is here since 3 years. Her major research area interest includes: Mobile Computing, Automata,Internet and Java Programming,Dataware Housing and Mining.