A Heap-based P2P Topology and Dynamic Resource Location Policy for Process Migration in Mobile Clusters

Y. Mohamadi Begum, M.A. Maluk Mohamed
Software Systems Group
M.A.M. College of Engineering
Anna University
Tiruchirappalli, India
ssg_mohamadi, ssg_maluk@mamce.org

***Abstract*— A mobile cluster experiences disruption in execution of long-running applications due to its highly dynamic nature. Process migration handles such dynamism to have seamless computing with minimal disruption. The challenge in process migration is that it should take considerably less time and techniques adopted for static networks are not suitable for mobile networks. This work is a novel effort that organizes the cluster as a heap-based super P2P structure and process state is transferred in terms of object migration between the peers. Also, while migrating processes, load balancing is dynamically done. As the mobile cluster has heterogeneous nodes with varying processing capabilities, we devise a mechanism for computing the capabilities of these nodes. Considering the capability and current load of the nodes the right destination for process migration is chosen and thus we attempt at a better location policy for the migrated process.**

Keywords- DHT; load balancing; mobile cluster; P2P networks; process migration.

I. INTRODUCTION

A mobile cluster (MC) is a Network of Workstations (NOW) or nodes that may be both stationary as well as mobile. The mobile nodes (MN) communicating over a cellular network, may leave or enter a cell any moment of time, making it difficult to run long-running applications on the MC. There are basically three issues that need to be addressed because of the mobility and the resource-poor nature of the nodes. Firstly, the mobile nodes may enter into doze mode or voluntarily disconnect from the entire network affecting the overall cluster availability and performance. Secondly, disconnections can be abrupt, where the device may enter into a region, out of coverage. However, during such disconnections, the communication link would be maintained by the Communication Subsystem (CS) of the Mobile-OS. Since the seamless communication is maintained by the CS, the out of coverage issue has no effect on the on-going computation. That is, we can simply move the computation along with the mobile node. Finally, there may be disruption in service due to the sudden failure of the node, which requires periodic checkpointing of the processes running on that node and applying migration strategy discussed in this paper. Hence, in this work, we consider only the voluntary disconnections as an appropriate issue that needs to be resolved.

This work focuses on pre-determined sign-off occurring due to such voluntary disconnections. By anticipating such disconnections, the MN has two options, namely, one that is followed in Coda file system [2], where it pro-actively downloads any data that is required so as to function independently of the network in carrying out the task assigned. Coda attempts at distributed file sharing applications and not compute-intensive applications. It assumes higher bandwidth communication with its servers. Further, it requires user's prediction on future needs for its cache management policy. The second option for the MN is to checkpoint its process state so that it transfers the same to another node for resuming execution. In our work, we choose the second option of process migration (PM) [3]. PM is associated with moving a process state from one node to another for resuming execution on the latter. In systems with only static nodes, there are a number of implementations for process migration. However, in MC those techniques adopted for static nodes are not applicable or even irrelevant owing to the mobility constraints. It adds to the complexity when process migration is to be coupled with load balancing.

The impact of mobility [1] on distributed computations is severe that it requires a totally different approach for both PM and load balancing. The migration cost is typically a function of address space size of the process and nevertheless includes the cost of locating an apt destination node. The cost incurred in locating such a node in a static network is obviously less compared to a dynamic one. In a MC, what magnifies the cost is the way the devices communicate with each other. Here the task is assigned to a mobile device only through its Base Station (BS). Hence, a copy of the program code as well as the static data is already available with the BS. Whenever a migration request comes from a MN, the BS pro actively sends this to the most eligible device in its cell based on its computational power and its current load. After receiving the process state from the MN, the BS transfers the same to the destination. The goal of load balancing is to assign to each node tasks

 ISBN: 978-1-61208-100-7

proportional to its performance. A MC whose nodes are highly heterogeneous, comprising of a combination of various resources, requires an efficient location policy to determine a suitable node as a destination for the migrating process.

Distributed scheduling related to process migration [3] decides on when to migrate, which process, and where to migrate. Some popular distributed scheduling policies like sender-initiated, receiver-initiated, random policies are not apt for resource-constrained mobile cluster. We propose a novel approach for minimizing such overheads using Peer-to-Peer (P2P) systems. Such systems share computer resources by direct exchange, rather than requiring the intermediation or support of a centralized server or authority. In our work, this property of P2P systems is exploited to implement a new location policy for dynamic process migration.

We begin by building a hierarchical structured P2P cluster of static workstations and mobile devices. The overlay network thus arrived by employing Distributed Hash Table (DHT) concept helps in linking resources (nodes) and enables easy sharing of processing among them. Traditionally Distributed Hash Table (DHT) is used only to find and share content / file / data only. In our work, we use the DHT to assist a mobile node find a destination node (to which it can off-load its current processes) when it voluntarily gets disconnected. A binary heap (max-heap) is built and a selection algorithm is written to obtain the node with maximum spare computational capacity. The contributions of this paper can be summarized as follows.

1. We propose a process migration policy for a dynamic MC built on a cellular network when it is processing a long-running, compute-intensive application.

2. While migrating processes we devise a more accurate mechanism for dynamic load-balancing considering the variations in computing power of the different mobile nodes.

3. A DHT-based approach for location policy on where to migrate is described.

4. Considering the resource-poor nature of the mobile device, we have formulated an algorithm for a node to accept a migration request or not.

5. We show how large prime numbers can be generated on a mobile cluster while striving to harness the idle time of the nodes. Also we demonstrate an efficient programming way of storing such large numbers in memory-poor mobile nodes.

This work does not spell any mechanism for PM, but a facility that reduces the time taken for PM in a very dynamic network and also increases the throughput of the system. We also restrict our work to homogeneous PM in which we migrate between nodes of same architecture. This restriction is reasonable as our concern is not with the mechanism but with the policy of when and where to migrate the process. The rest of the paper is organised as follows. Section 2 presents the related work in this area of research. In section 3 we provide the background and also highlight the rationale behind building a mobile P2P cluster. Section 4 provides system analysis and design of the mobile cluster. Section 5 accounts for the test application of large prime generation and a comparison of performance of the system with existing ones. Section 6 concludes and gives an insight into the future enhancements.

II. RELATED WORK

PM is extensively surveyed in [3, 4], which present a number of approaches. Some recent efforts on PM include [5] in which the authors describe the use of system-wide pointers and global dynamic data structures for migration. Gobelins DSM [6] moves processes or threads among cluster nodes using the distributed shared memory concept. But it again does not deal with mobile devices. Checkpointing for mobile computing systems has been discussed in [7, 8, 9]. An evaluation of different checkpointing protocols is done in [10]. One of the factors that amount to the cost of PM in a dynamic network is the time taken for deciding on the destination. All previous works on PM only focus on time taken for state transfer, but not for locating the destination. In [11], the issue of timeliness for rerouting and multicast when handoff occurs in a MC is discussed. A model for overcoming such issues in MC is presented in [12]. P2P networks [13] are self-organising structures apt for realising such clusters. Some important DHT-based P2P systems are found in [14 – 16] and are focused on fixed networks only. In [17], the authors propose a load balancing scheme for heterogeneous cluster using mobile agents. An algorithm for load balancing in heterogeneous dynamic P2P systems using the concept of virtual servers is presented in [18]. P2P networking in mobile environment is explored in JXME [19]. In [20], the mobile devices are assumed as low-performance nodes and hence only used to redirect their requests to their associated static nodes. However, in reality we cannot afford to keep mobile nodes without any useful processing. Hierarchical P2P systems are said to improve scalability. Such systems are discussed in [21, 22, 23]. Heaps based on the concept of a complete binary tree, are a good choice for implementing selection and priority based algorithms. A distributed heap-based data structure called CONE [24] has load balancing properties and is layered on Chord DHT. To the best of our knowledge, none of these works focus on a process migration facility for mobile clusters. Thus, this work is a new attempt at such a design.

III. BACKGROUND

A. *Problem Definition*

The fact that the mobile devices are becoming powerful in terms of computing cannot be overlooked. This calls for mechanisms to run high-end applications on such devices while taking care of their inherent mobility. The mobility issue can be seen in two perspectives. Firstly, the devices voluntarily disconnect themselves or move around; thus they leave or join the network / cell anytime. Secondly, disconnections can occur suddenly or the device itself may fail suddenly. Voluntary disconnection forces the termination of long-running applications. The problem is severe when such applications get terminated especially when they are nearing completion. PM eliminates this problem whereby the mobile node transfers the non-static part of the computation

 ISBN: 978-1-61208-100-7

state to the BS and the BS finds another compute node in its cell to resume the computation.

In systems with only static nodes, there are a number of implementations for PM. Nevertheless, there should be a justification between the choice of running a process from the start after disconnection and PM because the time taken for PM is not always negligible. It includes the time taken for choosing a destination node with sufficient capability while maintaining load balancing and transferring the process state to such a node. In static networks the time for negotiations between the nodes to choose a destination is generally a constant depending on the type of the network and therefore very negligible. To choose a destination node in mobile clusters, techniques like polling, random or nearest neighbor selection policies are not applicable or even irrelevant because of inherent mobility and other constraints posed by mobile nodes. This work attempts to define a dynamic resource location policy so as to choose an eligible and efficient destination node in a highly dynamic and heterogeneous network.

B. *Mobile P2P Cluster*

In designing a mobile cluster, P2P system is an attractive architectural alternative to the traditional client-server computing. It solves the problem of server being down or becoming overloaded. Therefore a number of critical, real-time, computationally high-end applications can be successfully implemented on a mobile P2P cluster. Further, P2P network is self-organizing, which is a key advantage for a dynamic network. They offer efficient search/location of nodes and load balancing facilities. These characteristics make P2P network a good choice for performing process migration.

IV. SYSTEM DESIGN

A. *Basic Model*

The mobile cluster (MC) contains a set of mobile nodes (MN) and static nodes or mobile support stations called as Base Stations (BS). Static wired network connects BSs to each other whereas a cellular network connects the MNs. In such a network, there are multiple cells and each cell has multiple channels to communicate with many mobile nodes. Also each cell is equipped with a BS that governs multiple MNs in that cell. Any MN stays in connection with at most one BS at any given time and communicates with other MNs and BSs only through the BS to which it is currently connected.

B. *Building a P2P Mobile Cluster*

We consider the mobile cluster to adopt a Super-Peer network model. In this model, the BS is categorized as super-peer. A typical distributed application is submitted to the cluster through a designated coordinator that is nothing but the BS. The BS distributes the application to the peers. The peers do not choose which processes to host. The processes are allotted to peers depending on their capability and current computational load. The peers periodically inform and update their load information to their super peer, the BS. Fig. 1 depicts BS as super-peer and all other nodes organized as a heap.

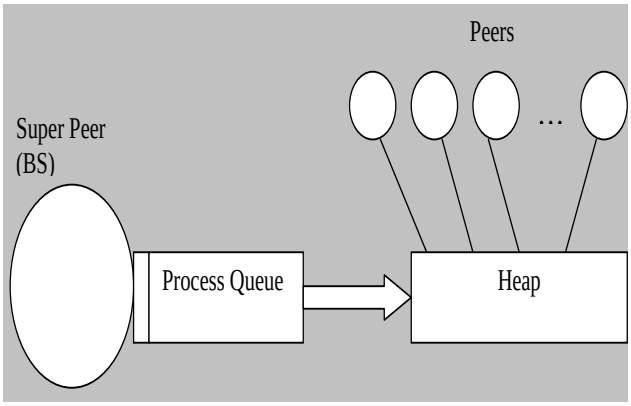

Figure 1. Super Peer and Peers

1) *DHT Design:*

When a mobile node sends a migration request to BS, the BS as a super-peer determines the destination on behalf its client (the source). In designing such structured networks, Distributed Hash tables (DHT) are employed. They are distributed data structures for building robust P2P applications. Conventional DHT maps keys to values, store key/value pairs and retrieve values using the given keys and are used in general for file storage and sharing. In contrast, in our work, every node in the mobile cluster stores a hash table and the resultant DHT performs two functions: (i) organizes cluster nodes as a max-heap and (ii) distributes processes to various nodes considering their current load and capability.

Each node in the cluster is known by a 128-bit identifier that is unique in the cluster. Each process is assigned a unique identifier that remains unchanged even when migrated. On applying consistent hash function, we allow nodes to join and leave the cluster with minimal disruption. The hash function determines the node identifier by hashing the IP address of the node and a key for each process by hashing the process identifier. The node identifiers are arranged in the form of a max-heap. A max-heap is a complete binary tree in which at every node the data stored at the node is no less than the data at either child. The heap is constructed based on the spare capacity S_i available at each node i. We show in Section 4.2.2 on how to calculate this spare capacity. Thus the root node always has maximum spare capacity. The DHT supports basically three operations, namely join, leave and migrate. The heap is reconstructed whenever a node joins/leaves or a process migrates and is done by heapify(). heapify() maintains the max-heap property that the spare capacity of parent node is always more than that of its child nodes.

The node that joins the cluster is added to the bottom of the heap, keeping up the shape of a complete binary tree. If it has more spare capacity than its parent, it is swapped with it. Then the node continues to move up until it finds a place so as to maintain the heap property. The node before leaving the cluster initiates migrate operation forwarded to the BS. Also

 ISBN: 978-1-61208-100-7

when a process migrates to a new node, that node now has less spare capacity and therefore moves down in the heap. The nodes maintain information that enables communication to their neighbor nodes. That is, except the leaf nodes, all other nodes keep an account of their child nodes.

The super peer maintains a heap manager (HM) which is responsible for keeping up-to-date information on the current load and spare capacity of every node in the cell. Every node that enters or leaves a cell causes an imbalance in the load and the spare capacity of all other nodes is bound to change and this is updated by the HM through appropriate function calls. At any point in time in a heap it is always the root node that has maximum spare capacity. This facilitates in searching a destination node among n peers in O (1) time; joining and leaving of nodes consumes O (log N) time where N refers to total number of peers.

2) Capability of Heterogeneous Nodes:

We illustrate an effective way of computing the capability of nodes in the cluster and thereby determine the potential candidate to take up the migrated process. A benchmark program is run on all the N nodes of the cluster. The execution time is recorded for every machine, say e1, e2, eN. Since the power of the node is inversely proportional to the execution time, we take the largest ei. Compute its inverse to make it our unit. We divide the inverse of every other ei by this unit. This gives the relative power of the various nodes of the cluster. For example if the execution times are say: 100 ns, 500 ns, 1 micro sec, 5 micro sec, and 10 micro sec. The inverse of 10 micro sec is 100,000. The computer that gave the result in 100 ns has an inverse of 10,000,000. On dividing 10,000,000 by 100,000 we get 100. Thus we say that the first machine is 100 times more powerful than the last machine.

We find the current load on the processor and find out the spare capacity Si of the node i by subtracting current load from 1. If the CPU utilization is k%, it means (100 - k) % is available for the task to be added. The fact that the utilization is less than 100% indicates that we could add a task to the mobile node. The added task would then use the spare CPU time, without degrading the performance of the system. For example, let the CPU utilization be 0.999 for the node which is 100 times more powerful and that of the slowest machine be 0.1 The fastest machine now has a spare capacity of (1 - 0.999) x 100 = 0.001 x 100 = 0.1. The slowest machine has a spare capacity of (1 - 0.1) x 1 = 0.9. Thus given the current load situation, the slowest machine is 9 times more powerful than the fastest machine. Accordingly the loads are assigned so that both machines complete their assigned tasks at about the same time. In [25], the authors propose Horse power utilization (HPU). Our approach differs from HPU in two ways. Firstly, in HPU the test is done once and the results are used again and again. But here we check the CPU activity register to test for CPU occupancy every time we migrate a process. Testing CPU occupancy of course delays the process migration. However, the decision would be more accurate as we do not model the performance of a very complex system like HPU, but measure it. Secondly, the HPU approach is based on the assumption that the relative power of heterogeneous systems reacts the same way under all load conditions, which is not really true. In our approach, for every migration request we query the CPU utilization and use it to calculate the relative power.

3) Status of the Mobile node:

The queue of migrating processes is held at the processor controlling the base station. We are desirous of removing the waiting process to a free or less loaded mobile node. Since the mobile nodes are resource-poor, we check for their readiness to accept a migrated process or not. As regards the checking of the status of a mobile node whether it could accept a migrating process or not, the decision could be as follows.

1. A time interval is decided for the mobile nodes to inform their current base station of their status on current load, say once every second. As long as the queue of migrating processes is small compared to a predetermined size, the reporting could be once every second, which means once every billion instructions or so.

2. When the migrating process queue becomes too big reduce this time from 1 second to 0.5 second and broadcast this change to all mobile nodes connected to the base station.

3. Continue decreasing the delay between successive reporting until the queue becomes smaller. In case the reporting activity occupies more than a predefined percentage of the processor time, say 5%, we suspend the migrating process and reactivate when the load decreases.

4. When we find that the migration queue is smaller than a predetermined size, we increase the delay again to 1 second between successive reports from the connected mobile nodes.

The above process is dynamic balancing the need for higher efficiency at the mobile nodes against the queue size of the migrating processes.

C. Process Migration

Before the MN signs off, it initiates a daemon that has two responsibilities: one, to inform BS the intention of the node to leave the cluster and second, to save the process image as an object with the process information available in task_struct (in Linux kernel). Using object serialization, the process state is transferred to the destination via the BS. The destination node i will be chosen based on its relative power (Pi) and current CPU load (Li) obtained periodically. The BS receives periodically the Li information from its peers as discussed above.

The algorithm is summarized with the various actions that take place in the following three entities:

(i) At Super-peer:

- Receives migration request from the node that is going to sign-off.
- Determines root node as destination in heap.
- Pro actively sends code and static data to the destination; Receives from MN and transfers process state and dynamic data to destination.

(ii) At mobile node (source):

- Before signing-off: Issues migration request to its BS.

- Checkpoints and saves it current state; Transfers process state to BS.
- On return to its previous cell / another: Gets assigned new process; Starts execution.

(iii) At static / mobile node (destination/root node in heap):

- Receives migration request from its BS
- Receives process state, code and data
- Resumes execution of the migrated process
- Process gets executed on this node until completion; else if the node leaves the cluster the heap gets reorganized and its child node now becomes the root node. The request is passed onto the new root node and process repeats until completion.

V. PERFORMANCE COMPARISON

A. Prime Number Generation

Generating very large primes it is a compute-intensive application. Let us assume that we assign 1 million numbers to each partition. The partition upper bounds then are 1M, 2M, 3M, ... At the beginning of this sequence the time taken by the different partitions would be appreciably different from each other. However, the time difference would decrease as the upper bound of the partition becomes much larger. An important issue is to resolve the memory constraint of nodes while dealing with such huge numbers. If we consider the memory requirement in terms of digits or bytes, it might increase linearly as the range moves away from 1. This is because the number of digits keeps increasing as the range moves away. Even here, since the digits change only over the first lowest weighted 6 digits, the higher value digits being common, a clever programming trick would store only the lowest 6 digits for every prime and store the higher value common digits in a separate place once. Then the memory would remain constant. For example, all the prime numbers between 123000000 to 123999999 are of the form 123xxxxxx, where the xxxxxx alone need to be stored. The required prime is generated by appending 123 to xxxxxx at run time. A similar approach is used in clusters to save the memory access time and can be found in [26].

B. Comparison

The two tasks that make PM time-consuming are the time to negotiate and choose a destination node and the state transfer from source to destination. Location policy in distributed scheduling determines the destination node for the migrated process, and some of these policies include polling, random and nearest neighbor algorithms. These techniques when employed for systems with static nodes have proven performance based on the system workload. However, when applied to mobile clusters these strategies incur more overheads and also sometimes not applicable at all. For example, polling involves checking for the status of a node to accept the migrated process and continues the same until a suitable node is found. Here the best case can be the first node that is approached and the worst case is the node that is finally approached. This strategy causes increased communication cost in the worst case. Further, a node which indicated its willingness to be the destination may choose to sign off later to conserve energy or because it simply chooses to move away. In this case the mobile node needs to repeat the process of polling. Also any communication from a mobile node to another is via the BS and this amounts to a huge overhead. The same issues are true in the case of random algorithms.

The third strategy of contacting the nearest neighbor is again not applicable for a mobile cluster. Here it is difficult to determine the nearest neighbor and also if we do so it will only be an estimate and not an accurate one. Moreover as mobile cluster is very dynamic, it is difficult and time-consuming to determine the right destination. In comparison with these existing approaches, our proposed model of a heap-based DHT approach takes very negligible time to fetch the destination as well as maintains load balance. This is made possible because as the nodes join and leave we organize them in the form of a heap based P2P cluster. Further, using appropriate benchmarking we estimate the power and also with the current CPU utilization rates, the heap gets reorganized. Now as in existing approaches the mobile node does not correspond with any other nodes for their availability. The root node in the heap is simply chosen as the destination node. Even if this node moves, the heap gets reorganized and the next root node is tried. Thus this model chooses the destination node with ease. In the following table, we provide a comparison of some existing systems with ours in terms of various features.

TABLE I. COMPARISON WITH EXISTING SYSTEMS

S.No.	System	Scheduling (Centralized / Distributed)	Load-balancing	Location Policy
1.	Heap	Distributed (P2P)	Continuously and Dynamic	Root node of the Heap
2.	Sprite	Centralized	Only during creation or eviction of a process	History of Idle Time Length
3.	Condor	Centralized & Priority-based	Only during creation or eviction of a process	Polling
4.	Mosix	Centralized	Continuously and Dynamic	Decentralized Load Vector with Load information on a set of random nodes
5.	MPVM	Centralized	Only during creation / eviction of a process and very high load on a node	Idle node availability

The task of transferring the process state is equally challenging and those adopted for static nodes again are not suitable for mobile clusters. The eager (all) approach is used by checkpoint/restart implementations in which the entire process state is transferred at a time to the destination. If we apply this approach in mobile cluster, the transfer is via the BS and there is a possibility that the power-constrained mobile device may go off while the transfer is going on. The eager (dirty), copy-on-reference, pre-copy and flushing strategies again are not applicable because of the dynamic nature, resource-constraints and heterogeneity prevailing in mobile clusters. Our model takes care of these issues by doing two tasks in parallel: as soon as the migration request is received by the BS, it chooses the destination and pro actively sends the static data and code; at the same time the remaining process state alone is extracted from the source and sent to the destination by means of object serialization, reducing the time taken for transferring the process state. Conclusion

The motivation to migrate a process in a mobile cluster is manifold. There are various components that contribute for the longer time taken to migrate a process. By organizing the cluster as a P2P network with nodes arranged as a heap topology, the time taken can be considerably reduced. Normally a migration request is issued to a remote node and only after negotiation, we decide to move the process. This has been avoided since the BS quickens the process migration by choosing the root node from the max-heap. Also, once the migration request is received, the BS sends the constant data and code to the destination even before the process state is dispatched from the source to the BS. This work can be extended for process migration among a finite collection of clusters and thereby a computational grid. Further we can consider migration in a heterogeneous environment because in a mobile cluster the likelihood of heterogeneity among nodes is more.

REFERENCES

[1] B.R. Badrinath, A. Acharya, and T. Imielinski, “Impact of Mobility on Distributed Computations,” ACM SIGOPS Operating Systems Review, vol. 27, no. 2, April 1993, pp. 15-20.

[2] James J. Kistler and M. Satyanarayanan, “Disconnected operation in the Coda file system,” ACM Transactions on Computer Systems, vol. 10, no. 1, Feb. 1992, pp. 3-25.

[3] D.S. Milogicic, F. Douglis, Y. Paindaveine , R. Wheeler , and S. Zhou , “Process Migration,” ACM Computing Surveys, vol. 32, no. 3, Sep. 2000, pp. 241-299.

[4] M. Singhal and N.G. Shivaratri, Advanced Concepts in Operating Systems, 2001, McGraw Hill.

[5] K. Noguchi, M. Dillencourt, and L. Bic, “Efficient Global Pointers with Spontaneous Process Migration,” Proc. 16th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP 2008), 2008, pp. 87-94.

[6] G. Vall´ee1, C. Morin, J. Berthou, Ivan D. Malen, and R. Lottiaux, “Process Migration based on Gobelins Distributed Shared Memory,” Proc. Workshop on Distributed Shared Memory (DSM'02), held in conjunction with CCGRID 2002, Germany, May 2002, pp. 325-330.

[7] R. Prakash and M. Singhal, “Low-Cost Checkpointing and Failure Recovery in Mobile Computing Systems,” IEEE Transactions on Parallel and Distributed Systems, vol. 7, no. 10, October 1996, pp. 1035-1048.

[8] B. Gupta, S. Rahimi, and Z. Liu, “A New High Performance Checkpointing Approach for Mobile Computing Systems,” IJCSNS International Journal of Computer Science and Network Security, vol. 6 no. 5B, May 2006.

[9] L. Chen, Q. Zhu, and G.Agrawal, “Supporting dynamic migration in tightly coupled grid applications”, Proc. ACM/IEEE Conference on supercomputing (SC’06), 2006, pp. 28.

[10] A. Agbaria and R. Friedman, “ Model-based performance evaluation of distributed checkpointing protocols,” Performance Evaluation, vol. 65, no. 5, 2008, pp. 345-365.

[11] H. Zheng, R. Buyya, and S. Bhattacharya, “Mobile cluster computing and Timeliness issues,” Informatica, vol. 23, 1999, pp. 5-17.

[12] M.A. Maluk Mohamed, A. Vijay Srinivas, and D. Janakiram, “Moset: An anonymous remote mobile cluster computing paradigm,” Journal of Parallel and Distributed Computing, vol. 65, 2005, pp. 1212 – 1222.

[13] E.K. Lua, J. Crowcort, M. Pias, R. Sharma, and S. Lim, “A survey and comparison of Peer-to-Peer overlay network schemes”, IEEE Communications Surveys and Tutorials, March 2004.

[14] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, “A scalable content addressable network,” Proc. ACM SIGCOMM, 2001, pp. 161–172.

[15] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H. Balakrishnan, “Chord: A scalable peer-to-peer lookup protocol for internet applications,” IEEE/ACM Transactions on Networking, vol. 11, no. 1, 2003, pp. 17–32.

[16] B.Y. Zhao, L. Huang, J. Stribling, S.C. Rhea, A.D. Joseph, and J.D. Kubiatowicz, “Tapestry: A resilient global-scale overlay for service deployment,” IEEE Journal on Selected Areas in Communications, vol. 22, no. 1, January 2004, pp. 41–53.

[17] M. Abdallah and E. Buyukkaya, “Fair Load Balancing under Skewed Popularity Patterns in Heterogeneous DHT-Based P2P Systems,” Proc. IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS), Cambridge, Massachusetts, USA, November 2007, pp. 484-490.

[18] B. Godfrey, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica, “Load Balancing in Dynamic Structured P2P Systems,” Performance Evaluation, vol. 63, no. 3, March 2006, pp. 217-240.

[19] A. Arora, C. Haywood, and K.S. Pabla, “JXTA for J2ME – Extending the Reach of Wireless with JXTA Technology,” Sun Microsystems, March 2002.

[20] S. Zoels, S. Schubert, W. Kellerer, and Z. Despotovic, “Hybrid DHT Design for mobile environments,” Proc. AP2PC Workshop at AAMAS 2006, Hakodate, Japan, May 2006.

[21] G. Erice, E.W. Biersack, K.W. Ross, P.A. Felber, and G.U. Keller, “Hierarchical Peer-to-Peer Systems,” Proc. ACM.IFIP International Conference on Parallel and Distributed Computing (Euro-Par), 2003.

[22] B. Yang and H.G. Molina, “Designing a Super-Peer Network,” Proc. International Conference on Data Engineering (ICDE), 2003.

[23] I. Rimac, S. Borst, and A. Walid, “Peer-Assisted Content Distribution Networks: Performance Gains and Server Capacity Savings,” Bell Labs Technical Journal, vol. 13, no. 3, Fall 2008, pp. 59-69.

[24] Bhagwan, R., Mahadevan, P., Varghese, and G., Geoffrey M Voelker, “CONE: A Distributed Heap-Based Approach to Resource Selection,” Technical Report CS2004-0784, UCSD, 2004.

[25] R.K. Joshi and D. Janaki Ram, “Anonymous Remote Computing: A Paradigm for Parallel Programming on Interconnected Workstations,” IEEE Trans. Software Eng., vol. 25 no. 1, 1999, pp. 75-90.

[26] S. Hwang, K. Chung, and D. Kim, “Load Balanced Parallel Prime Number Generator with Sieve of Eratosthenes on Cluster Computers,” Proc. 7th IEEE International Conference on Computer and Information Technology (CIT 2007), 2007, pp. 295-299.

 ISBN: 978-1-61208-100-7