Cluster Computing With Mobile Nodes: A Case Study

M. A. Maluk Mohamed, V. R. Devanathan and D. Janaki Ram
Distributed and Object Systems Lab, Department of CS & E
Indian Institute of Technology Madras, Chennai, India
maluk@cs.iitm.ernet.in, vrd@peacock.iitm.ernet.in, djram@lotus.iitm.ernet.in
http://lotus.iitm.ac.in

Abstract

Recent advances in hardware and software technologies have created a plethora of mobile devices with a wide range of communication, computing, and storage capabilities. This has helped in exploring the possibility of creating a world of ubiquitous mobile computing that keeps people connected to Internet applications and services at all times, regardless of their location or access device. To utilize the idle computing power of the computers, cluster computing provides an efficient solution. With the recent advent of mobile devices in personal and distributed computing, computing power has become ubiquitous. This has given rise to a new paradigm called Mobile Cluster Computing (MCC), which includes both static and mobile hosts. MCC model proposed provides mobility transparency, distribution transparency and heterogeneity transparency. Our model was successfully implemented on top of a reliable multicast protocol for distributed mobile systems. In this paper, we have given a detailed case study of a computation intensive image rendering application for MCC. The performance analysis shows a linear to super linear spen a, due to the communication overhead being amortized.

Keywords:*Cluster Computing, Reliable Multicast, Mobile Cluster Computing, Timeliness, Parallel Programming.*

1 Introduction.

The technological growth has transformed the less powerful large computing devices into highly powerful compact devices that enable, mediate, support and organize our daily activities. Inspite of this the hunt for computational power is continuing in the history of computing. As soon as a new, more powerful, computer is developed a larger problem to be solved appears on the horizon. This need for computational power, instead of leveling off, is growing day by day. In recent years many research work has been carried out with many high performance computer systems like vector computers, massively parallel processors, etc. have been built to feed the power hunters. Although such computers were developed they provided better performance only for certain classes of applications. They also failed due to high cost and a low performance/price ratio. The computers like multiprocessors could not support scalability. Distributed systems supports scalability but they could not offer a means for fast communication and ease of use.

Clusters were formed by the collection of interconnected computers working together as a single system. Clusters thus eliminated the need for super computers as they provided better performance and fault tolerance compared to the traditional mainframes or supercomputers. The availability of high-speed networks and high performance workstations has made networks of workstations an ideal system for parallel computing. Few proposals like the Con-

dor (Litzkow et al., 1988), NOW (Anderson et al., 1995) and Batrun (Tandiary et al., 1996) have been proposed earlier for improving the utility of workstation clusters which are connected using a wired network.

Parallel programming on workstation clusters is a recently proposed idea and has received a tremendous support from the researchers. Parallel programming on workstation clusters mostly follow the collection of processes model in which processes communicate via message or shared memory abstractions. This model did not suit for loosely coupled openended workstations because of heterogeneity in architecture and operating systems, load variations on machines, variations in machine availability and failure susceptibility of networks and workstations. (Joshi and Ram, 1999) and (Binu K. Johnson et al., 2001) were the recently proposed model for parallel programming on workstation systems of a wired cluster considering these issues.

Further growth and advancement in mobile wireless networking system will make the world a Mobile World with ubiquitous computing, in next few years. The convergence of mobile communications and the internet will result in various new technologies, like mobile computing. With processing power available everywhere, the mobile nodes will also enter as a contributing entity to cluster computing in addition to the static nodes, thus making the cluster computing as Mobile Cluster Computing(MCC). This could help researchers, scientists and public to interact with information and technology without the concern of where they are. Mobile Hosts(MHs) have severe constraints such as battery power, limited bandwidth, frequent disconnections and physical mobility from one cell[1] to another. These constraints may cause MH to lose messages even in an otherwise reliable error-free environment. Though the MHs are susceptible to the above constraints they could still go a long way being a part of the cluster, there are many applications like image rendering on a battlefield that could benefit out of the MHs.

To the best of our knowledge, only two proposals had come out on MCC, namely (Zheng et al., 1999) and (Basit and Chang, 2002). (Zheng et al., 1999) just define and analyze the potential application environment of MCC, and gives a generic architecture of a mobile cluster computing. On the other hand (Basit and Chang, 2002) gives a basic architecture of MCC which uses IPv6 as a primary protocol. We had proposed (M. A. Maluk Mohamed et al., 2003) a model for distributed processing on workstations of Mobile Cluster Computing (MCC), which includes both static and mobile workstation systems. The proposed novel paradigm provides better anonymity. In this paper we present a detailed case study for the proposed model of the MCC using a computation intensive image rendering application.

The rest of the paper is organized as follows. Section 2 gives a brief description about the system model considered. In Section 3, the MCC model is discussed. Section 4 discusses the developed image rendering application. Section 5 describes the performance analysis of the model and Section 6 concludes the paper with summary and future direction.

2 System Model.

Figure 1 illustrates our model of MCC (M. A. Maluk Mohamed et al., 2003). Each MH has a client process and a daemon. The client process and daemon run over a reliable multicast protocol. The client processes are used to submit tasks to the MHs (via multicast protocol) for distributed processing. The daemons are the computing entities at the MHs that execute a part of the submitted task concurrently with other daemons.

In our model, we use a notion of Cluster-Subgroups (or Subgroups). A Cluster-Subgroup refers to the characteristics of the task submitted to that Subgroup. Each host (static and mobile) based on its capabilities, joins the respective Subgroups. For example, Cluster-Subgroup LOW may refer to those tasks having memory requirements $<$ 10MB. A MODERATE Cluster-Subgroup may have tasks having memory requirements $<$ 50MB. A HIGH Cluster-Subgroup may have tasks having memory requirements $<$ 100MB. Tasks with memory requirements $>$ 100MB may be in Cluster-Subgroup VERY HIGH. A host is assigned as a Coordinator to the Cluster and it keeps track of the total number of

[1]Cell is a region under a Mobile Support Station(MSS), where MSS is capable of communicating with any mobile host under it.

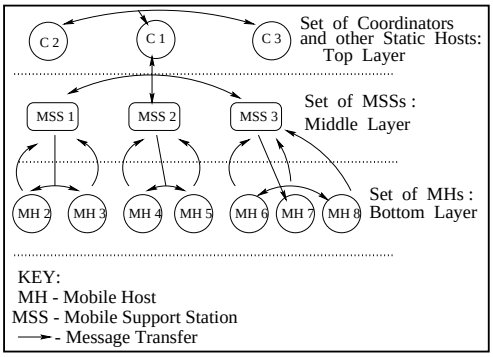

Figure 1: System Model

computing entities (called N) under each Cluster-Subgroup. Further, each computing entity has a unique membership identifier (called ID, ranging from 0 to $N-1$) associated with each group subscribed by it.

3 MCC Model Description.

Whenever a MH wants to participate in distributed processing, the daemon at the MH spawns a set of computing entities, based on its capabilities. The exact number of computing entities spawned by each daemon will depend upon the capability and the processing power of each MH (called Horse Power Factor, discussed in section 3.3).

3.1 Parallelism in the Model

The dataset which is very large is multicast to the Subgroup, based on the size of the file. Multicast provides an efficient mechanism for transferring the data to the computing entities for processing. Each computing entity independently splits the task based on its ID and N. For example, in the case of distributed image rendering application, frames having frame number 'f' such that $mod(f, N) = ID$ are rendered by the computing entity with identifier ID. When an entity completes its share, it sends the result back to the destination host.

3.2 Dynamic Join and Leave of Multicast Groups.

Initially, each computing entity sends a message $< Join, G >$ to the coordinator, for each Subgroup G the MH belongs. The coordinator upon receiving $< Join, G >$ message, increments the total number of computing entities and allocates membership identifier $N-1$ to the new entity. Further, the coordinator multicasts $< Join, G >$ to the subgroup G. Every entity subscribed to subgroup G, increments N upon receiving $< Join, G >$. Similarly, whenever an entity with identifier ID leaves a Cluster-Subgroup G, it sends $< Leave, ID, G >$ message to the coordinator. The coordinator decrements N and multicasts $< Leave, ID, G >$ to the subgroup G. Every entity subscribed to G, decrements N upon receiving $< Leave, ID, G >$. Additionally, if the identifier of the entity is greater than ID (of the leaving entity), then the entity decrements its identifier for the group G.

3.3 Dynamic Load Balancing.

As each host may have different capabilities (such as memory) and different processing power, it is essential to allocate tasks to the MHs based on their capabilities and processing power. To incorporate this, each host is allocated an integer called Horse Power Factor (HPF) (Joshi and Ram, 1999). HPF refers to the computational capability of the host. When a host has HPF 'h', then 'h' computing entities are allocated to the host. For example, if hosts A and B have h_1 and h_2 as their respective HPs, then time taken by A to compute h_1 amount of a task is approximately equal to the time taken by B to compute h_2 amount of the same task.

Dynamic load balancing (Singhal and Shivaratri, 1994) is done by maintaining two thresholds viz., Upper Threshold (UT) and Lower Threshold (LT) at the MH. These thresholds are shared by all the computing entities within a MH. This can be achieved by creating the computing entities as threads of the MH. When the load on the MH is greater than UT, a computing entity on that MH leaves the group and increments UT and LT on that MH. Both UT and LT are incremented so that all entities do not leave the groups at the same time. Similarly, when the load on the MH becomes lesser than LT, a computing entity on that MH joins the group and decrements both UT and LT. The need for two thresholds UT and LT is to avoid oscillations of frequent join and leave.

Further, the load balancing mechanism used is non-preemptive (Singhal and Shivaratri, 1994) and hence migration of already running task is not done. This is due to the additional communication overhead involved in process migration.

3.4 Timeliness Issue

Timeliness issue is an important issue especially in real-time systems. However in cluster computing systems, the system is mainly meant for computation intensive problems, like environmental modeling where the factor of time can be relaxed. But still the workstation cannot take an infinite amount of time for executing the subtask which it has accepted to execute. Incase if the workstation did not return within the stipulated time, then the subtask is submitted to some other idle workstation for getting executed or in the worst case gets executed in the client process itself. This ensures fault tolerance of the system.

3.5 Distributed Image Rendering Algorithm

The characteristics of image rendering application are as follows.

1. Very large datasets.
2. Computationally intensive.
3. Time consuming.

Due to the above characteristics, distributed processing is very essential for image rendering application.

The algorithm is explained by message transfer diagram shown in figure 2. Each client in the system stores the next frame number it expects from the daemons (or computing entities) in $nextFrame$. Also, each client has a buffer to store the out-of-order frames. From here on, both daemon and computing entity refers to the computing entity.

We use CAPS to denote the message tags that identify the type of the message. This is needed as we use a stateless approach. We denote A unicasting message M to B by,

A -$<$ M $>\rightarrow$ B.

We denote A multicasting message M to set of nodes Bi by,

A =$<$ M $>=>$ Bi.

We use C to refer Coordinators, M to refer MSS, H to refer MH, and G to refer Subgroup.

The Message Tag used for message transmission is.

- MCASTREQ - Request for multicast (from Application -> MH)

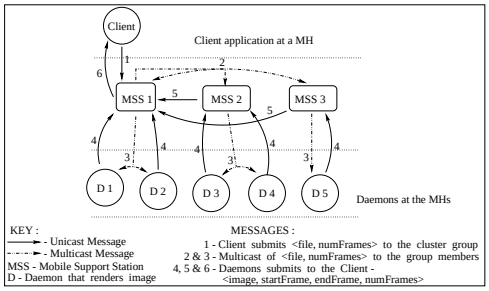

Figure 2: Distributed Image Rendering Architecture for Mobile Systems

1. **Actions taken at the client:**
   (a) Application -$<$ $file, numFrames$ $>\rightarrow$ ClientH
   (b) ClientH -$<MCASTREQ, file, numFrames>=>$ G
   (c) When ClientH receives $<$ $image, frameNumber, G$ $>$ from an entity, the clientH buffers the reply.
   (d) If ($frameNumber = nextFrame$)
   - Deliver all the frames from the next Frame

2. **Actions taken at the computing entity:**
   (a) if (entity with identifier ID receives $< file, numFrames, G >$)
   - render all the frames with frame number f such that $mod(f, N)$ = ID
   - MH-$<$ $image, f, G$ $>\rightarrow$ clientH, where $image$ is the rendered frame f

   (b) if (load on the MH $> UT$ and the entity is currently subscribed to some group) then
   - entity -$<$ $Leave, ID, G$ $>\rightarrow$coordinator and leaves the group G, for every group G subscribed by the entity.
   - The leaving entity sets ID to -1 for each group subscribed by it.
   - The leaving entity increments both UT and LT by 1.

   (c) if (load on the MH $<$ LT and the entity is currently not subscribed to any group), then

- entity -< $Join, G$ >→coordinator, for every group G the entity is capable, and joins the group G.
- The joining entity sets ID to 0 for every group joined.
- The joining entity decrements UT and LT by 1.

(d) On receiving $< Join, G >$ message and $< Leave, ID, G >$ messages have been discussed in section 3.2.

3. **Actions taken at the coordinator:**

(a) On receiving $< Join, G >$ message from the computing entity, for each Subgroup G the MH belongs
- Increments the total number of computing entities.
- Allocates membership identifier N-1 to the new entity.

(b) On receiving $< Leave, ID, G >$ message from the computing entity
- Decrements the total number of computing entities.
- C =< $Leave, ID, G$ >=>G, where G is the subgroup.

4 Implementation

The image rendering application developed renders an image obtained by CT scan. The characteristics of a CT scan image (Watt and Watt, 1992) is that they contain information from a transverse plane only. CT scanners produce 3-D stacks of parallel plane images that each consists of an array of X-ray absorption co-efficients. Due to the availability of stacks of parallel plane images, the volume data sets can be viewed as a 3-D field rather than individual planes.

The stack of planes is converted to an image by volume rendering (Watt and Watt, 1992; Lacroute and Levoy, 1994) using ray-casting. A ray-casting algorithm casts parallel rays from the viewer into the volume. At each point along the ray, the progressive attenuation due to particle fields is computed. At the same time, the light scattered in the eye direction from the light source is also computed at each point. These values are integrated along the ray and a single brightness value is computed for each ray.

Figure 3: A frame of the CT Scan image.

Table 1: Performance Analysis For Ultra SPARC 333Mhz

Number of Hosts	Time Taken(sec)	Speedup
1	421	
2	210	2.005
3	142	2.965
4	107	3.935

This rendering procedure is used in the volpack library (Volpack, 1995). The volpack library is an implementation of (Lacroute and Levoy, 1994) and it is used in this application. A frame of the final rendered image is shown in figure 3.

The client and the daemon were written in C++. The task was submitted to render 360 frames of the CT scan data (a frame for each 1° of image rotation). An online image animator was also written in C, using X11 graphics, to animate the frames as and when they arrive from the entities. Due to an online animation, the real-time effects (such as fast rendering when more hosts are involved) can be noticed.

5 Performance Analysis

The CT scan data of a human head with skull partially removed to reveal the brain is taken as the input data. The CT scan data consists of 84 slices of 128×128 samples each. The final output frame is 256×256 pixel image. The daemons were run

Table 2: Performance Analysis For Ultra SPARC 500Mhz

Number of Hosts	Time Taken(sec)	Speedup
1	292	
2	146	2.000
3	99	2.949
4	75	3.893
8	41	7.122

on SPARC 333Mhz and SPARC 500MHz machines. The daemons use (M. A. Maluk Mohamed et al., 2002) simulator for multicasting to other MHs. Further, as all the daemons where run is similar machines, each daemon had a HPF of 1. So, only one computing entity was spawned at each host.

Table 1 shows the results when daemons were executed on SPARC 333MHz. Super-linear speedup is noticed when number of hosts is 2. This is because, during the entire computation of 360 frames, the full volume data (which is reasonably large - 1.37 MB) needs to be in memory. During this time, page faults occur and the computation time increases. The context switch time is very negligible as the experiment was conducted when the load on the machines were only due to the daemons. When the task is split to two hosts, computation time is only for half the task. Hence, the data needs to be in the memory only for a much lesser time. As the memory residence time reduces, the page fault overhead also decreases. Reduction in these overheads (such as page faults and context switches, if any) and by overlapping computation and communication, the communication delay seems to be nullified. Thus, super linear speedup is achieved. But, as the number of hosts increases, the communication delay seems to dominate. This is partly because, the simulator runs on a single host and is not distributed. So, a multicast is simulated by multiple unicasts. If multicast is done physically, then the communication overheads can be reduced further.

Table 2 shows the results when SPARC 500 MHz machines were used to execute daemons. Results show a similar trend as above.

Figure 4 shows the effect of load on total execution time for rendering 360 frames. The figure shows

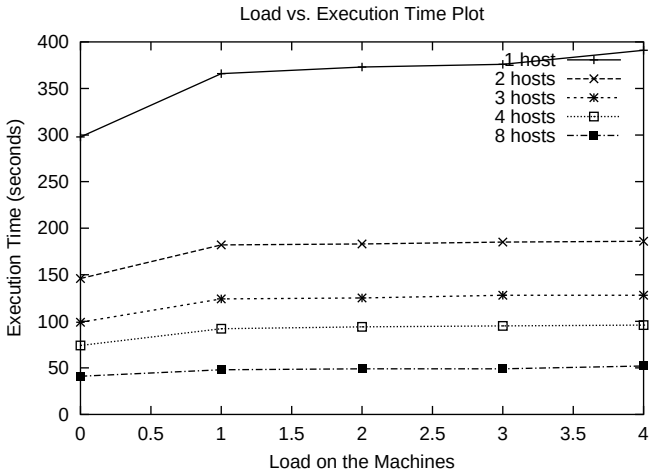

Figure 4: Effect of Load on Execution Time

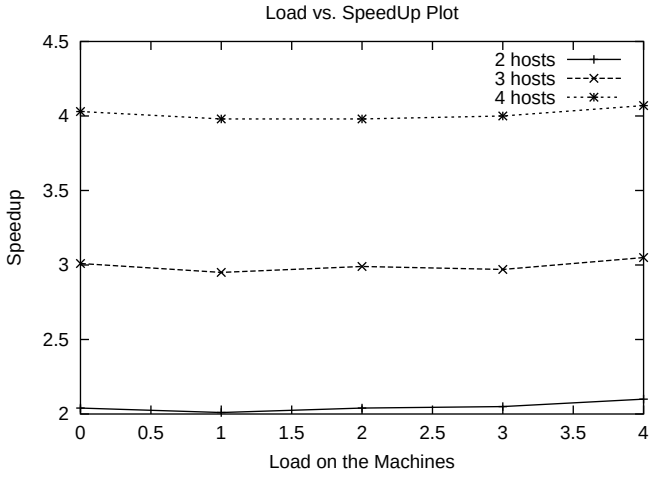

Figure 5: Effect of Load on Speedup

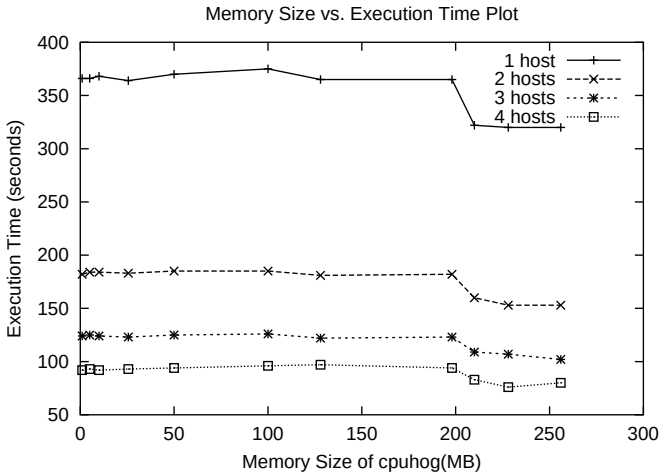

Figure 6: Effect of Process Memory Size on Execution Time

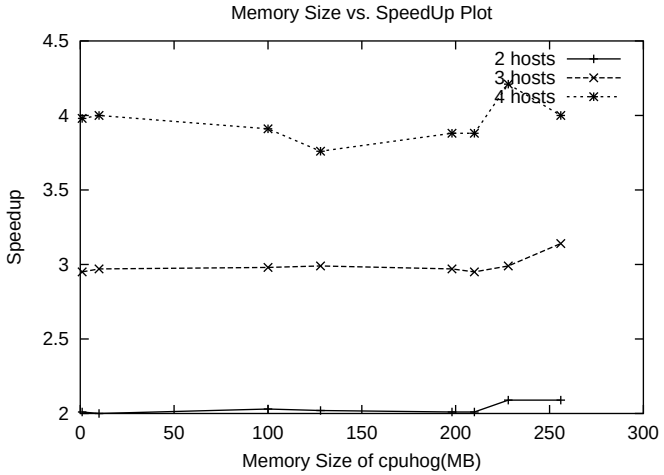

Figure 7: Effect of Process Memory Size on Speedup

the execution time in the presence of multiple half load processes in the processor run-queue. The half load on the processor is achieved using a program called cpuhog (Libenzi, 2001). As the cpuhog loads the processor only half the time, it creates a half loaded environment. The figure 4 shows that the execution time increases as the number of half loaded processes increase. This is due to the increased context switch time, due to an increase in the number of processes.

Figure 5 shows that speedup is almost constant irrespective of the load. If the number of hosts involved in computation increases, then speedup seems to increase very marginally with the load on the host. The effects of load on the execution time and the speedup where taken by using cpuhog with memory of 1 MB.

Apart from loading the processor, cpuhog also hogs the memory resource. The image rendering application needs 10MB of memory space and if cpuhog occupies more memory, then a lesser amount of memory is left for the image rendering application. By varying the memory used by the cpuhog, the effects on execution time and speedup of the image rendering application can be seen in figures 6 and 7 respectively. These figures show an *interesting behavior* explained below.

Figure 6 shows that as the memory of cpuhog increases from 1MB to 200MB, the execution time for the image rendering application is almost constant with only a very marginal increase. This is because as the memory occupied by cpuhog increases till a threshold, there are not much page faults. The reason for this behavior is that, due to the availability of large memory space (256MB), most of the pages are free. But, when the memory occupied by cpuhog increases above a threshold (200MB), cpuhog encounters frequent page faults. The main reason is that as cpuhog occupies a large number of pages, the oldest page in memory is very likely to be the page belonging to cpuhog. Hence, the oldest page of cpuhog in memory is swapped out and a new page for cpuhog is paged in. As a result, cpuhog sleeps for more time waiting for the required pages to be brought into memory. During the time when cpuhog sleeps, the image rendering application runs. Hence, the total execution time of the application drops sharply as the application gets more time to run.

Figure 7 shows the effect of memory used by cpuhog, on the speedup of the image rendering application. The speedup is almost constant as the memory of cpuhog increases till 200MB. Upon further increase in the memory usage, the speedup shows a marginal increase. Moreover, when the application is executed on 4 hosts, super-linear speedup of 4.21 is noticed.

Thus, the effect of load on the processor and the memory usage by cpuhog has only a marginal effect on the execution time and the speedup of the image rendering application.

6 Conclusions

A ideal model for Mobile Cluster Computing(MCC) has been proposed for distributed processing over mobile systems. The image rendering application was developed to prove the necessity of a reliable multicast protocol for distributed processing involving mobile systems. Reliable multicast provides an efficient mechanism for transferring the task (involving large datasets) among the computing entities. The main use of multicast is in reducing the communication delay, which otherwise would increase with the increase in the number of participating entities. Due to multicast, only one transmission of the task is needed to transfer the task to all the entities. This is also the reason for super-linear speedup. When multicast is used, conflicts may occur if two or more hosts leave the group at the same time. Totally Ordered Message Delivery (TOMD) ensures that such conflicts do not arise. The MCC proposed does not suite for all applications, it is restricted to applications which are similar to image rendering problems.

Possible extension could be to generalize the MCC model, such that it could be used for any computation intensive application.

References

T.E. Anderson, D.E. Culler, D.A. Patterson, and the NOW Team. 1995. A Case for Networks of Workstations(NOW). *IEEE Micro.*, 15(1):54–64, Feb.

Abdul Basit and Chin-Chih Chang. 2002. Mobile Cluster Computing using IPv6. *In Linux 2002 Symposium, Ottawa, Canada*, June.

Binu K. Johnson, R. Karthikeyan, and D. Janaki Ram. 2001. DP: A Paradigm for Anonymous Remote Computation and Communication for Cluster Computing. *IEEE Transactions on Parallel and Distributed Systems*, 12(10):1052–1065, Oct.

Rushikesh K. Joshi and D. Janaki Ram. 1999. Anonymous Remote Computing: A Paradigm for Parallel Programming on Interconnected Workstations. *IEEE Transactions on Software Engineering*, 25(1):75–90, Jan.

Philippe Lacroute and Marc Levoy. 1994. Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation. *Computer Graphics*, 28(Annual Conference Series):451–458.

Davide Libenzi. 2001. CPUHOG - A kernel scheduler latency tester, Free Software Foundation, Inc., Boston, MA, USA.

M. Litzkow, Miron Livny, and Matt W. Mutka. 1988. Condor - A Hunter of Idle Workstations. In *In Proceedings of the 8th International Conference on Distributed Computer Systems*, pages 104–111, June.

M. A. Maluk Mohamed, V. R. Devanathan, and D. Janaki Ram. 2002. EOMP: An Exactly-Once Multicast Protocol for Distributed Mobile Systems. Technical Report IITM-CSE-DOS-2002-13, Dept. Of C.S.E.,IIT Madras, Chennai, India, Nov.

M. A. Maluk Mohamed, V. R. Devanathan, and D.Janaki Ram. 2003. A Model for Mobile Cluster Computing: Design and Evaluation. In *To appear in the proceedings of the International Conference on Computer Science and its Applications*, San Diego, California.

Mukesh Singhal and Niranjan. G. Shivaratri. 1994. *Advanced Concepts in Operating Systems*. McGraw-Hill Inc.

F. Tandiary, S.C. Kothari, A.Dixit, and W. Anderson. 1996. Batrun: Utilizing Idle Workstations for Large-Scale Computing. *IEEE Parallel and Distributed Technology*, pages 41–49, Summer.

Volpack. 1995. Volpack - A Volume Rendering Library.

Alan Watt and Mark Watt. 1992. *Advanced Animation and Rendering Techniques: Theory and Practise*. Addison-Wesley Publishing Company.

Haihong Zheng, Rajkumar Buyya, and Sourav Bhattacharya. 1999. Mobile Cluster Computing and Timeliness Issues. *Informatica: An International Journal of Computing and Informatics*, 23(1).