Enhanced Communal Global, Local Memory Management for Effective Performance of Cluster Computing

P. Sammulal[†] and Dr.A. Vinaya Babu[††],

JNT University, Kakinada, India JNT University, Hyderabad, India

Summary
Memory management becomes a prerequisite when handling applications that require immense volume of data in Cluster Computing. For example when executing data pertaining to satellite images for remote sensing or defense purposes, scientific or engineering applications. Here even if the other factors perform to the maximum possible levels and if memory management is not properly handled the performance will have a proportional degradation. Hence it is critical to have a fine memory management technique deployed to handle the stated scenarios. To overwhelm the stated problem we have extended our previous work with a new technique that manages the data in Global Memory and Local Memory and enhances the performance of communicating across clusters for data access. The issue of the Global Memory and Local Memory Management is solved with the approach discussed in this paper. Experimental results show performance improvement to considerable levels with the implementation of the concept, specifically when the cost of data access from other clusters is higher and is proportionate to the amount of data.

Keywords:
High Performance Cluster Computing, Job Scheduling, Global Memory Management, Local Memory Management

1. Introduction

This paper is an extension work of our previous work [10]. The first inspiration for cluster computing was developed in the 1960s by IBM as an alternative of linking large mainframes to provide a more cost effective form of commercial parallelism [1]. However, cluster computing did not gain momentum until the convergence of three important trends in the 1980s: high-performance microprocessors, high-speed networks, and standard tools for high performance distributed computing. A possible fourth trend is the increasing need of computing power for computational science. The recent advances in these technologies and their availability as cheap and commodity components are making clusters or networks of computers such as Personal Computers (PCs), workstations, and Symmetric Multiple-Processors (SMPs) an appealing solution for cost-effective parallel computing.

Cluster computing can be described as a fusion of the fields of parallel, high-performance, distributed, and high availability computing. It has become a popular topic of research among the academic and industrial communities, including system designers, network developers, algorithm developers, as well as faculty and graduate researchers. The recent developments in high-speed networking, middleware and resource management technologies have pushed clusters into the mainstream as general purpose computing system. This is clearly evident from the use of clusters as a computing platform for solving problems in number of disciplines.

In some scientific application areas such as high energy physics, bioinformatics, and remote sensing, we encounter huge amounts of data. People expect the size of data to be terabyte or even petabyte scale in some applications [2]. Managing such huge amounts of data in a centralized manner is almost impossible due to extensively increased data access time. To illustrate the scenario where a scientific application is executed in a cluster computing environment the data requirement of the application would be enormous, the required data may be scattered across several clusters. In this case, streamlining data access through the usage of the proposed memory management technique will improve the performance of the entire operation.

In Cluster Computing Environment the data latency time has significant impact on the performance when the data is accessed across clusters. Memory management becomes a prerequisite when handling applications that require immense volume of data for e.g. satellite images used for remote sensing, defense purposes and scientific applications. Here even if the other factors perform to the maximum possible levels and if memory management is not properly handled the performance will have a proportional degradation. Hence it is critical to have a fine memory management technique deployed to handle the stated scenarios.

Scheduling is a challenging task in this context. The data-intensive nature of individual jobs means it can be important to take data location into account when determining job placement. Despite the other factors which contribute performance in a cluster computing environment, optimizing memory management can improve, the overall performance of the same. To address this problem, we have defined a combined memory management technique. The proposed technique focuses

Manuscript received June 5, 2008.
Manuscript revised June 20, 2008.

on optimizing memory usage, assuming the other factors which contribute to performance are performing to the optimum level.

The rest of the paper is organized as follows. Section 2 presents some of the existing works in job scheduling and memory management. Section 3 describes the previous combined memory management technique. Section 4 discusses Proposed Global and Local Memory Management. Section 5 discusses the Experimental setup and Results. Section 6 concludes the paper.

2. Related Work

Ann Chervenak et al. [3] review the principles that they are following in developing a design for data grid architecture. Then, they describe two basic services that they believe are fundamental to the design of a data grid, namely, storage systems and metadata management.

William H. Bell et al. [4] find the design of the simulator OptorSim and Various replication algorithms. After setting the simulation configuration they dedicated to a description of simulation results.

Kavitha Ranganathan and Ian Foster [5] describe a simulation framework that they have developed to enable comparative studies of alternative dynamic replication strategies. They present preliminary results obtained with this simulator, in which they evaluate the performance of five different replication strategies for three different kinds of access patterns.

Kavitha Ranganathan and Ian Foster [6] develop a family of job scheduling and data movement (replication) algorithms and use simulation studies to evaluate various combinations and they describe a scheduling framework that addresses the problems.

Houda Lamehamedi et al. [7] introduce a set of replication management services and protocols that offer high data availability, low bandwidth consumption, increased fault tolerance, and improved scalability of the overall system and their results prove that replication improves the performance of the data access on Data Grids, and that the gain increases with the size of the datasets used.

Sang-Min Park et al. [8] evaluate BHR strategy by implementing it in an OptorSim, a data grid simulator initially developed by European Data Grid Projects and their simulation results show that BHR strategy can outperform other optimization techniques in terms of data access time when hierarchy of bandwidth appears in Internet.

D. G. Cameron et al. [9] discussed an economy-based strategy as well as more traditional methods, with the economic models showing advantages for heavily loaded grids.

Khalil Amiri, David Petrou, Gregory R. Ganger, Garth A. Gibson [11] presents ABACUS, a run-time system that monitors and dynamically changes function placement for applications that manipulate large data sets and they evaluate how well the ABACUS prototype adapts to run-time system behavior, including both long-term variation (e.g., filter selectivity) and short-term variation (e.g.,multi-phase applications and inter-application resource contention).

Christine Morin [12] gives the design of Gobelins,a cluster operating system, aiming at providing these two properties to parallel applications based on the shared memory programming model and their experimentations are carried out on a cluster of dual-processor PC interconnected by a SCI high bandwidth network.

Michael R. Hines, Mark Lewandowski and Kartik Gopalan[13] address the problem of harnessing the distributed memory resources in a cluster to support memory-intensive high-performance applications and they presented the architectural design and implementation aspects of Anemone - an Adaptive NEtwork MemOry Engine – which is designed to provide transparent, fault-tolerant, low-latency access distributed memory resources

Michael R. Hines, Mark Lewandowski, Jian Wang, and Kartik Gopalan [14] discussed the benefits and tradeoffs in pooling together the collective memory resources of nodes across a high-speed LAN based cluster and they present the design, implementation and evaluation of Anemone – an Adaptive Network Memory Engine – that virtualizes the collective unused memory of multiple machines across a gigabit Ethernet LAN, without requiring any modifications to the large memory applications and also their results with Anemone prototype show that unmodified single-process applications execute 2 to 3 times faster and multiple concurrent processes execute 6 to 7.7 times faster, when compared to disk based paging.

Renaud Lottiaux and Christine Morin[15] introduce the concept of container at the lowest operating system level to build a COMA-like memory management subsystem and they have presented the concept of container and described how it can be used for global distributed memory management in Gobelins operating system targeted for clusters.

3. Combined Memory Management Technique

3.1 Global and Local Memory

Our memory management technique comprises global memory and local memory. Global memory (Mg) is common for all the clusters. Where as local memory is specific to the nodes of every individual cluster.

The global memory (Mg) constitutes a persistent storage and temporary storage. The data which are frequently accessed is stored in the persistent part and the less frequently accessed are stored in the temporary part. Intuitively the bandwidth between the global memory and the clusters will be significantly higher than the bandwidth across clusters. The local memory associated with every individual node of cluster hosts the data pertaining to the task assigned for that node. Simply the local memory consists of data that are required for the task deputed for the corresponding node.

3.2 The Scheduler and Memory Management

When a user makes a request, he specifies the required resources, the estimated execution time and the deadline. The request is forwarded to the scheduler. The scheduler consists of a resource management system which maintains details regarding the resource availability, resource under utilization. This information is updated periodically by the resource management system. Moreover the updations also happen after the completion of running requests.

The scheduler after the reception of a new request makes an analysis to identify a particular cluster to which the request can be forwarded. The scheduler primarily takes in to consideration the load of the processors of the nodes of the concerned clusters before the task is assigned. But this process of designating clusters for processing tasks would not yield optimum performance because bandwidth is also a major factor in determining the performance levels. So to overwhelm this problem we have proposed a new algorithm using global memory and local memory.

The conventional scheduling algorithm blindly fixes a particular cluster taking into account the availability of data the as the sole criterion. This method of designating a particular cluster for a request would lead to performance degradation. To illustrate the above scenario let us consider a particular request requires certain the cluster that is identified for the given request is based on the presence of major portion of required data and the cost for accessing remaining data is not considered and if it is significantly higher, then it has to be treated in a separately. At the same time, if the task is designated to a cluster irrespective of the percentage of data present in that cluster and considering the cost of accessing the remaining data from the rest of the clusters through global memory the performance can be optimized further. We have proposed a new algorithm in 3.3 that also gives the needed importance to the cost of accessing data

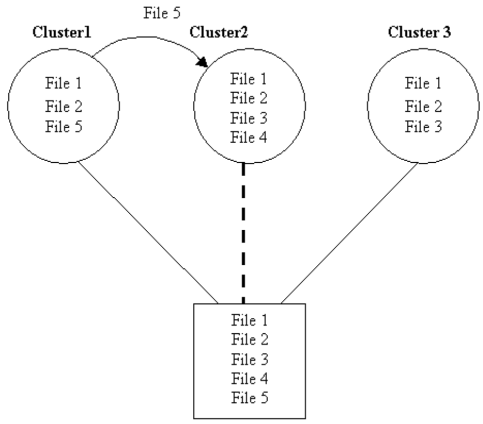

Figure 1: Example for General approach for selection of Cluster

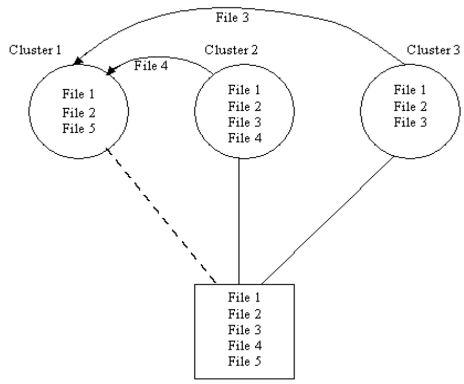

Figure 2: Example for Proposed approach for selection of Cluster

3.3 Scheduling Algorithm

Assumptions

N_C → Total Number of Clusters

CJ_i → The Cluster handling the current job.

S_F → Set of files required for the current job

SF_{WC} → Set of files available in CJ_i needed for current job

SF_{Mg} → Set of files to be transferred from

SS_C through M_g.

SN_{WC} → Set of nodes having SF_{WC} in the cluster CJ_i

SS_C → Set of clusters having SF_{Mg}

For Files within a Cluster:

for each files in SF_{WC}
  for each node in SN_{WC}
    $t = Calculate$ time
  end
  $t_{min} = \min(t)$
  Update SQN_{WC}
end

The total time needed to transfer the files between the nodes within the cluster is calculated as follows.

$$T_{WC} = \sum_{i=0}^{\text{size of }(SF_{WC})} t_{min}$$

For Files between Clusters:

for each files in SF_{Mg}
  for each cluster in SS_C
    $t = Calculate$ time to transfer file from SS_{Ci} *through* M_g
  end
    $t_{min} = \min(t)$
    Update S_{qc}
end

The total time needed to transfer the files between the clusters through the global memory Mg is calculated as follows.

$$T_{BC} = \sum_{i=0}^{\text{size of }(Sqc)} t_{min}$$

The total time needed to transfer the files required by the job is calculated as follows.

$$T = T_{WC} + T_{BC}$$

Repeat the above steps for all the clusters

$$S_T = (T_0, T_1, T_2 .... T_{NC})$$

$$T_Q = \min(S_T)$$

The cluster with minimum time is chosen to allot the job. At regular intervals the data access patterns in Global Memory is analyzed. If the data stored in temporary portion has been accessed more frequently then it is shifted to the permanent storage part. Similarly the data present in the permanent storage part is also deleted to pave the way for new storage if it is not frequently referred.

4. Global and Local Memory Management Technique

4.1 Memory management of nodes within a cluster

Files are normally transferred between both nodes and clusters when it is not found in a particular node. In between nodes, files are transferred directly without the need of global memory within the cluster. But in between the clusters, all the files transferred through the temporary memory in the global memory. Assume that the file access rate of each file is maintained in a vector V_{FAR} by every node in a cluster.

If the used memory of a node exceeds a predefined threshold value, the files which are not frequently accessed have to be removed from that node. For this purpose we have proposed the following algorithm.

4.2 Algorithm:

For each node in a cluster perform the following steps:

(1) Sort the vector V_{FAR} in ascending order.

(2) Retrieve the V_{FAR} of all the remaining nodes . within that cluster.

(3) Find the non replicated files in the node and remove that files from the vector.

for each value of V_{FAR}
remove the file from the node with in the cluster

  S_{MN} = size of all the remaining files in that node
  if $(S_{MN} < (\text{Thresh} - \varphi))$
    break
  end
end

Perform the above algorithm for the nodes of all the clusters in the Cluster Computing Environment.

4.3 Memory Management of Global Memory

The global memory consists of both permanent memory and temporary memory. All the files transferred between the clusters are transferred through the temporary memory in the global memory. Assume that the file access rate of all the files in the temporary memory and permanent memory are maintained in a vector. If a file in temporary memory is more frequently accessed, it must be transferred from temporary memory to primary memory. After transferring the files from temporary memory to permanent memory, the files must be removed from temporary memory. For this purpose we have proposed the following algorithm.

4.3.1 Temporary Memory Management:

Assumptions:

FA_R → Access rate of all the files in the temporary memory.

SFA_R → FA_R sorted in descending order.

P_M → Permanent Memory

T_M → Temporary Memory

for each value of SFA_R
  Move the file from T_M to P_M
    S_{TM} = Size of all the files in the temporary memory
      $If\ (S_{TM} < (Thresh - \varphi))$
        break
      end
end

4.3.2 Permanent Memory Management:

If the used memory of the permanent memory exceeds a predefined threshold value, the files which are not frequently accessed have to be removed from permanent memory. For this purpose we have proposed the following algorithm.

Assumptions:

FA_R → Access rate of all the files in the primary memory.

P_M → Permanent Memory

SFA_R → FA_R sorted in ascending order.

for each value of SFA_R
  Re*move* file from permanent memory
    S_{PM} = memory of all the remaining files in permanent memor
      $If\ (S_{PM} < (Thresh - \varphi))$
        break
      end
end

5. Results

We developed the algorithm in java. The performance of the algorithm was tested with few clusters. We had replication of files in the cluster also.

We stored the cluster and file information in the tables of a database and the designs are as follows

Table1: ***ClusterId:*** Table to store the Cluster Information

Filename	This list of files stored in the clusters
ClusterId	Id of the cluster
Node	Node id of the particular cluster

Table2: BetweenCluster*:* Table to store information about transferring files from one cluster to another cluster

Filename	This list of files stored in the clusters
From	Cluster id from which files are transferred
To	Cluster id to which files are transferred
Time	Time taken for transferring file from one cluster to another

Table3: WithinCluster: Table to store information about transferring files from one node to another node within a cluster

Filename	This list of files stored in the clusters
From	Node id from which files are transferred
To	Node id to which files are transferred
Time	Time taken for transferring file from one node to another
ClusterId	Cluster id of the particular node

Some result data are given in Table 4.

Files → File numbers
Size → Size of the file
From cluster id → Cluster id from which files are transferred
To cluster id → Cluster id to which files are transferred
Time → Time taken for transferring file from one node to another

Table 4: Result Data

Files	Size in kb	From Clusetr Id	To Cluster ID (Qualified)	Time in MS
1	20	3	1	29
2	23	2	1	28
3	28	2	1	38
4	31	3	1	51
5	38	2	1	49
6	45	2	1	54
7	51	3	1	62
8	55	3	1	60
9	57	2	1	71
10	67	2	1	73
1	187	1	2	220
2	204	3	2	249
3	221	3	2	300
4	276	1	2	387
5	285	1	2	338
6	347	1	2	460
7	389	3	2	465
8	401	3	2	473
9	481	3	2	512
10	480	1	2	623
11	525	1	2	607
12	528	1	2	612
13	600	1	2	698

Chart 1, 2, 3 depicts more results obtained from our experimental setup.

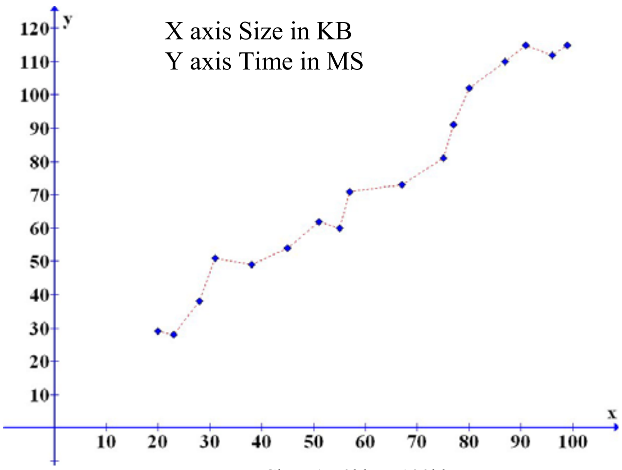

Chart 1 - 0kb to 100kb

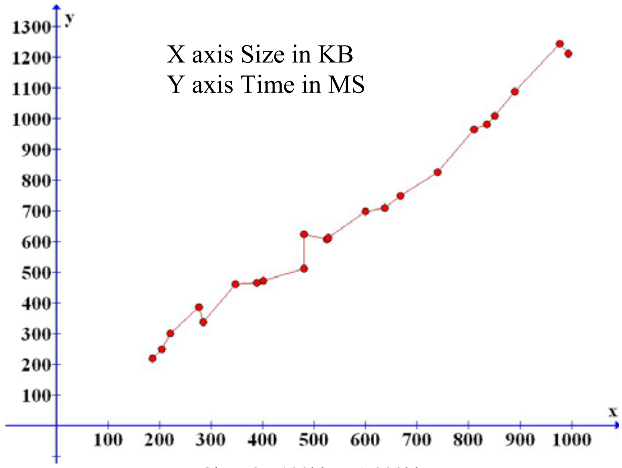

Chart 2 - 100kb to 1,000kb

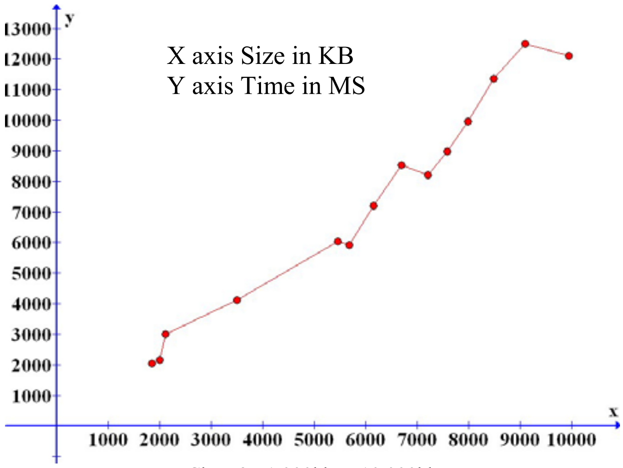

Chart 3 - 1,000kb to 10,000kb

6. Conclusion

The proposed technique for data access across clusters shows substantial improvement reducing execution time. Providing due consideration to data access latency besides computational capability proves worthwhile. The proposed concept uses a combination of Local Memory and Global Memory management scheme The Local Memory takes care of moderating communication across nodes within the cluster. In Global Memory Management, The file access rate of all the files in the temporary memory and permanent memory are considered. If a file in temporary memory is more frequently accessed, it is transferred from temporary memory to primary memory. After transferring the files from temporary memory to permanent memory, the files must be removed from temporary memory. In Local Memory Management, if the used memory of a node exceeds a predefined threshold value, the files which are not frequently accessed is removed from that node. The experimental results showed the proposed approach seems to give effective and better performance

References

[1] R. Buyya (ed.), High Performance Cluster Computing: Architectures and Systems, vol. 1, Prentice Hall, 1999.

[2] Wolfgang Hosehek, Francisco Javier Jaen-Martinez, Asad Samar, Heinz Stockinger, and Kurt Stockinger. "Data Management in an International Data Grid Project," Proceedings of First IEEE/ACM International Workshop on Grid Computing (Grid'2000), Vol. 1971, pages 77-90 Bangalore, India, December 2000.

[3] A.Chervenak, I. Foster, C, Kesselman, C. Salibury, and S. Tuecke, "The Data Grid: Towards an Architecture for Distributed Management and Analysis of Large Scientific Datasets", Journal of Network and Computer Applications, vol.23, Pages 187-200,2000.

[4] W. H. Bell, D.G. Cameron, L. Capozza, P. Millar, K. Stockinger, and F. Zini, " Simulation of Dymanic Grid Replication Strategies in OptorSim", Proceedings of the Third ACM/IEEE International Workshop on Grid Computing (Grid2002), Baltimore, USA, vol. 2536 of Lecture notes in Computer Science, Pages 46-57, November 2002.

[5] I. Foster, and K. Ranganathan, " Identifying Dynamic Replication Strategies for High Performance Data Grids", Proceedings of 3rd IEEE/ACM International Workshop on Grid Computing, vol. 2242 of Lecturer Notes on Computer Science, Pages 75-86, Denver, USA, November 2002.

[6] I. Foster, and K. Ranganathan, " Decoupling Computation and Data Scheduling in Distributed Data-intensive Applications", Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing (HPDC-11), IEEE CS Press, Pages 352-368, Edinburgh, U.K., July 2002.

[7] E. Deelman, H. Lamehaedi, B. Szymanski, and S. Zujun, " Data Replication Strategies in Grid Environments", Proceedings of 5th International Conference on Algorithms and Architecture for Parallel Processing (ICA3PP'2002), IEEE Computer Science Press, Pages 378-383, Bejing, China, October 2002.

[8] Sang-Min Park, Jae-Hoon Kim, Young-Bae Go, and Won-Sik Yoon, " Dynamic Grid Replication Strategy based on Internet ?Hierarchy", Lecturer Note in Computer Science, International Workshop on Grid and Cooperative Computing, vol. 1001, pp.1324-1331, Dec.2003.

[9] DG Cameron, AP Millar, and C. Nicholson, "OptorSim: a Simulation Tool for Scheduling and Replica Optimization in Data Grids", Proceedingsof Computing in High Energy Physics, CHEP 2004, Interlaken, Switzerland, September.

[10] P. Sammulal and A. Vinaya Babu , " Efficient and Collective Global, Local Memory Management For High Performance Cluster Computing", International Journal Of Computer Science and Network Security, Vol. 8 No. 4 pp. 81-84, 2008.

[11] Khalil Amiri, David Petrou, Gregory R. Ganger, Garth A. Gibson," Dynamic Function Placement for Data-intensive Cluster Computing"on USENIX Annual Technical Conference, San Diego, CA, June 2000.

[12] Christine Morin," Global and Integrated Processor, Memory and Disk Management in a Cluster of SMP's" in IRISA/INRIA Campus universitaire de Beaulieu, 35042 Rennes cedex (FRANCE) 1999.

[13] Michael R. Hines, Mark Lewandowski and Kartik Gopalan," Anemone: Adaptive Network Memory Engine" in proceedings of the twentieth ACM symposium on Operating systems principles Brighton, United Kingdom Year of Publication: 2005.

[14] Michael R. Hines, Mark Lewandowski, Jian Wang, and Kartik Gopalan," Implementation Experiences in Transparently Harnessing Cluster-Wide Memory" In: Proc. of the International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS 2005

[15] Renaud Lottiaux and Christine Morin," A Cluster Operating System Based on Software COMA Memory Management"in Proc. of the 1st Workshop on Software Distributed Shared Memory (WSDSM'99.

P. Sammulal is Pursuing his PhD (Computer Science and Engineering) from Osmania University, Hyderabad, India. He has received the B.E degree from Osmania University in 2000 and MTech degree from JNT University in Computer science and engg., in 2002. He is in teaching and research since 2002. He has published 4 papers in International conferences/journals. His research is focused on memory management in cluster computing, face recognition in image processing He is working as an Assistant Professor in JNT University, Kakinada, INDIA.

Prof. Dr. A. VinayaBabu received PhD (Computer Science) from JNT University, Hyderabad, India. He has received BE and ME from Osmania University in Electronics and Communication Engineering, MTech from JNT University in Computer Science and Engineering. He is a professor in CSE and Director of School of Continuing and Distance Education in JNT University, Hyderabad, India. He is a life member of CSI, ISTE member of FIE, IEEE, and IETE. He has published about 35 research papers in International/National journals, International/National Conferences and Refereed International Conferences. His current research interests are distributed/parallel computing, Cluster computing, Grid computing, Network security.