International Journal of Computer Applications (0975 – 8887)
Volume 56– No.8, October 2012

Performance Improvement in a Multi Cluster using a Modified Scheduling and Global Memory Management with a Novel Load Balancing Mechanism

P. Sammulal, PhD.
Assistant Professor, Department of CSE,
JNTUH College of Engineering,
Jagtial, Andhra Pradesh, INDIA

A. Vinaya Babu, PhD.
Professor in CSE & Principal
JNT University Hyderabad, Hyderabad,
Andhra Pradesh, INDIA.

ABSTRACT
In Cluster Computing Environment the data latency time has significant impact on the performance when the data is accessed across clusters. In this case, streamlining data access through the usage of the memory management technique with a proper scheduling mechanism will improve the performance of the entire operation. Memory management becomes a prerequisite criterion while handling applications that require large volume of data in various scientific applications. If memory management is not properly handled the performance will have a proportional degradation, even if the other factors perform to the maximum possible levels. Hence it is critical to have a fine memory management technique. The existing scheduling algorithms consider only data availability as the sole criterion in allotting an incoming job to a node in a cluster. But this process would not yield optimum performance because bandwidth is also a major factor in determining the performance level. So to overcome this problem a new scheduling algorithm is what required. Load balancing is a key technique used to improve the performance of cluster application by utilizing machines to the full extent without any idle or underutilized resources. We have tested our CWA load balancing algorithm for face recognition system and results are encouraging.

Keywords
High Performance Cluster Computing, Job Scheduling, Global Memory Management, Local Memory Management, Distributed Shared Memory, Load balancing.

1. INTRODUCTION
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]. Cluster computing can be described as a fusion of the fields of parallel, high-performance, distributed, and high availability computing. In some scientific application areas such as high energy physics, bioinformatics, and remote sensing, we encounter huge amounts of data [2]. Managing such huge amounts of data in a centralized manner is almost impossible due to extensively increased data access time. But in multi cluster, 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. 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. 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. To improve the performance of the application, there is a need to move data from one node to another among the clusters. It indicates that efficient data movement through the interconnection of a cluster can become a major breakthrough. Dynamic load balancing can solve the load imbalance problem of cluster system and reduce its communication time period. In this paper, different new algorithms have been proposed one each for Memory Management, Scheduling of an in coming job and Load Balancing among multi clusters. These methods assign weights for each node to schedule an incoming job and then balancing the load dynamically using memory locality as the main factor. The main parameters considered in Load Balancing algorithm are partition size, CPU usage, memory usage and execution time.

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.

Kavitha Ranganathan and Ian Foster [4] 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 [5] 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.

William H. Bell et al. [6] find the design of the simulator OptorSim and Various replication algorithms.

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.

Christine Morin [8] 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[9] address the problem of harnessing the distributed memory resources in a cluster to support memoryintensive 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.

KA-PO Chow and et.al. proposed a load balancing scheme called COMET: Communication Efficient Load Balancing Strategy for Multi agent cluster computing[10]. In this they used directed graphs, but they were confined to multi agent applications only.

Alex K. Y. Cheung and Hans-Arno Jacobsen,[11] "Dynamic Load Balancing in Distributed Content-based Publish/Subscribe", White papers on load balancing, University of Toronto, pages. 21, July 2006.

Ch. Satyanarayana and et.al. proposed a new face recognition system using PCA which is associated to the application and accomplished a noteworthy performance development on a chronological execution [12].

3. DESIGN OF COMBINED MEMORY MANAGEMENTTECHNIQUE

3.1 Global and Local Memory

The proposed memory management technique comprises global memory and local memory. Global memory (Mg) is a distributed shared memory and is common for all the clusters whereas, local memory (M_l) is specific to the nodes of every individual cluster. The Global memory (Mg) takes care of moderating data accessibility within and among clusters and its goal is to optimize the use of nodes memory for global performance. The Local Memory takes care of data availability within the node and its goal is to optimize the use of node or workstation memory for local performance. The global memory (Mg) constitutes of persistent storage and temporary storage units [10]. In Global Memory Management, the data access rate of all the data in the temporary memory and permanent memories are considered. The data which is frequently accessed is stored in the persistent part and the less frequently accessed is stored in the temporary part. The goal of global memory management is to provide the best overall system performance by minimizing access time for all jobs in the clusters.

3.2 The Scheduler and Memory Management

While making the request, the user specifies the appropriate resources, the estimated execution time and the deadline. Then request is forwarded to the scheduler. The scheduler consists of a resource management system which maintains details regarding the resource availability and the resource which is under utilization.

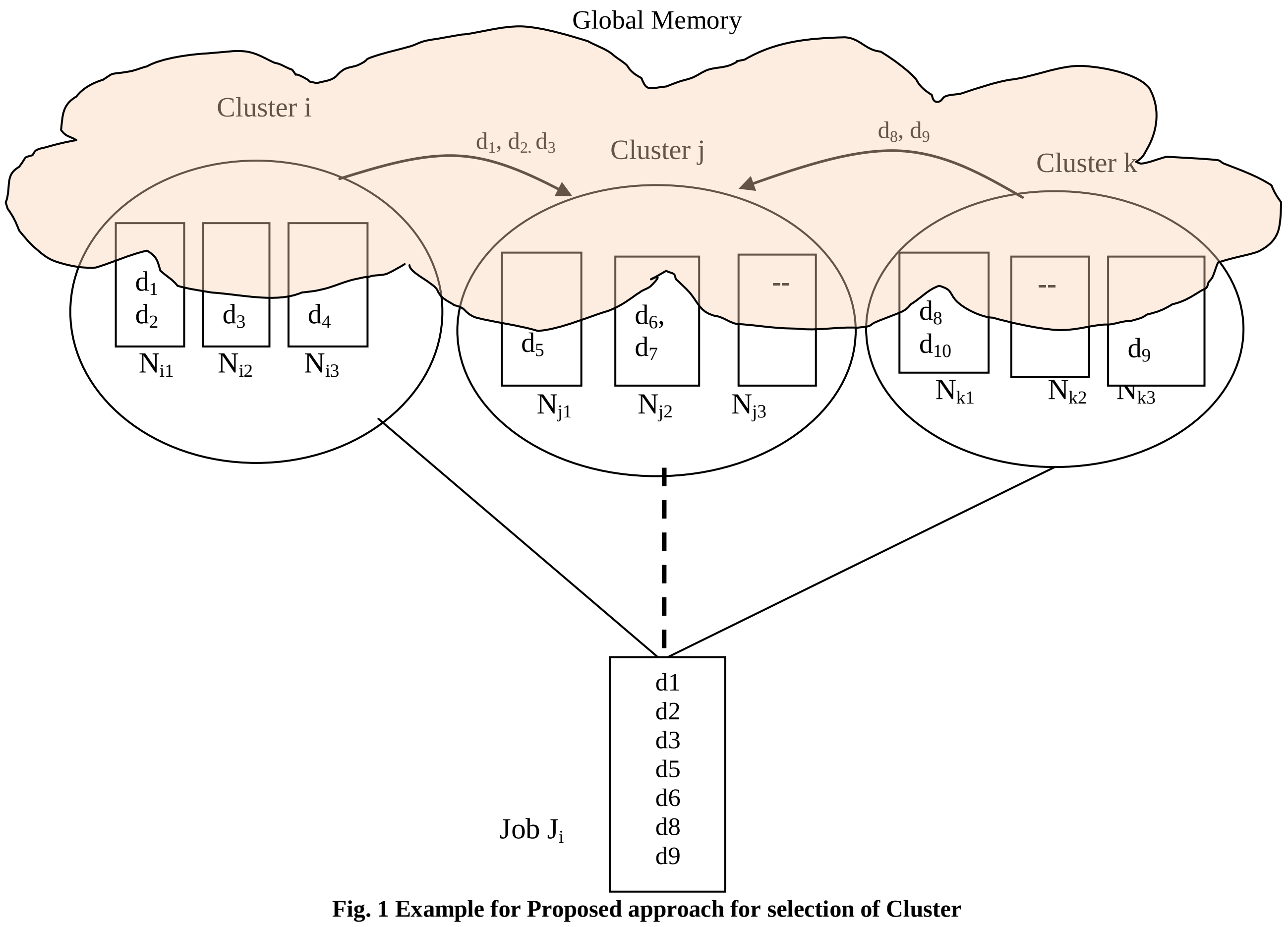


Fig. 1 Example for Proposed approach for selection of Cluster

The information is updated periodically and / or after the completion of running requests by the resource management system. 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 about the load of the processors of the nodes in 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 overcome this problem a new algorithm is proposed using global memory and local memory. The conventional scheduling algorithm blindly fixes a particular cluster taking into account the availability of data as the sole criterion and it would lead to performance degradation.

To illustrate the above scenario consider a particular request that requires a certain cluster that is identified for the given request which is based on the presence of major portion of required data and the cost for accessing remaining data is not considered. 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 scheduling algorithm that also gives the needed importance to the cost of accessing data as shown in fig. 1.

3.3 Scheduling Algorithm

Assumptions

a) SD_{Mg} is a set of data segments to be transferred from SS_C to global memory Mg.
b) SS_C is the set of clusters having the data segments in global memory SD_{Mg} that are required by the job J_i.
c) SS_{Ci} is the cluster C_i having data segments in the global memory SD_{Mg} that are required by the job J_i.
d) SD_{WC} is a set of data segments available within the concerned cluster $C_{Ji.}$
e) SQ_C is the information of all the clusters update between clusters in the system.

3.3.1 For data segments within a Cluster:

```
For each data segments in $SD_{Mg}$
  For each cluster in $SS_C$
    t = Calculated time to transfer data segments from $SS_{Ci}$ through Mg
  End
  $t_{min}$ = min (t)
  Update $SQ_C$
End
```

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

$$T_{WC} = \sum_{i=0}^{sizeof(SD_{WC})} t_{\min}$$

3.3.2 For data segments between Clusters:

```
For each data segments in $SD_{Mg}$
  For each cluster in $SS_C$
    t = Calculated time to transfer data segments from $SS_{Ci}$ through Mg
  End
  $t_{min}$ = min (t)
  Update $SQ_C$
End
```

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

$$T_{BC} = \sum_{i=0}^{sizeof(SD_{Mg})} t_{\min}$$

The total time needed to transfer the data segments 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 times 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.

4. GLOBAL AND LOCAL MEMORY MANAGEMENT TECHNIQUE

4.1 Memory management of nodes within a cluster

Data is normally transferred between nodes and clusters when it is not found in a particular node. When the data is available in the same node it uses its local memory. But, within the cluster and among the clusters, all the data is transferred through the temporary memory in the global memory where permanent memory contains the data that is accessed more frequently. Assume that the data access rate of each data segment is maintained in a vector V_{DAR} by every node in a cluster. There is a threshold value (*Thresh*) for the size of the node memory. Using V_{DAR} vector the used memory of the node can be calculated. If the used memory of a node exceeds threshold value, then the data segments which are not frequently accessed is to be removed from that node.

Each node in a cluster performs the following steps:

(1) Sort the vector V_{DAR} in ascending order
(2) Retrieve the V_{DAR} of all the remaining nodes within that cluster
(3) Find the non replicated data in the node and remove that data from the vector

S_{MN} is the size of the node memory within the cluster i.e. number of data segments residing in the concerned node.

```
Remove the data segment from the node with in the cluster
  $S_{MN}$ = size of all the remaining data segments in that node
        if ($S_{MN}$ < (Thresh - $\theta$))
              break
        end
    end
```

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

4.2. Memory Management of Global Memory

The global memory consists of both permanent memory and temporary memory. All the data segments are transferred between the clusters through the temporary memory in the global memory. Assume that the data access rate of all the data segments in the temporary memory and permanent memory are maintained in a vector. If a data in temporary memory is more frequently accessed, it must be transferred from temporary memory to permanent memory. After transferring the data, they must be removed from temporary memory.

4.2.1 Temporary Memory Management:

$SDA_R \rightarrow DA_R$ sorted in descending order.

P_M , $T_M \rightarrow$ Permanent and Temporary Memories

S_{TM} is the size of the temporary memory i.e. number of data segments (amount of data) residing in the temporary memory.

```
$for$ each value of $SDA_R$

$Move$ the data segment from $T_M$ to $P_M$
$S_{TM}$=size of all the data segments in the temporary memory
        $If (S_{TM} < (Thresh - \varphi))$
            $break$
        $end$
$end$
```

4.2.1 Permanent Memory Management:

If the used memory of the permanent memory exceeds a predefined threshold value, the data segments which are not frequently accessed are removed from permanent memory.

Assumptions:

DA_R $\rightarrow$Access rate of all the data segments in the P_M

SDA_R $\rightarrow DA_R$ sorted in ascending order.

S_{PM} is the size of the permanent memory i.e. number of data segments (amount of data) in the permanent memory.

```
$for$ each value of $SDA_R$
  Remove data segment from permanent memory
    $S_{PM}$=memory of all the remaining data segments in the
        permanent memory
        $If (S_{PM} < (Thresh - \varphi))$
            $break$
  $end$
$end$
```

5. EFFICIENT LOAD BALANCING IN FACE RECOGNITION SYSTEM

The main focus is on the load distribution and load balancing aspects of implementing a complex face recognition system using PCA which is an image processing application. Face recognition system has practical and potential application in areas such as security systems, criminal identification, video telephony, medicine, and so on. In the proposed system, the image database is decomposed into several sub databases, each placed in individual nodes of all clusters in a multi cluster environment. Each node calculates the features of all images of the sub database present in that node. When an input query image is given, its features are extracted and replicated to all the individual nodes in the different clusters. The features of the query image are compared against the features of the images in the sub databases. The results of the nearest matching images are back propagated to the master node in the multi cluster. The algorithm proposed for this application considers the load from each node, the node with minimum load is selected to make that node as load manger. The load manager is responsible for the distribution of image database and to all nodes. The Cluster Weightage Allocation algorithm gives minimum and maximum loaded nodes in the procedure. The proposed load balancing algorithm for face recognition is explained in section A.

5.1. Algorithmic steps for Load Balancing Scheme

The assumptions of scheduling algorithm for this system are as follows:

SN_{WC}	: Set of nodes within the cluster
SP_{WN}	: Set of processes running on the node N
NL_i	: The load on the node i
SQN_{WC}	: The information of all nodes at the load manager
A_c , C_c	: Vector of available clusters, Current cluster

```
For each $C_c$ cluster in $A_c$
  For each node in $SN_{WC}$
    For each Process in $SP_{WC}$
      M=calculate the load of a node based on the proposed
        Cluster Weightage Allocation (CWA) algorithm
      $NL_i=NL_i + M$
    End
   Update $SQN_{WC}$
  End
 $NL_{min} = \min \{NL_i\}$
 $NL_{heavy}=\max \{NL_i\}$
End
```

Here NL_{min} and NL_{heavy} are the nodes with minimum and maximum load respectively. And the load information of all nodes is collected by the load manager. This information is used for distribution and balance of the load and the threads are to be migrated from overloaded nodes to the under loaded nodes dynamically. If a node fails in the middle of processing, then the load manager finds the failure node and it takes the responsibility to share the load across the cluster with very less performance degradation. The CWA algorithm is explained in section B.

The new load J_L i.e. face database has to be distributed to every node in the multi cluster environment. After distribution of face data base, the load of each node is calculated as

$K= (\sum (LC_{ij}) + J_L)/N$

The distribution of load on each node is done by the algorithm given below

```
For each $C_c$ cluster in $A_c$
    For each node in $SN_{wc}$
        If $(LC_{ij}<k)$
        $LC_{ij}=LC_{ij} + (K - LC_{ij})$
    End
End
```

Where: Lc_{ij} $\rightarrow$ Load on ith cluster, jth node

J_L $\rightarrow$ $\sum LC_{ij}$ load on all the clusters .where $1<=i<=n$ and $1<=j<=N$

N $\rightarrow$ number of nodes in multi-clusters

SN_{WC} $\rightarrow$ set of nodes having the SF_{WC} in Mg within Cluster

SF_{WC} $\rightarrow$ set of files available with in the concerned cluster

5.2. Cluster Weightage Allocation (CWA) Algorithm

The proposed algorithm is mainly defined to allocate the jobs in the corresponding cluster with minimum memory utilization ($\mu_{lm)}$ to balance the load [10]. To illustrate the above scenario, let us consider a number of available clusters with many nodes. The main problem is how to allocate the incoming job to the corresponding cluster with immediate execution as well as the minimum load. Analyze the μ_{lm} of every cluster and sort it in ascending order. The cluster having least μ_{lm} has faster time than the other is chosen initially to allot the job. If the job is allocated to the particular node, it should always be noticed that the load balances each cluster. The total t_e of each node in a cluster is evaluated for assigning the job to the equivalent cluster. The weightage of each node in a cluster is calculated on the basis of time for which the job is allocated to the node which is having maximum weightage. The cluster weightage allocation algorithm is clearly explained as follows:

5.2.1 Algorithm for Weightage calculations of Clusters:

```
Set $\mu = 0$, s=0;
for each $C_C$ cluster in $A_C$
        s = size ($\mu_{lmn}$)
        for each memory utilization c in $\mu_{lmn}$
                $\mu_m = \mu + c$;
        end for
    For each CPU utilization c in $\mu_{lmn}$
         $\mu_l = \mu + c$;
        end for
         $\mu_{lmn}=\mu_m+\mu_1$;
        C[i]= $\mu_{lmn}$ /s;
    end for
$SA_C$ = sort ($A_C$ using C) dec;
for each index i of $SA_C$
        $W_C[i] = (i+1) / N_C$;
end for
```

Where: A_C	→	Vector of all available clusters
N_C, C_C	→	No of available clusters, Current cluster
W_C	→	Weightage of all clusters
C	→	Vector of total available memory utilization from all available clusters
μ_{lmn}	→	Memory utilization of each node in a cluster

In the above pseudo code, the CPU and memory usage of each node in the available clusters are sorted and stored it in SA_C and the weightage of each cluster is calculated.

5.2.2 Algorithm to select a job based on Partition size:

```
for i=1 to $N_c$
    for each node in $N_N$
        $\mu=\mu+ \mu_{lmn}$ [j]
    end
end
$C_A=\mu/S_p$
$\mu_{lmsj}$ =Sort ($\mu_{lmJ}$)$_{Asc}$
 $t_{sj} = t_j (\mu_{lmsj} [i])$
   for i = 0 to $N_J$-1
        $j_c= j_c+ \mu_{lmsj}[i]$
      if $j_c>=C_A$ then
            Index=current index of $\mu_{jp}$
            break
        end
   end
   for i=0 to index
     $\mu_{lmjp} [i] = \mu_{lmsj}[i]$
     $t_{jp}[i] = t_{sj}[i]$
end
```

Where: N_C	→	Number of available clusters
N_N	→	Number of nodes in each cluster
t_n	→	Total execution time of each node
t_r	→	Remaining time of each node
t_{ne}	→	Elapsed time of each node
μ_{lmn}	→	Memory utilization of each node in a cluster
N_j , S_P	→	Number of jobs, Partition size
μ_{lmj}	→	Memory needed in job utilization
t_j	→	Time required in job execution
μ_{lmcj}	→	Memory utilization of Current job
C_A	→	Available memory utilization
μ_{lmsj}	→	Memory utilization of sorted jobs
μ_{lmjp}	→	Memory utilization of job's partition size
μ_{lmsn}	→	Memory utilization of sorted nodes

In the above pseudo code, the job's CPU usage and the Execution time is selected by one-third of the jobs in the partition size.

```
for j=1 to $N_c$
    for each node i in $N_N$
        $$t_r[i] = \sum_{k=0}^{Nj} (t_n[k] - t_{ne}[k])$$
    end
      $t_{sr}$ =Sort $(t_r)_{Asc}$
         $\mu_{lmsn}[i] = \mu_{lmn}(t_{sr}[i])$
    for each memory utilization in $\mu_{lmsn}$
        $\mu=\mu+\mu_{lmsn}[i]$
        if $\mu>\mu_{lmcj}$ then
            Index = current memory utilization of $\mu_{lmsn}$
              break
        end
    end
   for each remaining time in $t_{sr}$
          t = t + $t_{sr}$ [i]
           If i++ > index then
              Break
          end
    end
end
```

From this, the cluster with minimum time is chosen to allot the job. Then the weightage of each node is calculated with the help of the memory usage. The formula for finding the weightage of each node is given as,

$$[1-(\mu_{lmn}[i]/100)] * W_c$$

The weightage is calculated for each node in a cluster and the selected job is allotted to the node which is having maximum weightage. In order to check the balances of each cluster, we have used to check with the following conditions.

$$J_c \geq \frac{N_J}{N_c} \quad |J_c[i] - J_c[i+1]| < \frac{2N_J}{N_c}$$

Where J_c is the number of jobs given to the cluster.

6. RESULTS

The algorithms are implemented and tested on PelicanHPC Operating System. The cluster and the file information are stored in the tables of a database. The data base is designed with MySQL tool. It uses three tables to store the information of all the data segments and clusters. There are three clusters in a multi-cluster architecture, they are {C_i, C_j, C_k} and the data segments that reside on system are {d_1, d_2, d_3… d_{10}} as shown in fig 1. Let the distribution of these data segments be C_i containing {d_1, d_2, d_3, d_4}, Cj containing {d_5, d_6, d_7} and C_k containing {d_8, d_9, d_{10}}. The conventional algorithm blindly assigns the job to the cluster C_i as this cluster contains more number of data segments as shown in fig.1. Before the actual experiment, a large database has been created which includes the data segments with various sizes and computed the data access times by transferring the data segments among the nodes in a multi-cluster environment, considering average network traffic. When a job is assigned to a particular node in a cluster its corresponding data segments may be present in other nodes of the same or remote clusters. The size of each data segment present in a node is compared with those data segment sizes that are in the database to get the nearest access time. Based on these access times, the total access times have been calculated for each node of a cluster. The data segment is moved in terms of data chunks. Data chunk is a maximum transmission unit. The size of a data chunk is fixed and varies from one file system to another. The numbers of data chunks for any data segment can be calculated by taking ceil value of the ratio of data segment size in kb and data chunk size.

The transfer time of data segment is the time required to transfer all its data chunks. For example data segment d4 of size 438 kb requires 4 data chunks and its transfer time is the time required to transfer 4 data chunks. If we consider the link bandwidth of 1Gbps and data chunk size of 128KB for NTFS then the minimum time required for transferring 128KB segment is 1048 micro seconds. The proposed scheduling algorithm first calculates the minimum access times taken for the data segments within the clusters (from data base tables) by each node in that cluster. If a data segment resides on a node then the access time for that data segment by that node is treated as zero. Similarly, the access times are calculated for all the data segments in each cluster. The access times between the nodes of the cluster C_i is shown in table I.

Table I: The Access Times Between each Pair of Nodes In Cluster C_i.

C_i	d1	d2	d3	D4
NT_{i1-i2}	3156	2107	1053	4656
NT_{i2-i3}	3219	2133	1056	4600
NT_{i1-i3}	3242	2108	1064	4658

Here NT_{i1-i2} represents the access time for a data segment from node N_{i1} to node N_{i2}. In our experiment cluster C_i, the node N_{i1} contains the data segments d_1, d_2, node N_{i2} contains d_3 and node N_{i3} contains the data segment d_4. The cluster C_i contains the data segments d_1, d_2, d_3 which are required by job J_i. So here the set SD_{WC} is $\{d_1, d_2, d_3\}$.

Now the total access times for each node in cluster C_i is as follows: t (N_{i1}) = 0 + 0 + 1053 = 1053 µsec
t (N_{i2}) = 3156 + 2107 + 0 =5263 µsec
t (N_{i3}) = 3242 + 2188 + 1056 =6406 µsec

From the above values the minimum access time is 1053 µsec from N_{i1} and this is T_{WC} for cluster C_i.

The information in table II shows the access times of the each pair of nodes N_{j1}, N_{j2} and N_{j3}. And these access times are only for data segments that reside on cluster C_j. The cluster C_j contains the data segments d_5, d_6 which are required by job J_i. So here the set SD_{WC} is $\{d_5, d_6\}$.

Table II: The access times between each pair of nodes in cluster C_j.

C_j	d5	d6	d7
NT_{j1-j2}	7346	4947	1053
NT_{j2-j3}	7350	4954	1088
NT_{j1-j3}	7400	4960	1145

Now the total access times for each node in cluster C_j is as follows: t (N_{j1}) = 0 + 4947 = 4947 µsec
t (N_{j2}) = 7346 + 0 = 7346 µsec
t (N_{j3}) = 7400 + 4954 =12354 µsec

From the above values the minimum access time is 4947 µsec from N_{j2} and this is T_{WC} for cluster C_j.

Table III: The access times between each pair of nodes in cluster C_k.

C_k	d8	d9	d10
NT_{k1-k2}	6292	3147	4198
NT_{k2-k3}	6380	3421	4242
NT_{k1-k3}	6396	3528	4213

The information in table III shows the access times of the each pair of nodes N_{k1}, N_{k2} and N_{k3}. The cluster C_k contains data segments d_8, d_9 which are required by job J_i. So SD_{WC} is $\{d_8, d_9\}$.

Now the total access times for each node in cluster C_k is as follows: t (N_{k1}) = 0 +3528 = 3528 µsec
t (N_{k2}) = 6292 + 3421 =9713 µsec
t (N_{k3}) = 6396 + 0 = 6396 µsec

From the above values the minimum access time is 3528 µsec from N_{k1} and this is T_{WC} for cluster C_k.

In the second part of algorithm, for each cluster, the minimum of the access times for the data segments that reside on other clusters is calculated.

Table IV: The access times of data segment between each pair of clusters.

	CT_{ij}	CT_{jk}	CT_{ik}
d1	3154	3296	3222
d2	2099	2244	2136
d3	1051	1053	1074
d4	4236	4245	4226
d5	7576	7431	7351
d6	5342	5628	5252
d7	1051	1053	1074
d8	6388	6497	6298
d9	3154	3296	3222
d10	4236	4245	4226

The access times of all data segments for each pair of clusters are given in Table IV. Here in table CT_{ij} represents the access time between cluster C_i and cluster C_j. CT_{jk} and CT_{ik} represent the access times between remaining pair of clusters (C_j, C_k) and (C_i, C_k) respectively. The data segments are transferred in between nodes across the remote cluster using global memory.

The access times between different clusters are shown as follows:

T_{BC} (C_i) = 0+0+0 + 7576 + 5342 + 6298 + 3222=22438 µsec
$T_{BC}$$(C_j)$ =3154+2099+1051+0+0+6497 + 3296 = 16097 µsec
$T_{BC}$$(C_k)$ = 3222+2136+1074+7431+ 5628+0+0= 19491 µsec

To designate a cluster the algorithm uses the summation of T_{BC} and T_{WC} as the parameter. The minimum of these are considered to assign the job.

T (C_i) = 22438 + 1053 = 23491 µsec
T (C_j) = 16097 + 4947 = 21044 µsec
T (C_k) = 19491 + 3528 = 23019 µsec

In our experiment N_{i1} has more data segments compare to all other nodes. So existing scheduling algorithm assigns the job j_i to N_{i1} node in the cluster C_i. As per the above calculations of data access times, it takes 23491 µsec to gather other segments from the remote clusters. Based on the proposed scheduling algorithm it is clear that minimum of these total access times is for cluster C_j that designated for the job J_i as shown in fig. 1. The cluster C_j takes 21044 µsec to gather other segments from the remote clusters. The proposed scheduling algorithm assign the job j_i to node N_{j2} in cluster C_j. Thus the proposed algorithm schedules the jobs to the clusters considering the data access times as the key factor. The fig. 2 shows the transfer time between nodes of various data segments sizes from 0kb to 512kb.

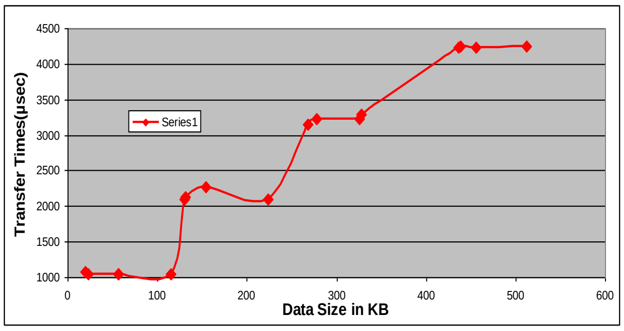
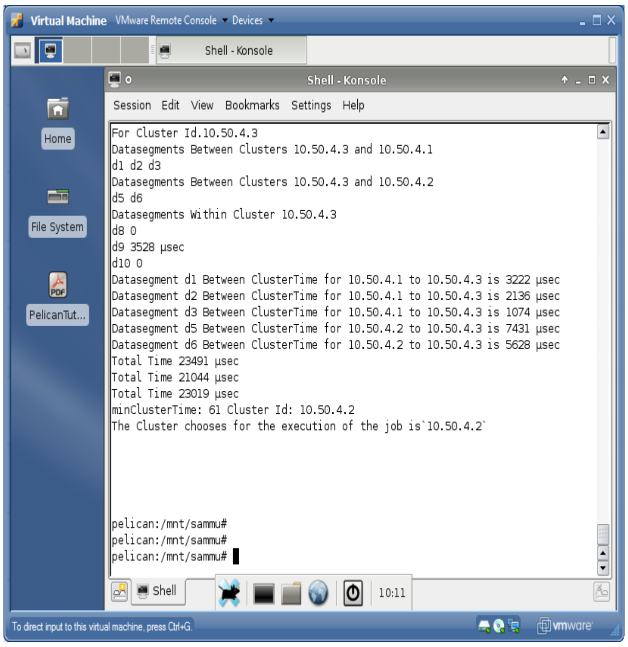


Fig. 2 Transfer time between nodes of various data segment sizes from 0kb to 512kb.

A snapshot of data access times between the clusters and within the cluster is illustrated in fig. 3.



```
For Cluster Id.10.50.4.3
Datasegments Between Clusters 10.50.4.3 and 10.50.4.1
d1 d2 d3
Datasegments Between Clusters 10.50.4.3 and 10.50.4.2
d5 d6
Datasegments Within Cluster 10.50.4.3
d8 0
d9 3528 µsec
d10 0
Datasegment d1 Between ClusterTime for 10.50.4.1 to 10.50.4.3 is 3222 µsec
Datasegment d2 Between ClusterTime for 10.50.4.1 to 10.50.4.3 is 2136 µsec
Datasegment d3 Between ClusterTime for 10.50.4.1 to 10.50.4.3 is 1074 µsec
Datasegment d5 Between ClusterTime for 10.50.4.2 to 10.50.4.3 is 7431 µsec
Datasegment d6 Between ClusterTime for 10.50.4.2 to 10.50.4.3 is 5628 µsec
Total Time 23491 µsec
Total Time 21044 µsec
Total Time 23019 µsec
minClusterTime: 61 Cluster Id: 10.50.4.2
The Cluster chooses for the execution of the job is`10.50.4.2`



pelican:/mnt/sammu#
pelican:/mnt/sammu#
pelican:/mnt/sammu#
```

Fig. 3 snapshot of proposed scheduling algorithm.

Among three clusters, cluster C_j has minimum total access time and the job is allocated to the cluster C_j. Several experiments have been performed on PelicanHPC. First the face recognition application is executed on a stand alone system of configuration Pentium IV with 512MB memory. Fig.4 shows the evaluation of the execution. At each time, the work load i.e., the number of images, for which the Eigen faces are to be calculated, is increasing and the corresponding execution time has to be considered. The execution time for 200 images is relatively small i.e., 3.8 sec. The load of images is increased with regular interval and tested. Beyond 2000 images the system's response time is very high and the application is executed up to 2000 images. For heavy work load, the execution time is 24.8 sec which is very high.

The same application is executed on described cluster under the proposed system, varying work load as before. The results have shown that, under the small work load the difference between the execution times of sequential and proposed system is relatively small.

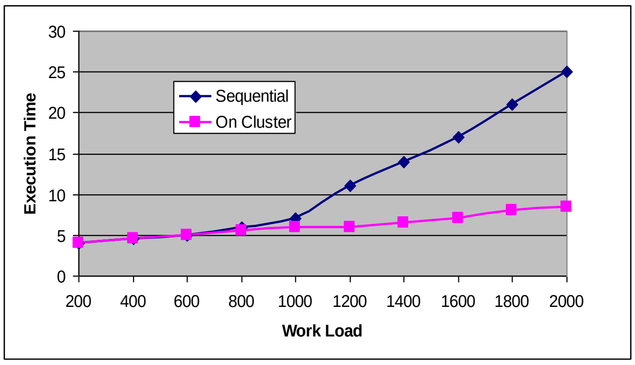


Fig.4 Execution time of Application in sequential and on cluster environment with 6 nodes.

Beyond 600 images the difference between these two is noticeable, with 2000 images; the difference between the execution times is very high. At heavy workloads, the cluster is performing very well than the other approaches.

The application is tested on the PelicanHPC cluster under the proposed system varying number of nodes and with workload of 2000 images.

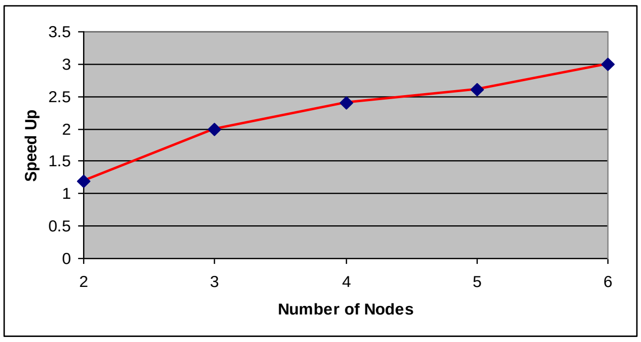


Fig.5 Relative Speed up for the corresponding number of nodes.

At each time a node is connected to the cluster, and the speedup is calculated under the same workload, the results have shown that the cluster with 6 nodes exhibits high speedup in the response time as in fig.5. When there are many nodes, the load management overhead should be considered.

With two nodes the execution shows a speed up of 1.5 to 2.5, for six nodes it has shown speedup 3 in execution as shown in fig 5.

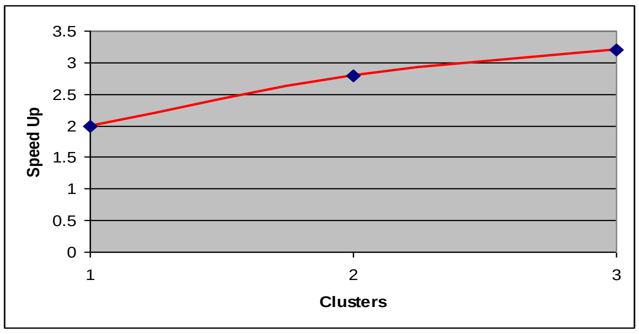


Fig.6 Relative Speed up for the corresponding number of clusters.

The efficient distribution at initial step, improved the performance. There is a threshold on the number of nodes according to the application, because the overhead concerned with the load management and monitoring could increase with the number of nodes in the cluster, which in turn shows impact on the performance. The same application has been tested on multi cluster having three clusters with three nodes each. With single cluster the speed up is 2, the 2nd cluster it is 2.8 and for 3rd cluster the speed up is 3.2 shown in the bellow fig.6.

6. CONCLUSION

The Proposed *Scheduling Algorithm* considers data access times as the main factor to decide a cluster or node on which that particular request is to be served. Experimental results show a substantial improvement in data access time across the clusters by reducing the execution time. Providing due consideration to data access latency besides computational capability proves worthwhile.

The proposed, *Memory Management Technique* uses a combination of Local Memory and Global Memory that buffers data, based on access patterns. The global Memory takes care of moderating data accessibility across nodes within the clusters and among the clusters. The Local Memory takes care of data availability within the node and its goal is to optimize the use of node or workstation memory for local performance. The Global Memory is bifurcated into a Temporary Storage part and a Persistent Storage part. Data placed in these two parts is updated dynamically and at regular intervals, based on the pattern of usage. The concept comes handy when the data to be accessed from another cluster is markedly time taking than the same data accessed through the Global Memory.

An efficient *Dynamic Load Balancing Mechanism* has been developed with a new load metric that considers Partition Size, CPU usage, Memory usage and Execution time of jobs as main factors to decide the load of each workstation. These factors are optimized by proposed *Cluster Weightage Allocation Algorithm*. Based on the CPU and Memory usage, CWA assigns weights to each node in the clusters which in turn helps in assigning a new incoming job. CWA also considers the remaining time at each cluster in assigning weights which balances the load across the clusters. And using the number of page faults as parameter to represent memory locality for efficient process migration from heavily loaded to low loaded node will show the optimum performance. A number of tests were performed on different scenarios and from these results we can conclude that the combination of the proposed Scheduling Algorithm, Memory Management Algorithm and Load Balancing Algorithm exhibit better performance than traditional schemes. The task distribution and load balancing of a complex application increases the speedup in the execution time and also reduces the response time. The face recognition system using PCA is the image processing application which undergoes many more computations and also heavy I/O, memory intensive loads. The execution of this application using Load Balancing Algorithm on a pelicanHPC cluster has shown much improvement in speedup in the execution rather than sequential execution.

7. 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] 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.

[5] 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.

[6] 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.

[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] 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.

[9] 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.

[10] KA-PO CHOW, YU-KWONG KWOK1, HAI JIN, AND KAI HWANG, Comet: *A Communication-Efficient Load Balancing Strategy for Multi-Agent Cluster Computing*.

[11] Alex K. Y. Cheung and Hans-Arno Jacobsen, "Dynamic Load Balancing in Distributed Content-based Publish/Subscribe", White papers on load balancing, University of Toronto, pages. 21, July 2006.

[12] Ch.Satyanarayana, D.Haritha, P.Sammulal and L.pratap Reddy "*Updation of facespace for Face Recognition using PCA*", in the proceedings of IEEE international conference on RF and Signal Processing Systems, RSPS-2008 conducted in 1st-2nd February at KLCE, Vijayawada.AP. 195-202