Assignment No.: 04

Title: Implement a graph of unloaded cluster for several number of nodes

Aim: Write a program on an unloaded cluster for several different numbers of nodes and record the time taken in each case. Draw a graph of execution time against the number of nodes.

Objectives:

- To understand clustering
- To implement Message Passing Interface

Theory:

Cluster:

A computer cluster consists of a set of loosely or tightly connected computers that work together so that, in many respects, they can be viewed as a single system. Unlike grid computers, computer clusters have each node set to perform the same task, controlled and scheduled by software.

The components of a cluster are usually connected to each other through fast local area networks ("LAN"), with each node (computer used as a server) running its own instance of an operating system. In most circumstances, all of the nodes use the same hardware and the same operating system, although in some setups (i.e. using Open Source Cluster Application Resources (OSCAR)), different operating systems can be used on each computer, and/or different hardware.

They are usually deployed to improve performance and availability over that of a single computer, while typically being much more cost-effective than single computers of comparable speed or availability.

Computer clusters emerged as a result of convergence of a number of computing trends including the availability of low-cost microprocessors, high speed networks, and software for high-performance distributed computing. They have a wide range of applicability and deployment, ranging from small business clusters with a handful of nodes to some of the fastest supercomputers in the world such as IBM's Sequoia.

Loading & Unloading Clusters/ Onlining & Offlining:

After a cluster resource is enabled, the load and unload scripts take care of starting and stopping the services or mounting and dismounting the volumes that are configured in the resource. You start services and mount devices by onlining the resource. You stop services and unmount devices by offlining the resource.

Onlining a resource runs the load script, which loads the resource on its primary preferred node, or on an alternate preferred node if possible, according to the order in its Preferred Nodes list. No action is taken if the preferred nodes are not active in the cluster, or if there are Resource Mutual Exclusion conflicts on its active preferred nodes.

Offlining a resource runs the unload script and unloads the resource from the server. The resource cannot be loaded on any other servers in the cluster and remains unloaded until you load it again.

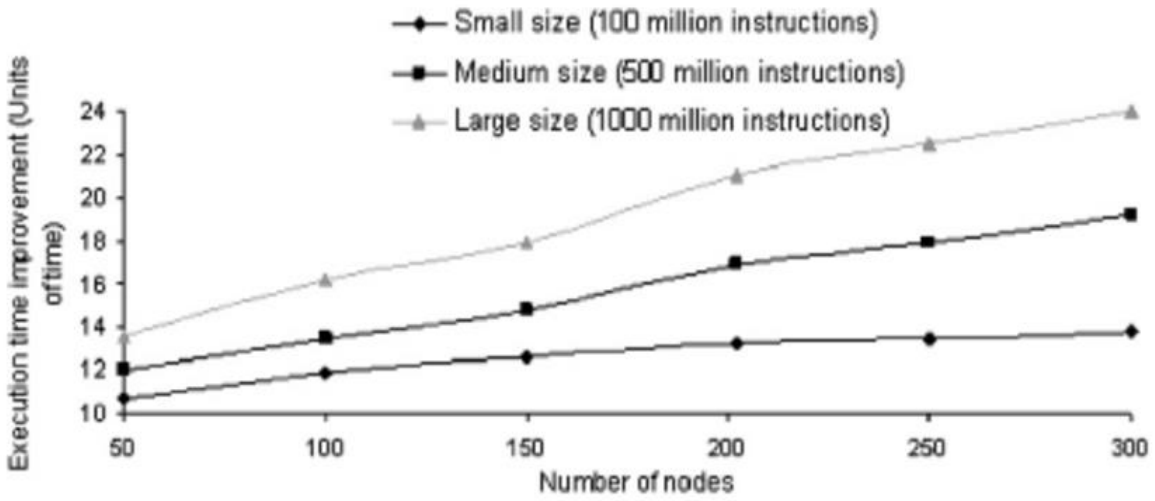

Figure 1 illustrates the improvement of the exe- cution time according to the nodes number.

n: number of nodes in a cluster (its cardinality load : load of a node i = (2) n

μ : average load of a cluster.

Load Balancing Threshold value: The threshold is a value that is used to indicate whether a node is heavily or lightly loaded. It is important to de- termine an appropriate threshold for a good load balancing algorithm.If the threshold is set too low, excessive load balancing will occur, thus causing degradation in performance (as this will result in high communication costs) .

However, if the threshold is set too high, the load balanc- ing mechanism will not be very effective.

Communication cost: This is the total number of load balancing-related messages sent by a node. It gives a measure of the overhead created for balancing loads, which is also an indication of our algorithm cost in terms of energy. After simulations, the obtained results are ana- lyzed by studying the impact of the considered parameters on our load balancing algorithm per- formance. To evaluate the performance of our algorithm in term of execution time improvement, we varied the number of nodes and registered the execu- tion time for different sizes of works (small, average, large) .

Benefits of clustering

1. Clustering solves this as the load of increasing users can be evened out on all the nodes of the cluster.

2. Combined resources of all the nodes of a cluster would greatly boost the performance, as well as allow flexible scalability.

3. Clustering can also reduce the time complexity of the program, another node can take on the responsibilities of a node that expires without letting the affect of the expired node to be felt by the users.

4. Caching provides better data management as well as fast data access. Efficient recovery from any sort of malfunction is possible.

Mathematical Model:

Let M be the system.

M = {I, P, O, S, F}

Let I be the input. I = {k} k = any number.

Let P be the process. P = {n, q, r} n=squaring the number. q=calculating the run time

r= plotting the graph

Let O be the Output. O = {c} c =square of the number and run time.

Let S be the case of Success. S = {l} l = Satisfied all conditions.

Let F be the case of Failure.

F = {f} f = Satisfied result not generated.

Conclusion: We have implemented a graph of unloaded cluster for several number of nodes.

Code

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>

#define v 1 /* verbose flag, output if 1, no output if 0 */
#define tag 100 /* tag for sending a number */

int main ( int argc, char *argv[] )
{
        int p,myid,i,f,*x;
        double start, end;
        MPI_Status status;

        MPI_Init(&argc,&argv);
        MPI_Comm_size(MPI_COMM_WORLD,&p);
        MPI_Comm_rank(MPI_COMM_WORLD,&myid);

        if(myid == 0) /* the manager allocates and initializes x */
        {
                x = (int*)calloc(p,sizeof(int));
                x[0] = 50;
                for (i=1; i<p; i++) x[i] = 2*x[i-1];

                if(v>0)
                {
                        printf("The data to square : ");
                        for (i=0; i<p; i++)
                        printf(" %d",x[i]);
                        printf("\n");
                }
        }

        if(myid == 0) /* the manager copies x[0] to f */
        {               /* and sends the i-th element to the i-th processor */
                f = x[0];
                for(i=1; i<p; i++)
                MPI_Send(&x[i],1,MPI_INT,i,tag,MPI_COMM_WORLD);
        }




        else /* every worker receives its f from root */
        {
                MPI_Recv(&f,1,MPI_INT,0,tag,MPI_COMM_WORLD,&status);

                if(v>0)
                        printf("Node %d will square %d\n",myid,f);
        }

        start = MPI_Wtime();

        f *= f;         /* every node does the squaring */

        if(myid == 0) /* the manager receives f in x[i] from processor i */
                for(i=1; i<p; i++)
                        MPI_Recv(&x[i],1,MPI_INT,i,tag,MPI_COMM_WORLD,&status);

        else     /* every worker sends f to the manager */
```

```
                MPI_Send(&f,1,MPI_INT,0,tag,MPI_COMM_WORLD);

        if(myid == 0) /* the manager prints results */
        {
                x[0] = f;
                printf("The squared numbers : ");
                for(i=0; i<p; i++)
                printf(" %d",x[i]);
                printf("\n");

                end = MPI_Wtime();
                printf("Runtime = %f\n", end-start);
        }

        MPI_Finalize();
        return 0;
}
```

Code

```
mpiu@DB21:/mirror/mpiu/CL4 prog/A-4$ mpicc a4.cpp



mpiu@DB21:/mirror/mpiu/CL4 prog/A-4$ mpiexec -n 4 -f machinefile ./a.out

The data to square :  50 100 200 400

Node 1 will square 100

Node 2 will square 200

Node 3 will square 400

The squared numbers :  2500 10000 40000 160000

Runtime = 0.000386

mpiu@DB21:/mirror/mpiu/CL4 prog/A-4$
```