Solution of Exercise Sheet 2

Exercise 1 (Cluster Computing)

1. Give a short definition of Cluster Computing.

   Clustering is parallel computing on systems with distributed memory.

2. What is a Cluster of Workstations?

   The nodes are only available at specific times. During normal working times, the employees use the nodes of such a cluster system as workstations.

3. Name an advantage of Clusters, which follow the Glass-house installation concept.

   Some advantages are:

   - *Fast access to all components for maintenance and troubleshooting.*
   - *Nodes can be connected via high-performance networks.*
   - *Increased protection against sabotage.*

4. Name a drawback of Clusters, which follow the Glass-house installation concept.

   In case of a power failure or fire in the building, the operation of the entire cluster is at risk.

5. Name an advantage of Clusters, which follow the Campus-wide installation concept.

   It is hard to destroy the cluster completely.

6. Name a drawback of Clusters, which follow the Campus-wide installation concept.

   Some drawbacks are:

   - *It is impossible to use high-performance computer networks.*
   - *Often, the nodes contain different hardware components.*

7. How can the availability of a system be calculated?

   mean uptime = Mean Time Between Failures
   mean downtime = Mean Time To Repair

$$\text{availability} = \frac{\text{mean uptime}}{\text{mean uptime} + \text{mean downtime}}$$

8. What is the objective of High Availability Clustering?

   High availability of the cluster service(s).

9. By which approach does High Availability Clustering achieve its objective?

   With redundancy of nodes and their components and by avoiding a single point of failure.

10. What is an Active/Active-Cluster?

    All nodes run the same services. All nodes are in active state. If nodes fail, the remaining active nodes need to take over their tasks.

11. What is an Active/Passive-Cluster?

    During normal operation, at least a single node is in passive state. Nodes in passive state do not provide services during normal operation. If a node fails, a passive node takes over its services.

12. What is the meaning of Failover?

    A node takes over the services of a failed node.

13. What is the meaning of Failback?

    If failed nodes are operational again, they report their status to the load balancer and get new jobs assigned in the future.

14. What causes a Split Brain?

    A split brain may result, when a connection failure between nodes occur, while the computers operate properly. A tool like Heartbeat, which monitors the presence (or disappearance) of nodes, assumes that nodes are broken. Each node declares itself to be the primary node. In a Active/Passive-Cluster, this causes a failure of the cluster (offered services).

15. What is the result of a Split Brain?

    If distributed storage is used, write requests cause differing data on the nodes. The data is no longer consistent. It is difficult to fix the broken consistency without loosing data.

16. Explain the difference between the Shared Nothing and the Shared Disk Clusters.

    In a Shared Nothing cluster, each node has its own storage resource.

In a Shared Disk cluster, all nodes have access to a shared storage.

17. Explain the main difference between a SAN (Storage Area Network) and a NAS (Network Attached Storage).

    A SAN provides block-level access to storage devices via the network.

    A NAS provides file system-level access to storage devices via the network.

18. What is the objective of High Performance Clustering?

    High computing power

19. Name an advantage of High Performance Clusters versus supercomputers.

    Some examples for advantages of High Performance Clusters are:

    - *Low price and vendor independence.*
    - *Defective components can be obtained in a quick and inexpensive way.*
    - *It is easy to increase the performance in a short time via additional nodes.*

20. Name a drawback of High Performance Clusters versus supercomputers.

    High administrative and maintenance costs, compared with mainframes.

21. What is a Beowulf Cluster?

    It is a High Performance Cluster and the nodes use a free operating system. Beowulf clusters consist of commodity PCs or workstations. The nodes of a Beowulf cluster are used only for the cluster

22. What is a Wulfpack Cluster?

    It is a High Performance Cluster and the nodes use a Windows operating system.

23. What is the objective of High Throughput Clustering?

    Maximize the throughput (handling of small tasks/requests).

24. Can High Throughput Clusters be used to process the same tasks as High Performance Clusters?

    No. Such clusters are not used for extensive calculations. Tasks for High Throughput Clusters must not be split into sub-tasks. The individual tasks (requests) are small and a single PC could handle them. Typical fields of application of High Throughput Clustering web servers and internet search engines.

If large compute jobs need to be processed, a High Performance Cluster is required.

If multiple small compute jobs (in a short time) need to be processed, a High Throughput Cluster is required.

Exercise 2 (MPI Cluster)

Build up a MPI Cluster with at least 2 virtual nodes (e.g. with VirtualBox or any other virtualization software) or alternatively with at least 2 physical nodes.

Write down precise instructions with the steps you performed to build up the MPI cluster.

Execute a simple program to demonstrate the functioning of your cluster.

This example works for a cluster of eight physical Raspberry Pi nodes with the operating system Raspbian, based on Debian Wheezy. First, we need to install some packages.

```
$ sudo apt-get update && sudo apt-get -y install make gcc g++ openmpi-bin
    openmpi-common libopenmpi-dev
```

Next, the SSH keys need to be distributed amoung the nodes. It is required that the master node (which is used to execute the MPI jobs) can execute commands on all worker nodes (the master is in out example a worker not too) without the need to type in a password.

The nodes have in our example the IP addresses `10.0.0.110 - 10.0.0.117`

```
$ ssh-copy-id -i ~/.ssh/id_rsa.pub pi@10.0.0.110
$ ssh-copy-id -i ~/.ssh/id_rsa.pub pi@10.0.0.111
$ ssh-copy-id -i ~/.ssh/id_rsa.pub pi@10.0.0.112
$ ssh-copy-id -i ~/.ssh/id_rsa.pub pi@10.0.0.113
$ ssh-copy-id -i ~/.ssh/id_rsa.pub pi@10.0.0.114
$ ssh-copy-id -i ~/.ssh/id_rsa.pub pi@10.0.0.115
$ ssh-copy-id -i ~/.ssh/id_rsa.pub pi@10.0.0.116
$ ssh-copy-id -i ~/.ssh/id_rsa.pub pi@10.0.0.117
```

Insert the hostnames (`pi110 - pi117`*) and the IP addresses into the file* `/etc/hosts`

```
10.0.0.110    pi110
10.0.0.111    pi111
10.0.0.112    pi112
10.0.0.113    pi113
10.0.0.114    pi114
10.0.0.115    pi115
10.0.0.116    pi116
```

```
10.0.0.117    pi117
```

It is helpful to have a distribute file system which can be accessed by all nodes of the cluster. This can be used to store the executable and the hosts file. Otherwise it must be distributed manually to all nodes of the cluster. In this example, a GlusterFS distributed file system is mounted on all nodes inside the folder `\glusterfs`

```
sudo mkdir /glusterfs/MPI_examples
sudo chmod 777 /glusterfs/MPI_examples/
```

Insert the following source code into the file `/glusterfs/MPI_examples/hello_world.c`

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

int main(int argc, char *argv[]) {
  int size, rank, namelen;
  char processor_name[MPI_MAX_PROCESSOR_NAME];

  // Initialize the MPI environment
  MPI_Init(&argc, &argv);

  // How many CPU (cores) contains the MPI environment?
  MPI_Comm_size(MPI_COMM_WORLD, &size);

  // What is the number of our CPU core?
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);

  // What is the name of the CPU core?
  MPI_Get_processor_name(processor_name, &namelen);

  // Output of each process
  printf("I am process nr. %d of %d on %s\n", rank, size, processor_name);

  // Terminate the MPI environment
  MPI_Finalize();

  // Terminate the programm with exit code 0 (EXIT_SUCCESS)
  return 0;
}
```

Create a file `hosts.mpi` *with contains the hostnames of all hosts, which shall be used for the MPI environment.*

```
$ for i in 0 1 2 3 4 5 6 7; do echo pi11$i slots=4 >> /glusterfs/
    MPI_examples/hosts_4cores.mpi; done
$ cat /glusterfs/MPI_examples/hosts_4cores.mpi
pi110 slots=4
```

```
pi111 slots=4
pi112 slots=4
pi113 slots=4
pi114 slots=4
pi115 slots=4
pi116 slots=4
pi117 slots=4
```

Compile the MPI application:

```
$ mpicc /glusterfs/MPI_examples/hello_world.c -o /glusterfs/MPI_examples/
    hello_world
```

Execute the MPI application:

```
$ mpirun -np 32 --hostfile /glusterfs/MPI_examples/hosts_4cores.mpi /
    glusterfs/MPI_examples/hello_world
I am process nr. 0 of 32 on pi110
I am process nr. 2 of 32 on pi110
I am process nr. 3 of 32 on pi110
I am process nr. 7 of 32 on pi111
I am process nr. 6 of 32 on pi111
I am process nr. 5 of 32 on pi111
I am process nr. 27 of 32 on pi116
I am process nr. 17 of 32 on pi114
...
```

Exercise 3 (Approximate π via Monte Carlo)

π can be approximated via Monte Carlo simulation.

Approach: Inscribe a circle of radius r inside a square with side length $2r$.

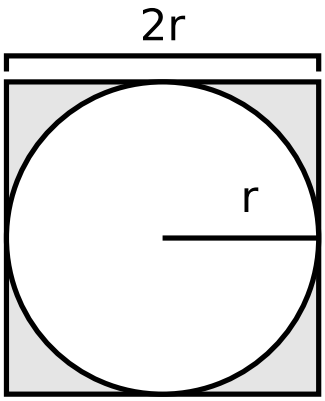
Generate random dots in the square. The number of dots in A_C in relation to the number of dots in A_S is equal to the surface ratio.

$$A_S = (2r)^2 = 4r^2$$

$$A_C = \pi r^2 \implies \pi = \frac{A_C}{r^2}$$

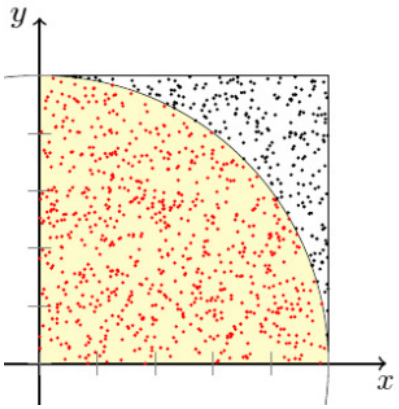
$$\frac{A_C}{A_S} = \frac{\pi r^2}{4r^2} \implies \frac{A_C}{A_S} = \frac{\pi}{4}$$

The dots can be generated in parallel by the workers. The master receives the dots and calculates π.



A = Surface ratio
r = Radius
C = Circle
S = Square

Image source: Wikipedia



Develop a MPI application, which calculates π via Monte Carlo simulation.

Test your MPI application with with the MPI cluster you already created and with different numbers or worker nodes to discover if your application scales well with a growing number of worker nodes.

It is helpful to have a distribute file system which can be accessed by all nodes of the cluster. This can be used to store the executable and the hosts file. Otherwise it must be distributed manually to all nodes of the cluster.

Insert the following source code into the file `/glusterfs/MPI_examples/pi_mpi2.c`

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

int main(int argc, char *argv[]) {

  int myid,nprocs;

  long int npts = 1e8;

  long int i,mynpts;

  double f,sum,mysum;
  double xmin,xmax,x;

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

  if (myid == 0) {
    mynpts = npts - (nprocs-1)*(npts/nprocs);
  } else {
    mynpts = npts/nprocs;
  }

  mysum = 0.0;
```

```
  xmin = 0.0;
  xmax = 1.0;

  srand(time(0));

  for (i=0; i<mynpts; i++) {
    x = (double) rand()/RAND_MAX*(xmax-xmin) + xmin;
    mysum += 4.0/(1.0 + x*x);
  }

  MPI_Reduce(&mysum,&sum,1,MPI_DOUBLE,MPI_SUM,0,MPI_COMM_WORLD);

  if (myid == 0) {
    f = sum/npts;
    printf("PI calculated with %ld points = %f \n",npts,f);
  }

  MPI_Finalize();
}
```

Compile the MPI application:

```
$ mpicc glusterfs/MPI_examples/pi_mpi2.c -o /glusterfs/MPI_examples/
    pi_mpi2
```

Switch to the appropriate directory:

```
$ cd /glusterfs/MPI_examples/
$ pwd
/glusterfs/MPI_examples/
```

In this example, the MPI application was executed on a cluster with 128 CPU cores (4 cores per physical node) with different numbers of CPU cores:

```
$ time mpirun -np 128 --hostfile hosts_4cores.mpi pi_mpi2
PI calculated with 1000000000 points = 3.141601

real  0m6.276s
user  0m14.050s
sys 0m1.520s

$  time mpirun -np 64 --hostfile hosts_4cores.mpi pi_mpi2
PI calculated with 1000000000 points = 3.141592

real  0m7.088s
user  0m20.120s
sys 0m1.080s

$ time mpirun -np 32 --hostfile hosts_4cores.mpi pi_mpi2
PI calculated with 1000000000 points = 3.141580

```

```
real  0m10.928s
user  0m35.800s
sys 0m0.850s

$ time mpirun -np 16 --hostfile hosts_4cores.mpi pi_mpi2
PI calculated with 1000000000 points = 3.141604

real  0m19.280s
user  1m9.210s
sys 0m0.670s

$ time mpirun -np 8 --hostfile hosts_4cores.mpi pi_mpi2
PI calculated with 1000000000 points = 3.141601

real  0m36.172s
user  2m17.150s
sys 0m0.500s

$ time mpirun -np 4 --hostfile hosts_4cores.mpi pi_mpi2
PI calculated with 1000000000 points = 3.141612

real  1m9.820s
user  4m33.240s
sys 0m0.420s
```

Explanation of the output of the time command.

Real is wall clock time - time from start to finish of the call. This is all elapsed time including time slices used by other processes and time the process spends blocked (for example if it is waiting for I/O to complete).

User is the amount of CPU time spent in user-mode code (outside the kernel) within the process. This is only actual CPU time used in executing the process. Other processes and time the process spends blocked do not count towards this figure.

Sys is the amount of CPU time spent in the kernel within the process. This means executing CPU time spent in system calls within the kernel, as opposed to library code, which is still running in user-space. Like 'user', this is only CPU time used by the process. See below for a brief description of kernel mode (also known as 'supervisor' mode) and the system call mechanism.

Real represents the actual elapsed time, while user and sys values represent CPU execution time. As a result, on a multicore system, the user and/or sys time (as well as their sum) can actually exceed the real time.

Source of the explanation: `http://stackoverflow.com/questions/556405/what-do-real-user-and-sys-mean-in-the-output-of-time1`