An EC2 Placement Group is a logical grouping of instances that allows the creation of a virtual

cluster. When several instances are launched as an EC2 Placement Group, the virtual cluster has

a high-bandwidth interconnect system suitable for network-bound applications. The cluster computing

instances require a hardware virtual machine (HVM) ECB-based machine image, whereas other

instances use a paravirtual machine (PVM) image. Such clusters are particularly useful for highperformance

computing when most applications are communication intensive.

Once a placement group is created, MPI (Message Passing Interface) can be used for communication

among the instances in the placement group. MPI is a de facto standard for parallel applications

using message passing, designed to ensure high performance, scalability, and portability; it is

a language-independent “message-passing application programmer interface, together with a protocol

and the semantic specifications for how its features must behave in any implementation” [146]. MPI

supports point-to-point as well as collective communication; it is widely used by parallel programs

based on the same program multiple data (SPMD) paradigm.

The following C code [146] illustrates the startup of MPI communication for a process group

MPI\_COM\_PROCESS\_GROUP consisting of a number of nprocesses; each process is identified by its

rank. The run-time environment mpirun or mpiexec spawns multiple copies of the program, with the

total number of copies determining the number of process ranks in MPI\_COM\_PROCESS\_GROUP.

#include <mpi.h>

#include <stdio.h>

#include <string.h>

#define TAG 0

#define BUFSIZE 128

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

{

char idstr[32];

char buff[BUFSIZE];

int nprocesses;

int my\_processId;

int i;

MPI\_Status stat;

MPI\_Init(&argc,&argv);

MPI\_Comm\_size(MPI\_COM\_PROCESS\_GROUP,&nprocesses);

MPI\_Comm\_rank(MPI\_COM\_PROCESS\_GROUP,&my\_processId);

MPI\_SEND and MPI\_RECEIVE are blocking send and blocking receive, respectively; their

syntax is:

int MPI\_Send(void \*buf, int count, MPI\_Datatype datatype,

int dest, int tag,MPI\_Comm comm)

int MPI\_Recv(void \*buf, int count, MPI\_Datatype datatype,

int source, int tag, MPI\_Comm comm, MPI\_Status \*status)

with

buf − initial address of send buffer (choice).

count − number of elements in send buffer (nonnegative integer).

datatype − data type of each send buffer element (handle).

dest − rank of destination (integer).

tag − message tag (integer).

comm − communicator (handle).

Once started, every process other than the coordinator, the process with rank = 0, sends a message

to the entire group and then receives a message from all the other members of the process group.

if(my\_processId == 0)

{

printf(“%d: We have %d processesÄn", my\_processId, nprocesses);

for(i=1;i<nprocesses;i++)

{

sprintf(buff, “Hello %d! “, i);

MPI\_Send(buff, BUFSIZE, MPI\_CHAR, i, TAG, MPI\_COMM\_PROCESS\_

GROUP);

}

for(i=1;i<nprocesses;i++)

{

MPI\_Recv(buff, BUFSIZE, MPI\_CHAR, i, TAG, MPI\_COMM\_PROCESS\_

GROUP,&stat);

printf(“%d: %sÄn“, my\_processId, buff);

}

}

else

{

/\* receive from rank 0: \*/

MPI\_Recv(buff, BUFSIZE, MPI\_CHAR, 0, TAG, MPI\_COMM\_PROCESS\_

GROUP, &stat);

sprintf(idstr, “Processor %d “, my\_processId);

strncat(buff, idstr, BUFSIZE-1);

strncat(buff, “reporting for dutyÄn‘ BUFSIZE-1);

/\* send to rank 0: \*/

MPI\_Send(buff, BUFSIZE, MPI\_CHAR, 0, TAG, MPI\_COM\_PROCESS

\_GROUP);

}

MPI\_Finalize();

return 0;

}

An example of cloud computing using the MPI is described in [119]. An example of MPI use on

EC2 is located at http://rc.fas.harvard.edu/faq/amazonec2.