MPI AND ALL THAT...

-

MPITOOLS

JOHAN MESSCHENDORP, JUNE 2009, PANDA CM, TURIN

Main Motivation

- Learn about the usage of MPI and its possibilities

Resulted in ...

- Light-weighted and easy to use generic tool for "small-scale" job parallelism
  - running on distributed and shared-memory systems
  - platform independent
  - running in user-space (no root-access needed)
  - no need for additional servers
  - easy scalable
- Application: multi-core machines, HPC facility at University (not part of PandaGRID)
- Usage of High Performance Computing standard: Message Passing Interface (MPI)

What is MPI?

What is MPI?

- A computing standard which defines higher-level communication and synchronization methods

What is MPI?

- A computing standard which defines higher-level communication and synchronization methods
- Development since 1993 by a group of parallel computer vendors, computer scientists, and application developers

What is MPI?

- A computing standard which defines higher-level communication and synchronization methods
- Development since 1993 by a group of parallel computer vendors, computer scientists, and application developers
- Bindings for C, C++, Fortran 77/90/95

What is MPI?

- A computing standard which defines higher-level communication and synchronization methods
- Development since 1993 by a group of parallel computer vendors, computer scientists, and application developers
- Bindings for C, C++, Fortran 77/90/95
- Available on wide variety of architectures (super comps, clusters, desktop, ...)

What is MPI?

- A computing standard which defines higher-level communication and synchronization methods
- Development since 1993 by a group of parallel computer vendors, computer scientists, and application developers
- Bindings for C, C++, Fortran 77/90/95
- Available on wide variety of architectures (super comps, clusters, desktop, ...)
- Distributed Memory Paradigm: inter-task communication by message passing

What is MPI?

- A computing standard which defines higher-level communication and synchronization methods
- Development since 1993 by a group of parallel computer vendors, computer scientists, and application developers
- Bindings for C, C++, Fortran 77/90/95
- Available on wide variety of architectures (super comps, clusters, desktop, ...)
- Distributed Memory Paradigm: inter-task communication by message passing
- Several implementations: MPICH(2), LAM, OpenMPI, Vendor MPI

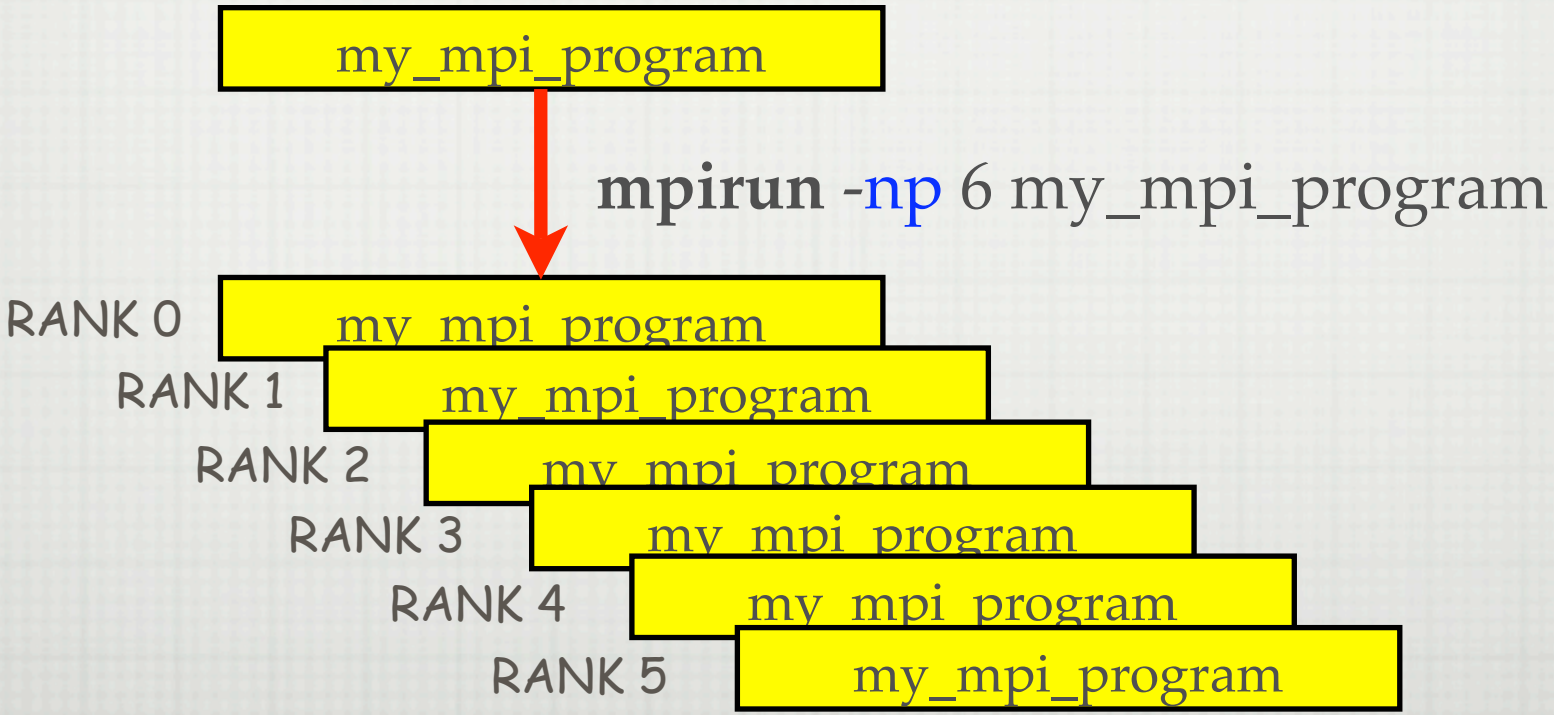
MPI is SPMD

MPI is SPMD

- Parallel programming paradigm -
  SPMD=Single Program Multiple Data
- A copy of the same program runs on each processor
- Same code replicated to each process via rsh, ssh, ...

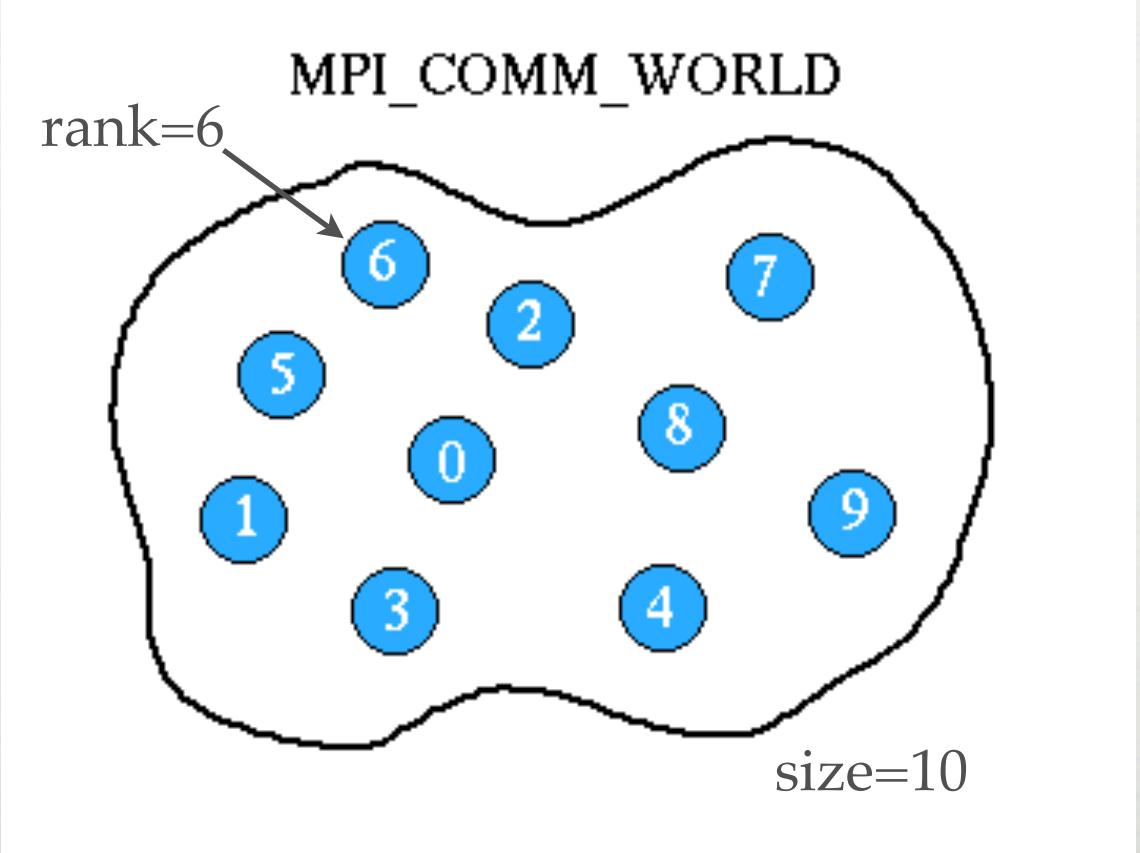
MPI is SPMD

- Parallel programming paradigm - SPMD=Single Program Multiple Data
- A copy of the same program runs on each processor
- Same code replicated to each process via rsh, ssh, ...



MPI is abstract

- The details of the underlying architecture and communication are completely hidden for the developer



A typical MPI skeleton

```
#include "mpi.h"                                    // MPI header file

int main(int argc, char *argv[])                    // Your main program
{
int rank, size;
MPI_Init(&argc, &argv);                             // Initialize MPI

MPI_Comm_rank(MPI_COMM_WORLD, &rank);               // rank = "id" of process
MPI_Comm_size(MPI_COMM_WORLD, &size);               // size = number of processes
....
do_your_work(rank,size);                            // Your calculation, likely
                                                    // rank&size dependent
....
MPI_Finalize();                                     // Free up MPI stuff
}
```

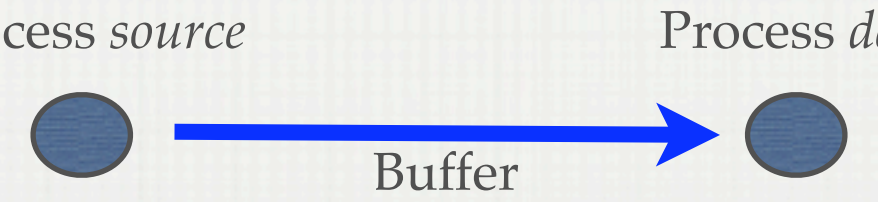
```
mpicc -o my_mpi_program my_mpi_code.cc
```

Communication Tools

Communication Tools

Point-to-point communication

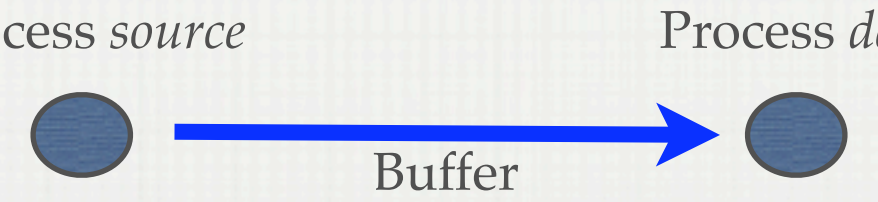
MPI_Send(...)
MPI_Recv(...)



Communication Tools

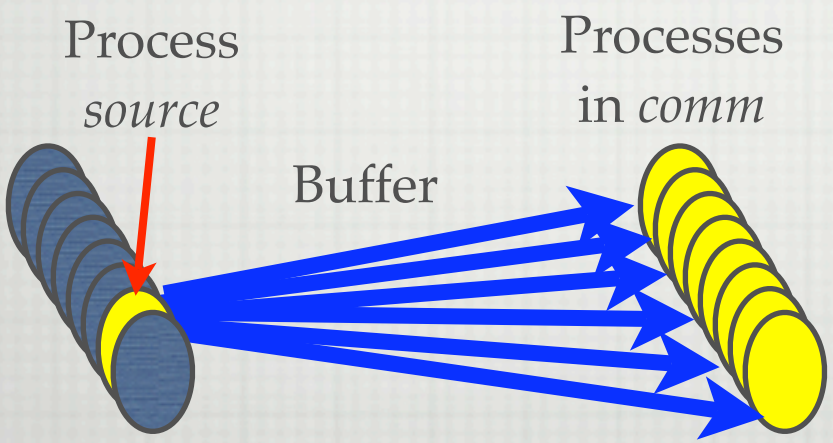
Point-to-point communication

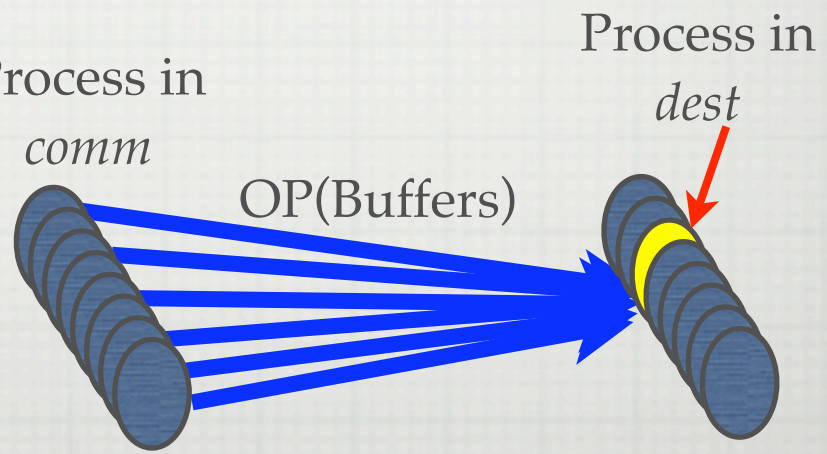
MPI_Send(...)
MPI_Recv(...)



Collective communication

MPI_Bcast(...)
MPI_Reduce(...)





mpiTools

- MPI-based program to execute in parallel user-defined scripts
  - easy configurable!
  - straight-forward input file for the job description
  - lots of features: job synchronization, splitting, nice level, time-out, transparent log file, extensive statistics output, ...
- Boss-Workers Model
  - one boss with many workers running on different nodes or cores
  - pull model: excellent load balancing
  - highly scalable
- Application: infinite IF workers can process or generate data independently
- /pandaroot/trunk/PndTools/mpiTools + panda-wiki.gsi.de
  source, example scripts, documentation, and much more

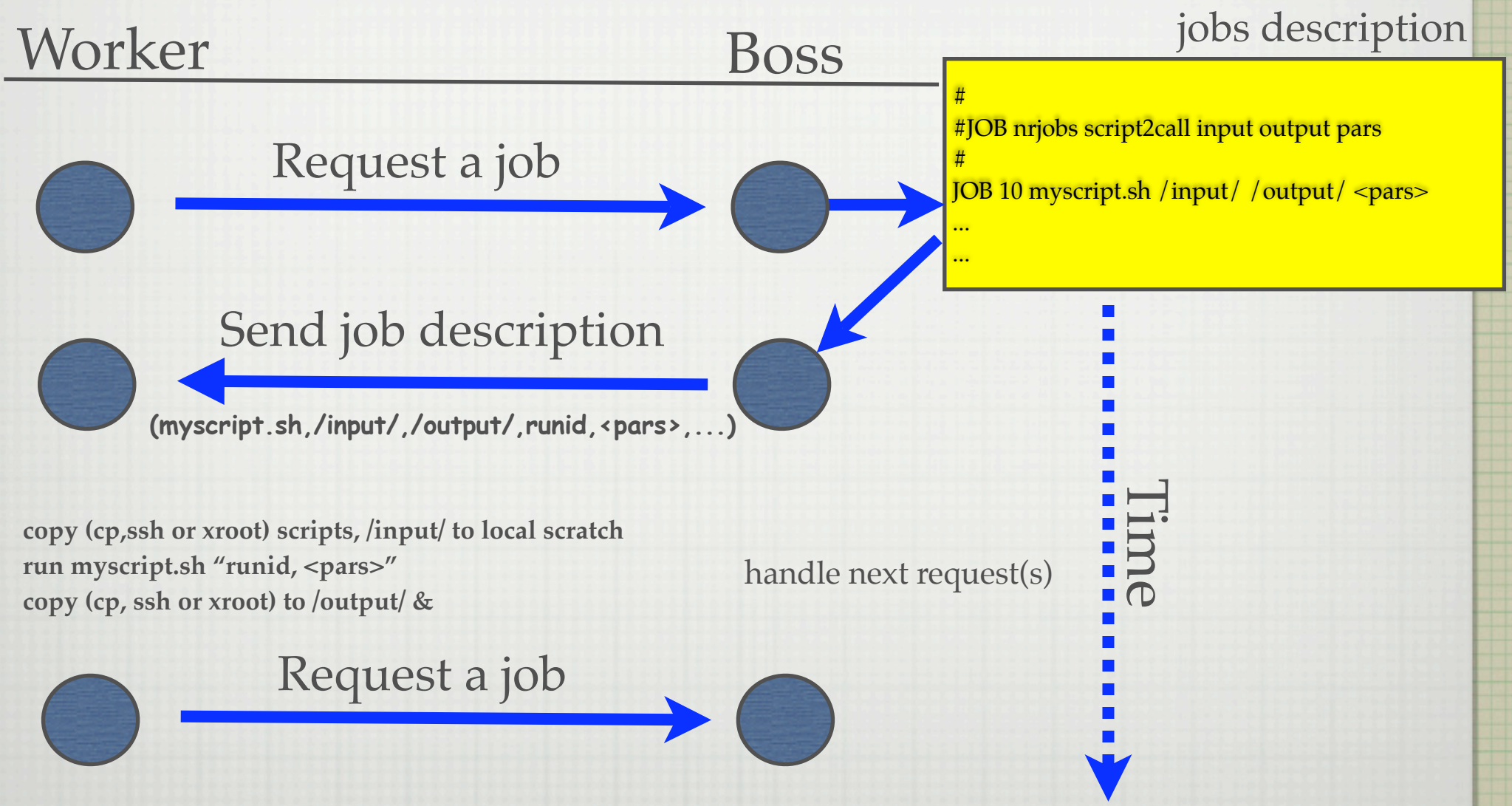
mpiTools-implementation

mpiTools-implementation

jobs description

```
#
#JOB nrjobs script2call input output pars
#
JOB 10 myscript.sh /input/ /output/ <pars>
...
...
```

mpiTools-implementation



Putting things together

jobs description

```
NICE          5
TIMEOUT   3600
SCRATCH    /scratch/

RUNID         0
JOB 100 myscript.sh ~/input/ ~/output/ 1000 e- 5.0
BARRIER

RUNID        1000
JOB 1 merge.sh ~/output/ ~/output2/
```

myjobs.jdl

Putting things together

Example bash script

```
#!/bin/bash
#$1: run id, $2: #evts, $3: ptype, $4: p
# ...
# run macros, etc...
#
root -l -b -q “mymacro.C($1,$2,$3,$4)”
...
if [ -z “$error” ] ; then
  exit 0
fi
exit -1
```

myscript.sh

jobs description

```
NICE        5
TIMEOUT   3600
SCRATCH   /scratch/

RUNID       0
JOB 100 myscript.sh ~/input/ ~/output/ 1000 e- 5.0
BARRIER

RUNID       1000
JOB 1 merge.sh ~/output/ ~/output2/
```

myjobs.jdl

Putting things together

Example bash script

```
#!/bin/bash
#$1: run id, $2: #evts, $3: ptype, $4: p
# ...
# run macros, etc...
#
root -l -b -q "mymacro.C($1,$2,$3,$4)"
...
if [ -z "$error" ] ; then
  exit 0
fi
exit -1
```

myscript.sh

jobs description

```
NICE        5
TIMEOUT   3600
SCRATCH   /scratch/

RUNID       0
JOB 100 myscript.sh ~/input/ ~/output/ 1000 e- 5.0
BARRIER

RUNID       1000
JOB 1 merge.sh ~/output/ ~/output2/
```

myjobs.jdl

list of machines

```
kvit14.kvi.nl
kvit15.kvi.nl
kvit16.kvi.nl
kvip81.kvi.nl
```

mynodes.list

Putting things together

Example bash script

```
#!/bin/bash
#$1: run id, $2: #evts, $3: ptype, $4: p
# ...
# run macros, etc...
#
root -l -b -q "mymacro.C($1,$2,$3,$4)"
...
if [ -z "$error" ] ; then
  exit 0
fi
exit -1
```

myscript.sh

jobs description

```
NICE        5
TIMEOUT   3600
SCRATCH   /scratch/

RUNID       0
JOB 100 myscript.sh ~/input/ ~/output/ 1000 e- 5.0
BARRIER

RUNID       1000
JOB 1 merge.sh ~/output/ ~/output2/
```

myjobs.jdl

list of machines

```
kvit14.kvi.nl
kvit15.kvi.nl
kvit16.kvi.nl
kvip81.kvi.nl
```

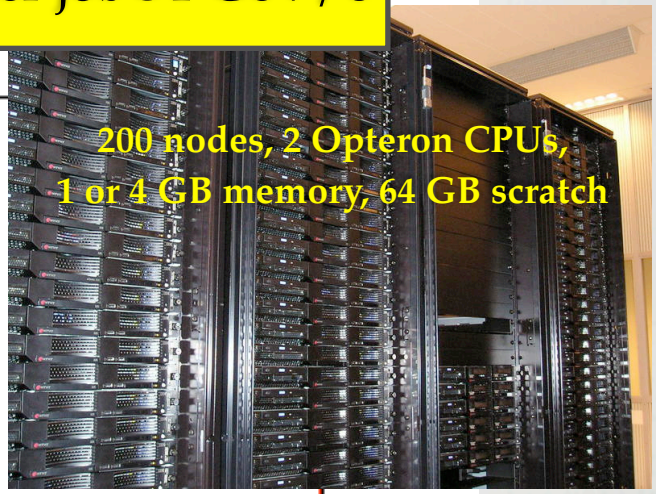
mynodes.list

execution

mpirun -np 17 -machinefile *mynodes.list* boss_worker_mpi -j *myjobs.jdl*

A benchmark example...

Panda EMC :40 jobs, 500 photon events per job@1 GeV/c



qsub run_emc_mpi.job **-l** nodes=*1,2,4,5,10*:ppn=*1,2*

Summary

- Parallelism using MPI
  - high-level tool suited for many HPC infrastructures
  - in HEP not well known, although some activities in Geant4 are ongoing
  - MPI is relatively easy to learn
- "Play" project: mpiTools
  - very easy to use generic tool for job parallelisation
  - only depends upon MPI, trivially installed
  - more people interested? include in external packages?
  - does not replace central PandaGRID (which is much more advanced)!!!
  - does not provide interactive parallelisation, such as PROOF