GPPD

EURO-PVM/MPI’06
Bonn, Germany

Improving the Dynamic Creation of Processes in MPI-2

Márcia C. Cera, Guilherme P. Pezzi, Elton Mathias, Nicolas Maillard and Philippe O. A. Navaux

Work sponsored by CNPq, CAPES and HP Brazil

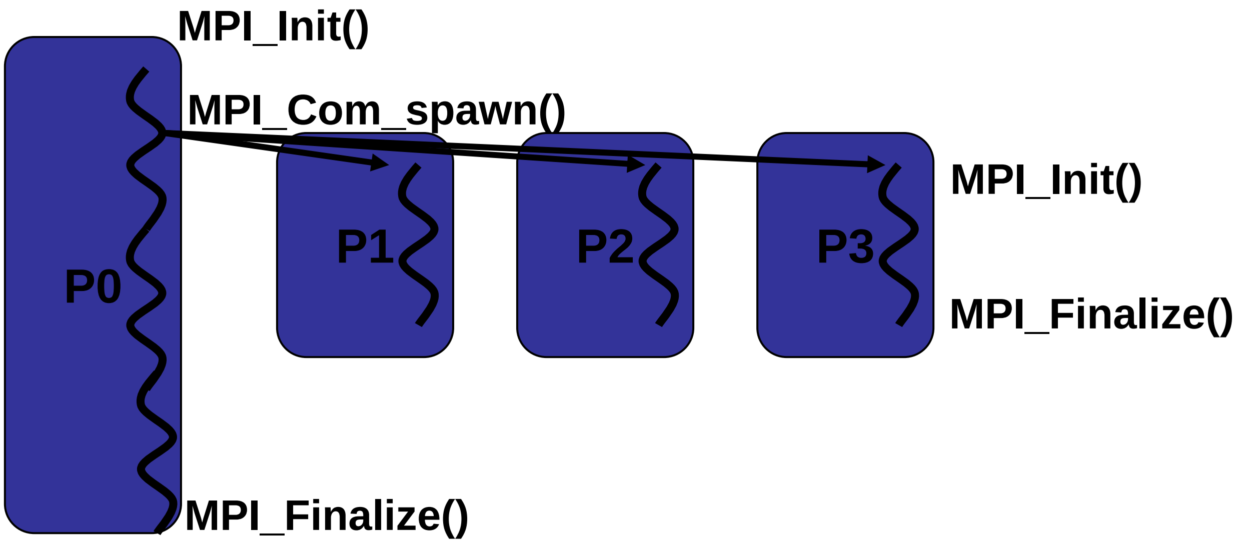
GPPD

MPI, MPI-2, ...

- **Message Passing Interface** is the de-facto standard for Cluster Computing
  - inherited from PVM;
  - MPI 1.2 does not provide the dynamic creation/management of processes
- **MPI-2**: has been defined in 1998.
  - Parallel I/O, RMA, etc... ;
  - Dynamic creation of processes (MPI_Comm_spawn)
- Recent implementations of MPI-2:
  - LAM: since the start of the 2000 years.
    - Lamgrow/lamshrink
  - MPI-CH: Jan., 2005.
  - HP-MPI: Dec., 2005.
- Towards a MPI for Grids ?
  - MPI-CH-G2, Mpi-CH/Madeleine: supports heterogeneity, but not the dynamicity;
  - Checkpoint/Restart in MPI-CHv2 and LAM (/BLCR)
    - builds upon MPI 1.2;
  - Open-MPI: fusion between MPI-FT and LAM.
    - Fully functionnal?

MPI_Comm_spawn()

- MPI_Comm_spawn(cmd, argv, argc, nbprocs, info, root, comm_root, &intercomm, err);



MPI_Comm_spawn()

- MPI_Comm_spawn(cmd, argv, argc, nbprocs, info, root, comm_root, &intercomm, err);
  - cmd: name of the MPI executable.
  - argv, argc: command line arguments to be passed to ‘cmd’.
  - nbprocs: number of MPI processes to be created.

MPI_Comm_spawn()

- MPI_Comm_spawn(cmd, argv, argc, nbprocs, info, root, comm_root, &intercomm, err);
  - info :backdoor left to the implementation.
    - MPI-2 defines the dataype ‘MPI_Info’
    - Ex. of use:

MPI_Info_set(info, “lam_spawn_sched_round_robin”, *rank*)

- Starts a Round-Robin from proc number ‘rank’
- (Round-Robin is the default)

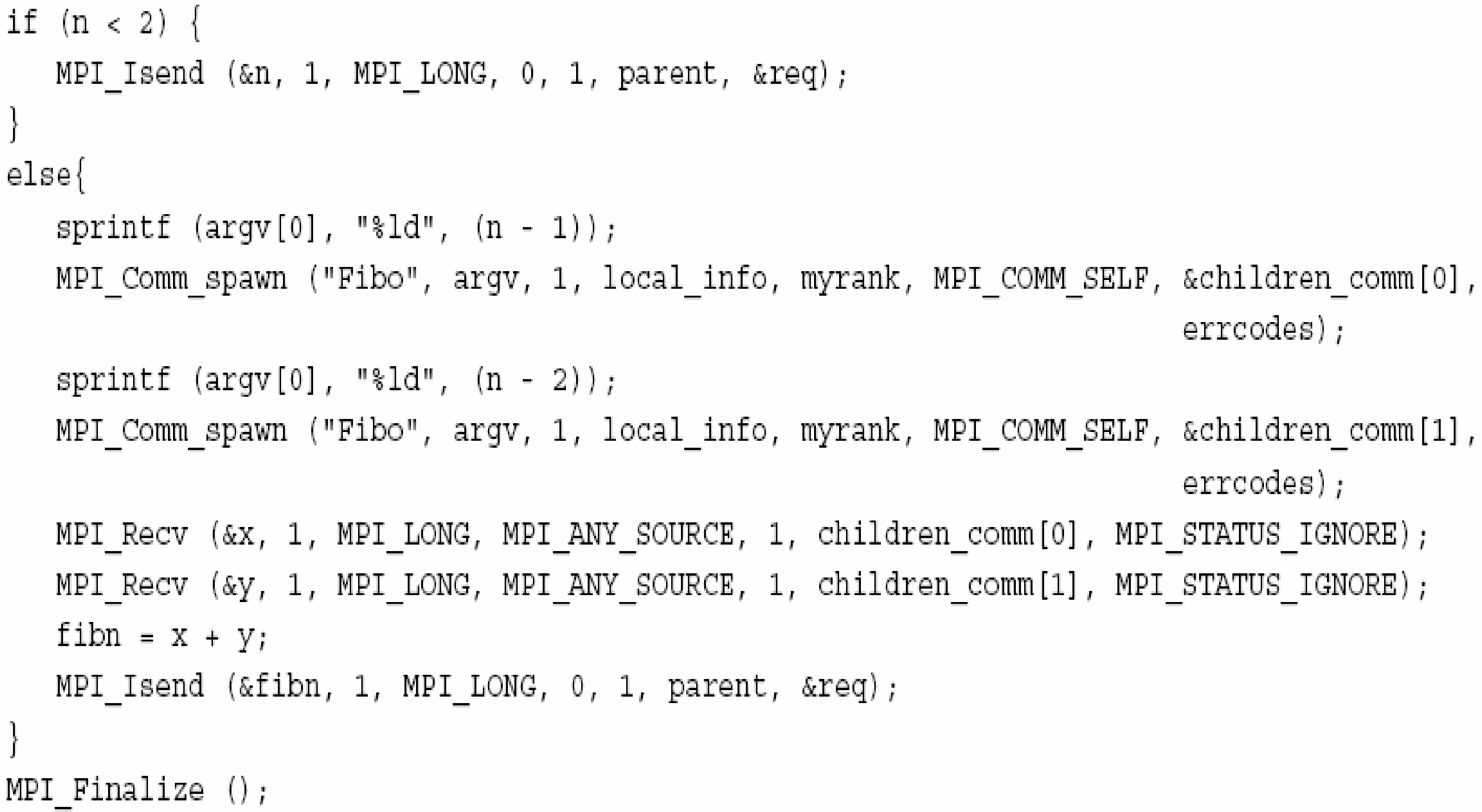
MPI_Comm_spawn()

- MPI_Comm_spawn(cmd, argv, argc, nbprocs, info, root, comm_root, &intercomm, err);
  - root : rank of the father process.
  - comm_root : intra-communicator of the parent process (MPI_Communicator).
  - intercomm : inter-communicator that enables the communication Send/Recv bwteen the processes in ‘comm_root’ and those of the children’s MPI_Comm_world.

Communication between the Processes

- The parent uses the inter-communicator to send/recv messages with its children.
- The children have to call **MPI_Get_parent**() to obtain their parent’s communicator.
  - If the return is NULL, the children have been “mpirun” directly, and not MPI_Comm_spawned.
  - The parent has rank 0 in this communicator.

Example: Fibonacci with MPI-2

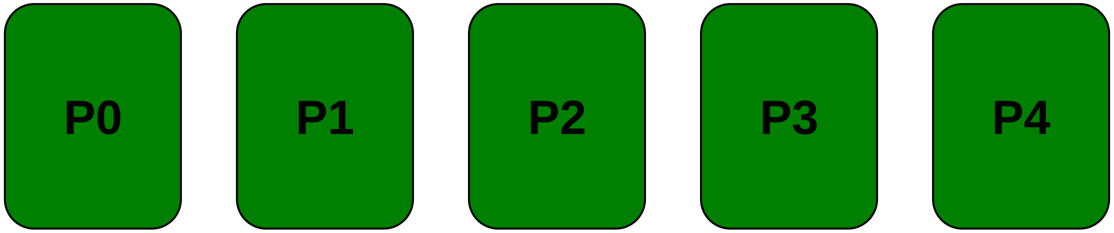


```
if (n < 2) {
   MPI_Isend (&n, 1, MPI_LONG, 0, 1, parent, &req);
}
else{
   sprintf (argv[0], "%ld", (n - 1));
   MPI_Comm_spawn ("Fibo", argv, 1, local_info, myrank, MPI_COMM_SELF, &children_comm[0],
                                                                       errcodes);
   sprintf (argv[0], "%ld", (n - 2));
   MPI_Comm_spawn ("Fibo", argv, 1, local_info, myrank, MPI_COMM_SELF, &children_comm[1],
                                                                       errcodes);
   MPI_Recv (&x, 1, MPI_LONG, MPI_ANY_SOURCE, 1, children_comm[0], MPI_STATUS_IGNORE);
   MPI_Recv (&y, 1, MPI_LONG, MPI_ANY_SOURCE, 1, children_comm[1], MPI_STATUS_IGNORE);
   fibn = x + y;
   MPI_Isend (&fibn, 1, MPI_LONG, 0, 1, parent, &req);
}
MPI_Finalize ();
```

Two Main Issues with Dynamic Processes

1. How to be efficient in the communication between parent and children?
   - If anybody want to communicate with everybody, the comm have to be merged (MPI_Comm_merge).
   - One should hierarquize the processes
     - -> Divide & Conquer.

2. How does MPI_Comm_spawn allocate the processes ?
   - Default: Round-Robin from a fixed rank (0).
   - Problem if a series a Spawns are repeated.
   - Problem when more than one process perform spawns in parallel...

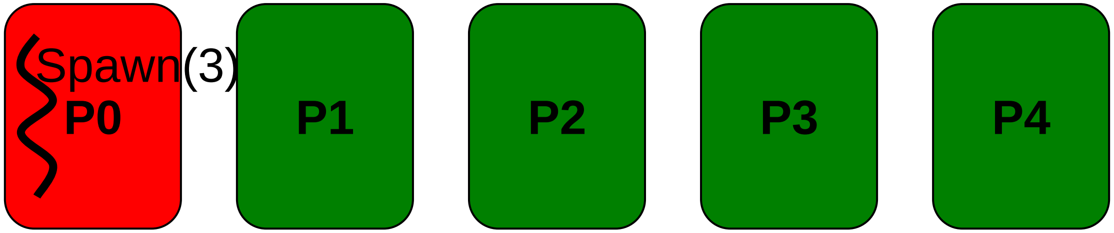


P0 P1 P2 P3 P4

Two Main Issues with Dynamic Processes

1. How to be efficient in the communication between parent and children?
   - If anybody want to communicate with everybody, the comm have to be merged (MPI_Comm_merge).
   - One should hierarquize the processes
     - -> Divide & Conquer.

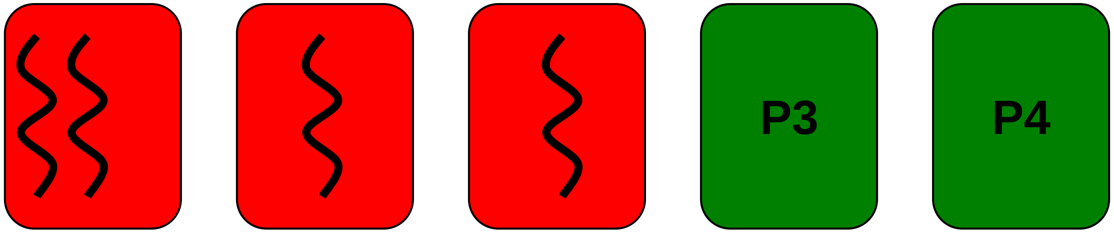
2. How does MPI_Comm_spawn allocate the processes ?
   - Default: Round-Robin from a fixed rank (0).
   - Problem if a series a Spawns are repeated.
   - Problem when more than one process perform spawns in parallel...



Two Main Issues with Dynamic Processes

1. How to be efficient in the communication between parent and children?
   - If anybody want to communicate with everybody, the comm have to be merged (MPI_Comm_merge).
   - One should hierarquize the processes
     - -> Divide & Conquer.

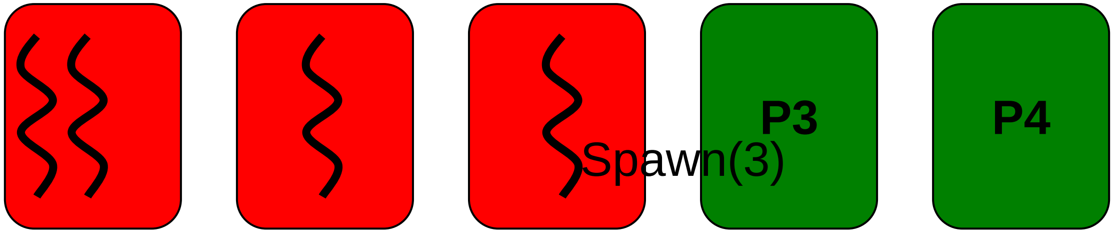
2. How does MPI_Comm_spawn allocate the processes ?
   - Default: Round-Robin from a fixed rank (0).
   - Problem if a series a Spawns are repeated.
   - Problem when more than one process perform spawns in parallel...



Two Main Issues with Dynamic Processes

1. How to be efficient in the communication between parent and children?
   - If anybody want to communicate with everybody, the comm have to be merged (MPI_Comm_merge).
   - One should hierarquize the processes
     - -> Divide & Conquer.

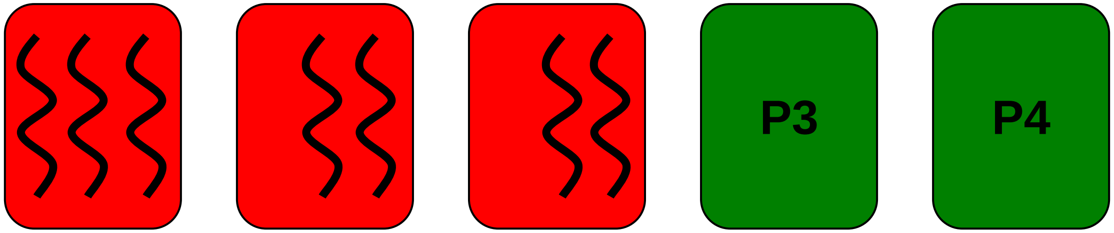
2. How does MPI_Comm_spawn allocate the processes ?
   - Default: Round-Robin from a fixed rank (0).
   - Problem if a series a Spawns are repeated.
   - Problem when more than one process perform spawns in parallel...



Two Main Issues with Dynamic Processes

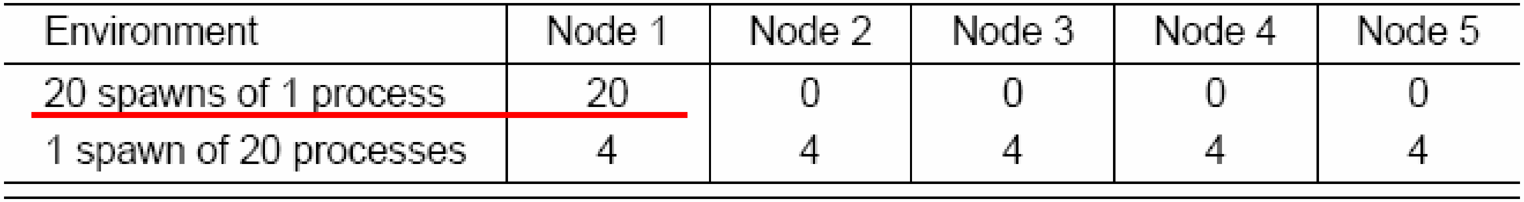
1. How to be efficient in the communication between parent and children?
   - If anybody want to communicate with everybody, the comm have to be merged (MPI_Comm_merge).
   - One should hierarquize the processes
     - -> Divide & Conquer.

2. How does MPI_Comm_spawn allocate the processes ?
   - Default: Round-Robin from a fixed rank (0).
   - Problem if a series a Spawns are repeated.
   - Problem when more than one process perform spawns in parallel...



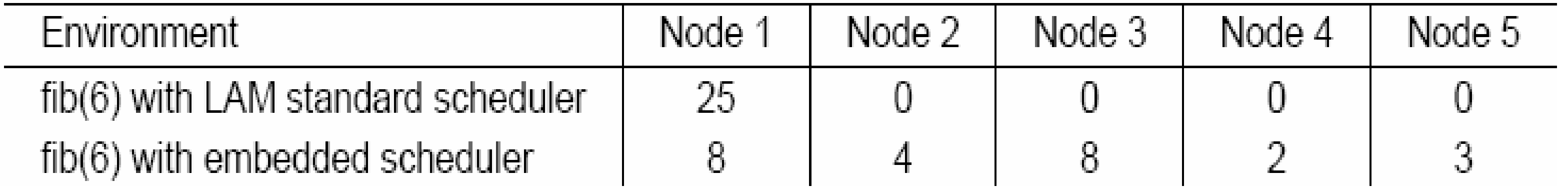
Native Allocation of Processes

- The native mechanism may allocate all processes to one processor !



Environment	Node 1	Node 2	Node 3	Node 4	Node 5
20 spawns of 1 process	20	0	0	0	0
1 spawn of 20 processes	4	4	4	4	4

- Improvement with one variable that controls where to launch the processes.

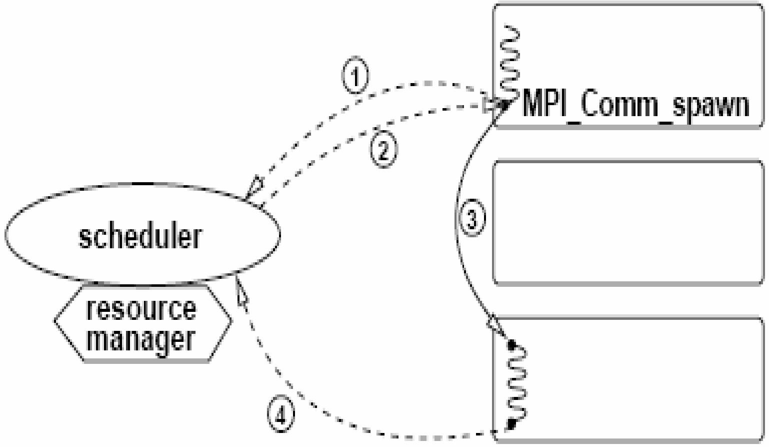


Environment	Node 1	Node 2	Node 3	Node 4	Node 5
fib(6) with LAM standard scheduler	25	0	0	0	0
fib(6) with embedded scheduler	8	4	8	2	3

Solution: a Centralized Scheduler

- Simple idea:
  - A daemon is run together with the MPI application to centralize the allocation decision.
  - MPI_Comm_spawn et MPI_Finalize() are redefined to notify the daemon at process creation/finalization.
- The scheduler daemon:
  - Can manage the task graph of the application;
  - Can decide about the location of the spawned processes, with a Round-Robin algorithm;
    - Centralized R.R.
  - Can monitor /proc and base the decision about the load of each node...
  - Etc...
- Simple tests have been performed with a prototype
  - To be included in a LAM distribution!

Implementation of the Scheduler



1. MPI_Comm_spawn/ Notification of the creation of a process
2. Scheduling decision
3. Physical creation
4. Notification of the completion of the process

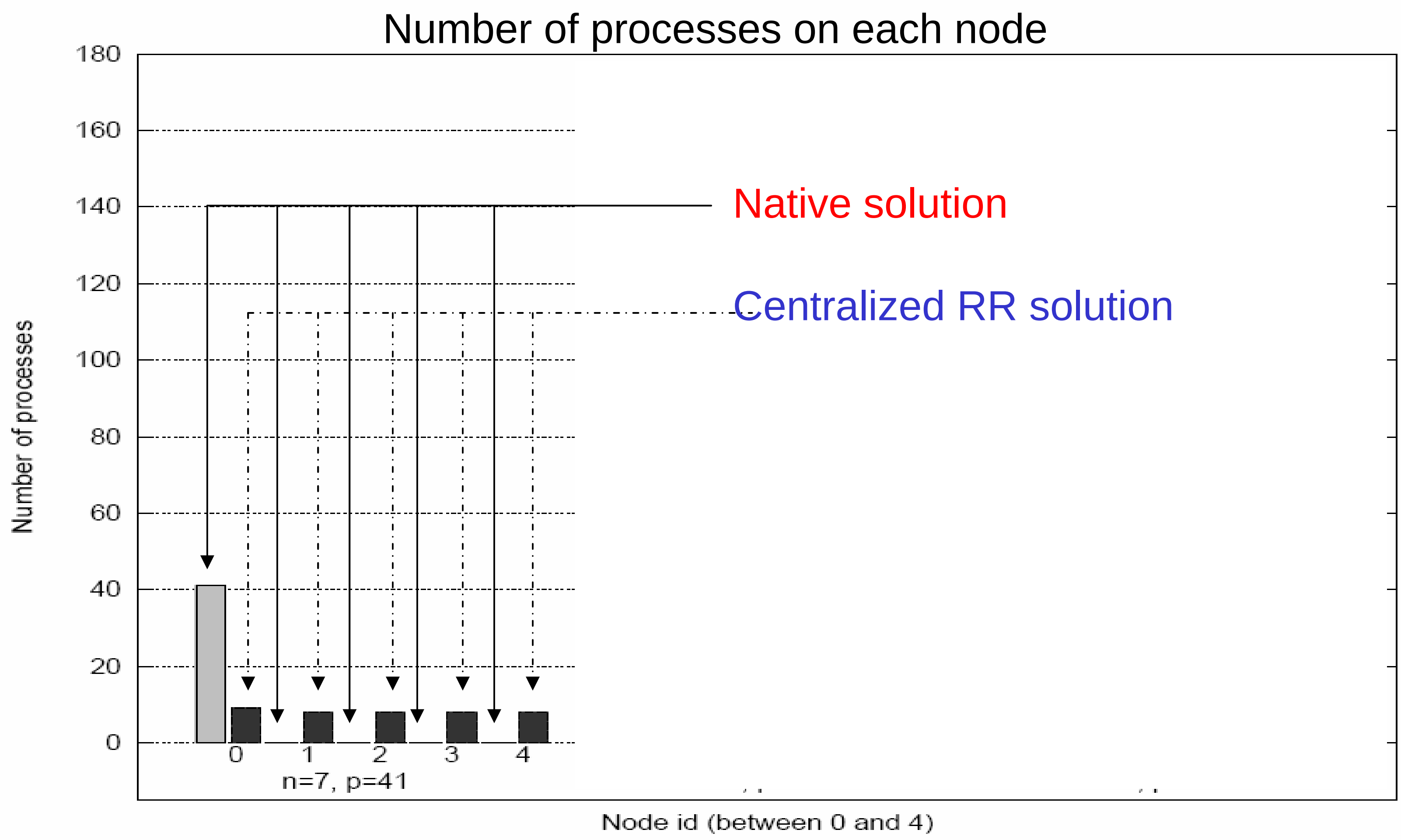
Three Experiments

1. Application of the centralized RR to the computation of Fibo(7), Fibo(10) and Fibo(13).
   - This benchmark creates many processes of very short duration
   - Balancing the processes.

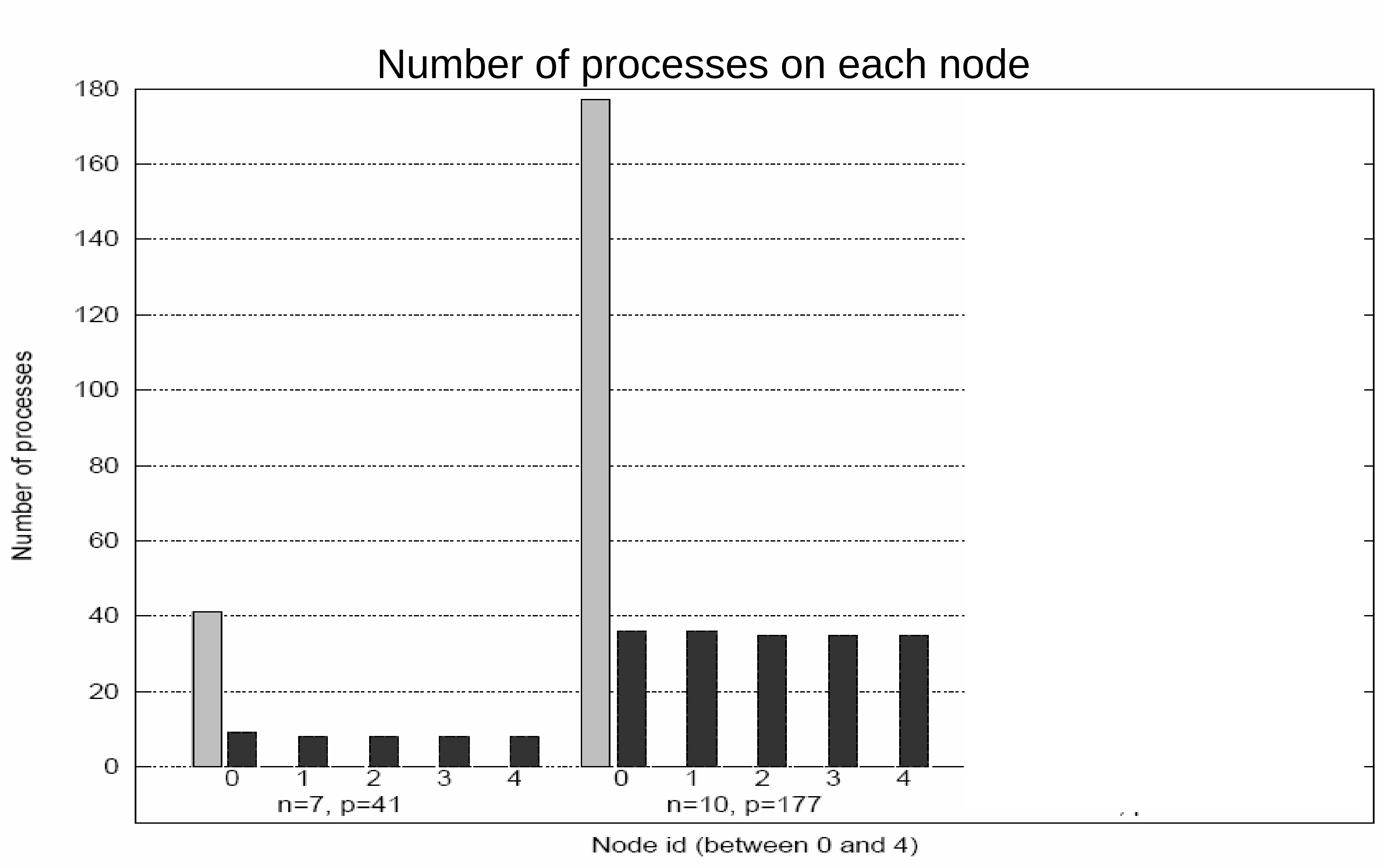
2. Recursive computation of the prime numbers in the interval [1...N], with measure of the load
   - Irregular run-time
   - Improving the computation time.

3. Round-Robin with a dynamically increasing number of nodes (lamgrow)
   - Dynamic creation of processes and resources
   - Load balancing with dynamic resources.

1 - Fibonacci – Native Solution *vs.* Centralized Round-Robin Allocation

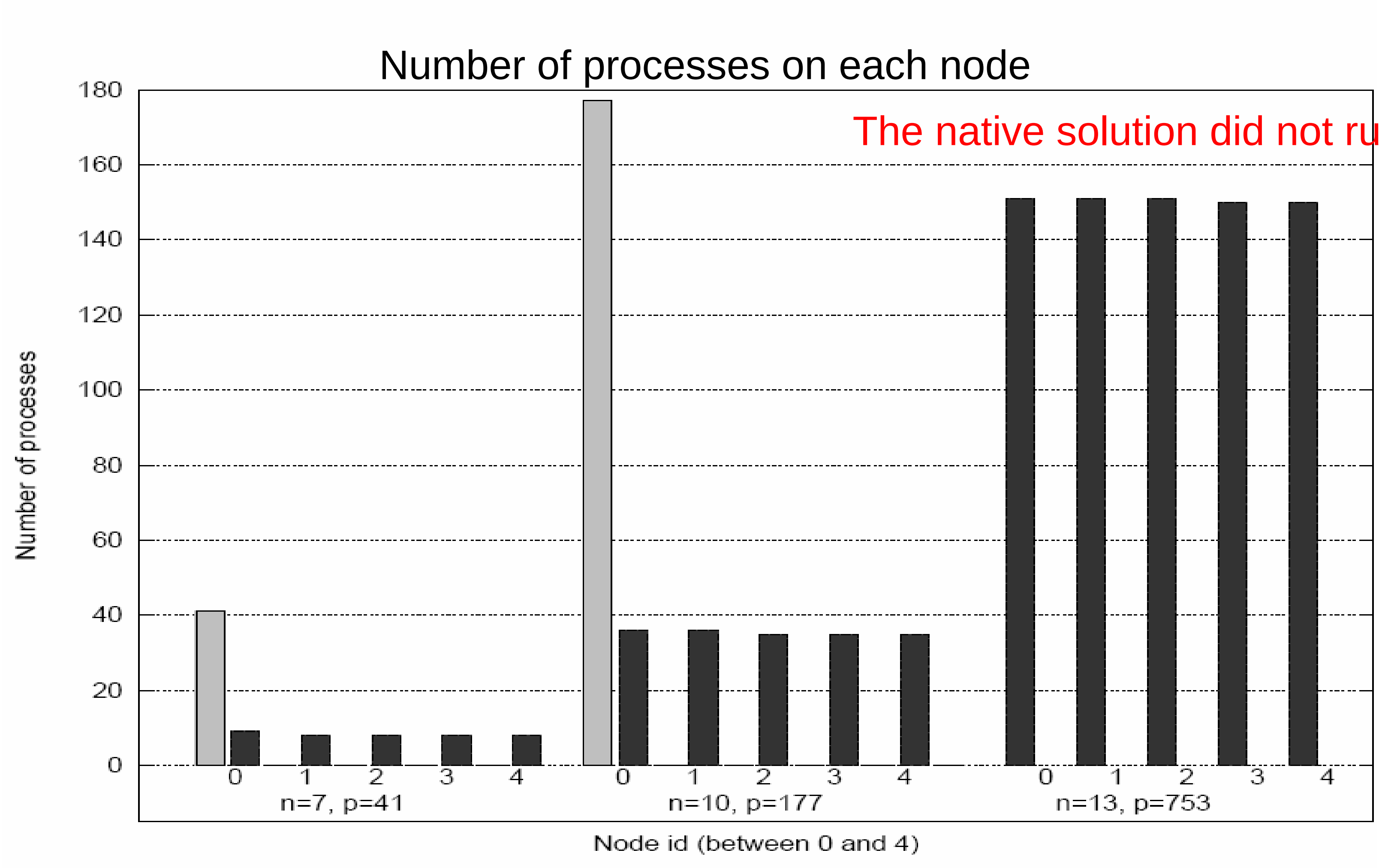


1 - Fibonacci – Native Solution *vs.* Centralized Round-Robin Allocation



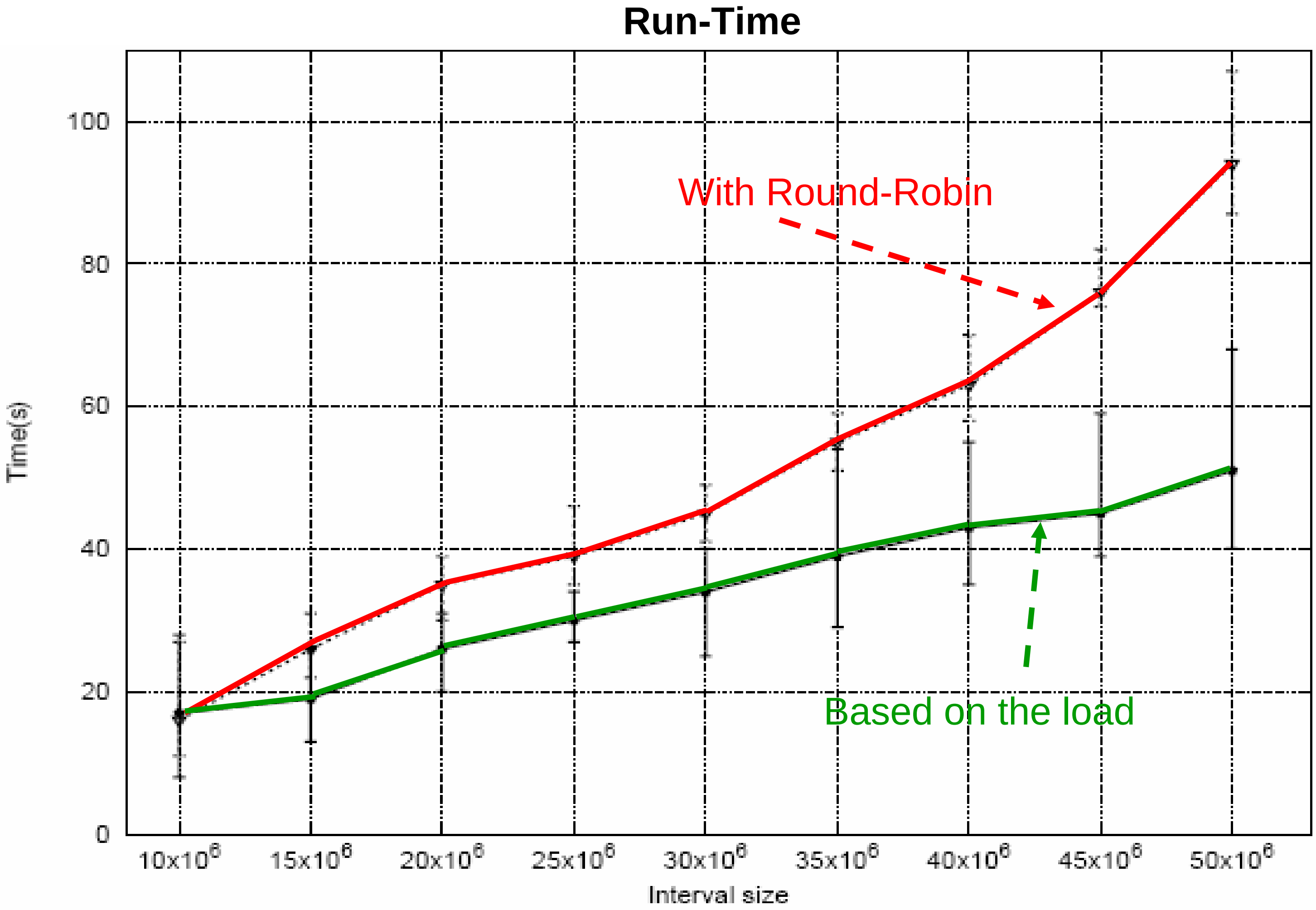
GPPD

1 - Fibonacci – Native Solution *vs.* Centralized Round-Robin Allocation

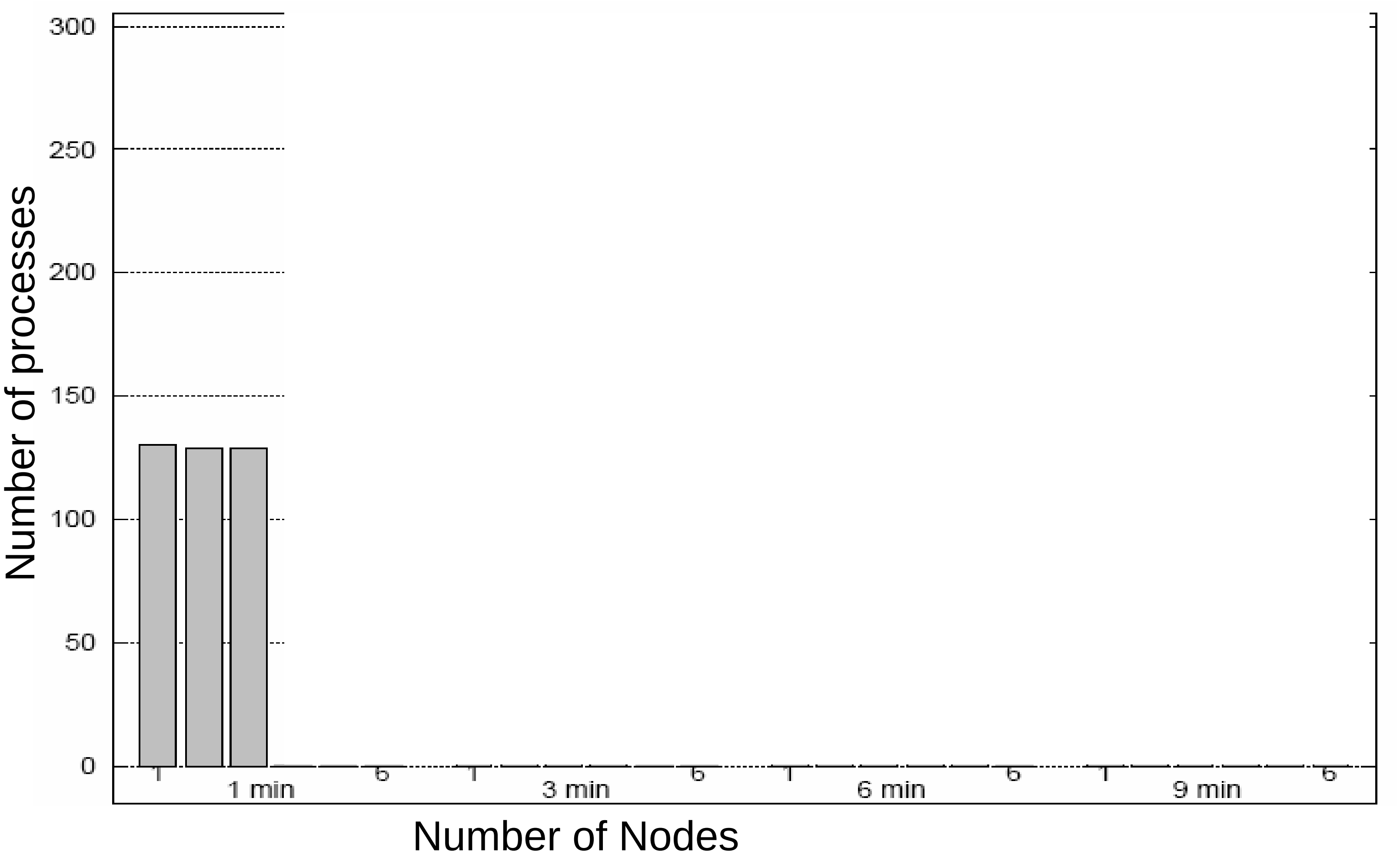


The native solution did not run!

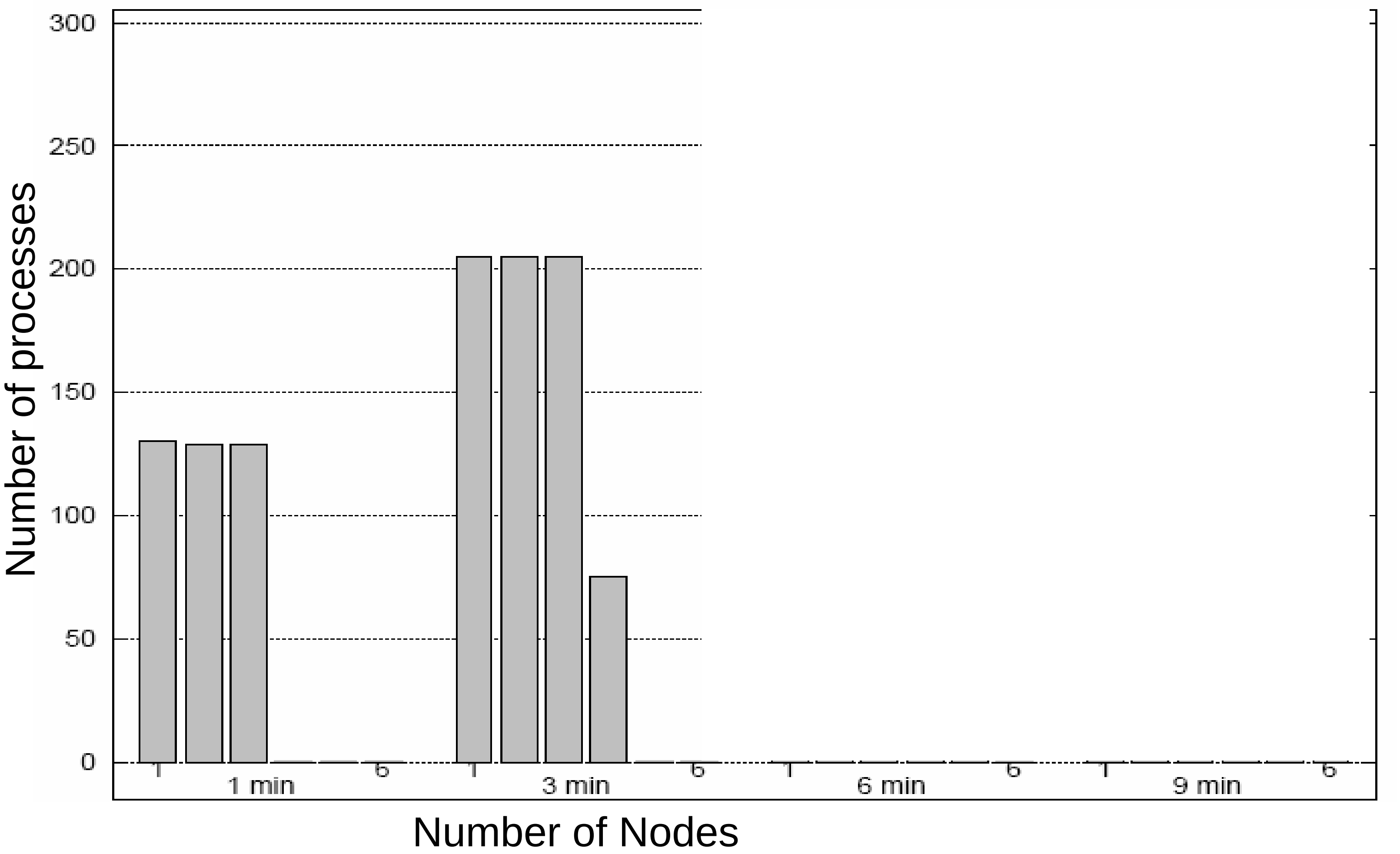
2 - Prime Numbers – Balancing the Load of an Irregular Application



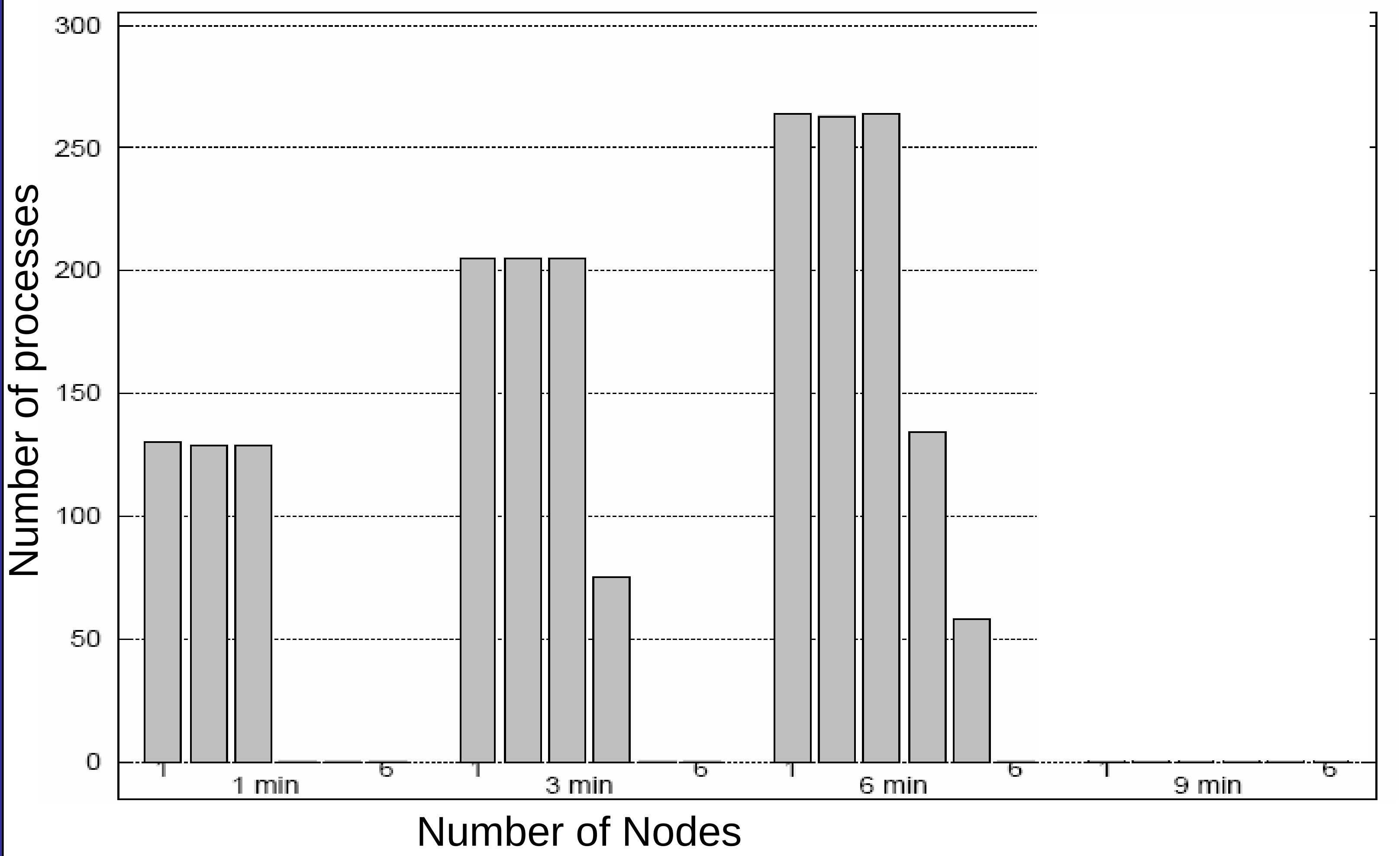
3 - Fibonacci – Allocation of Processes with lamgrow



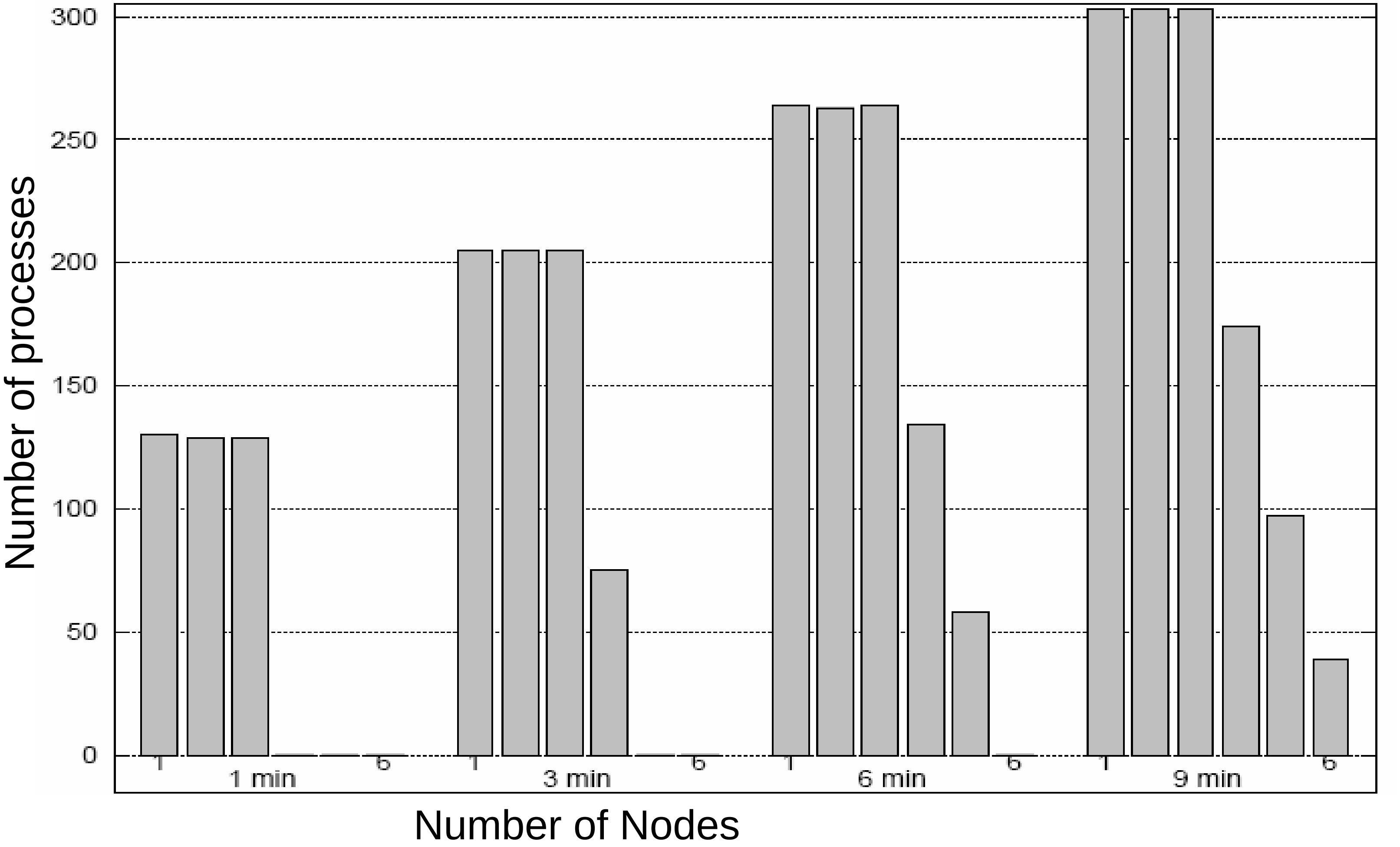
3 - Fibonacci – Allocation of Processes with lamgrow



3 - Fibonacci – Allocation of Processes with lamgrow



3 - Fibonacci – Allocation of Processes with lamgrow



Conclusions

- Dynamic creation of processes with MPI-2 is okay.
  - Interesting for coarse-grained applications
  - One needs to find a way to manage efficiently the communication
    - Parent/children
  - LAM enables the dynamic integration of new resources (lamgrow)
- LAM's native allocation of Spawned processes is weak.
  - Well, it respects the norm !...
  - A simple, centralized solution leads to clear improvements.
  - Why not providing such add-ons in the distributions?
- Natural idea: distribute the scheduler
  - Workstealing?

Limitations & Next Steps

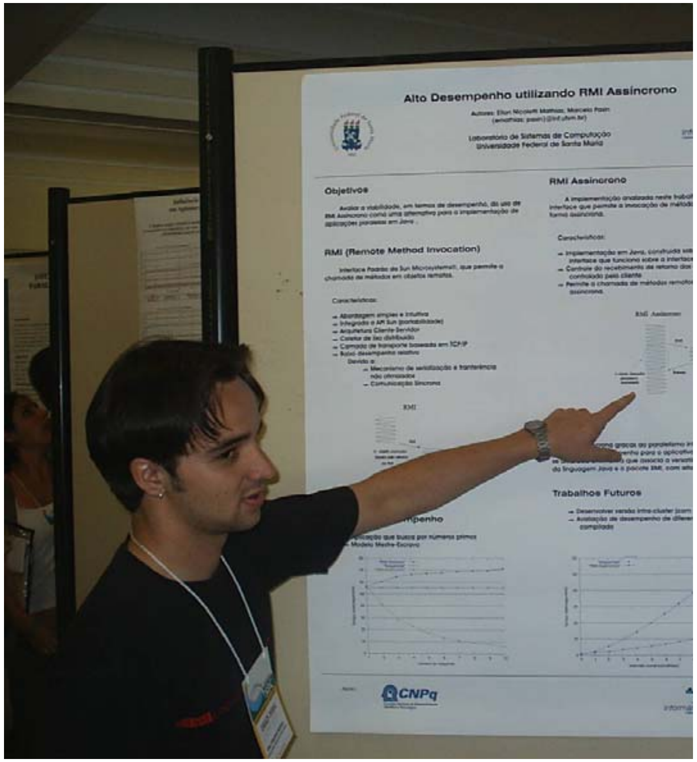
- Limited to LAM-MPI
  - Yet, easy to port!
    - The only Lam-dependent part is the integration into the MPI_Comm_spawn implementation.
- Lamgrow is fine... What about lamshrink ?
  - One needs some checkpoint/restart mechanism...
  - Open-MPI could provide it ?
- In a view to working with coarse-grained applications, the benchmarks are somewhat limited...
  - Current work includes “real-world” applications.
- Using such mechanisms in Grids?
  - Does MPI-2 run on the Grid ?
  - Globus enabled MPI distribution does not seem to focus MPI-2...

UFRGS
UNIVERSIDADE FEDERAL
DO RIO GRANDE DO SUL

GPPD

Any return will be welcome!

nicolas@inf.ufrgs.br


Alto Desempenho utilizando RMI Assíncrono
Laboratório de Sistemas de Computação
Universidade Federal de Santa Maria
Objetivos
RMI (Remote Method Invocation)
RMI Assíncrono
Trabalhos Futuros
CNPq