Legio: Fault Resiliency for Embarrassingly Parallel MPI Applications

Roberto Rocco

roberto2.rocco@mail.polimi.it

CSE Track

HONOURS PROGRAMME HP-SR in Information Technology

Overview

- Problem definition
- Previous solutions
- Legio framework
- Hierarchical Legio evolution
- Experimental campaign
- Conclusions & Future work

High Performance Computing

- **Field of computer architectures aimed at reaching the highest computation capabilities.**
- **Performance is core, no trade-offs with power consumption, space, costs.**
- **Continuous evolution.**

Most performing architectures

11/2000

Rank	System	Cores	TFLOP/s
1	ASCI White, United States	8,192	4.9380
2	ASCI Red, United States	9,632	2.3790
3	ASCI Blue-Pacific SST, United States	5,808	2.1440

11/2020

Rank	System	Cores	TFLOP/s
1	Supercomputer Fugaku, Japan	7,630,848	442,010.0
2	Summit, United States	2,414,592	148,600.0
3	Sierra, United States	1,572,480	94,640.0

- **Growth in the number of cores of ≈ 1000 factor;**
- **Growth in performance of ≈ 100000 factor.**

Impact of faults in the HPC environment

- **How much frequent are faults in HPC systems?**

- **Analyze reliability (R(t)): probability that the system will operate correctly up until time t.**

- **For simplicity, let's assume exponential distributions for each core, with Mean Time To Failure (MTTF) equal to 1 century.**

$$R(t):\ P(no\ faults\ in\ [0,t])\ = e^{-\frac{1}{MTTF}t}$$

Impact of faults in the HPC environment

- The probability that there are no faults in the system until time t can be computed as follows:

$$R_{sys}(t) = \left(R(t)\right)^n = e^{\frac{-n}{MTTF}t} = e^{\frac{-1}{\left(\frac{MTTF}{n}\right)}t}$$

where n is the number of cores of the system.

- The MTTF of the system is equal to the one of the core divided by the number of cores.

Evolution of faults in HPC

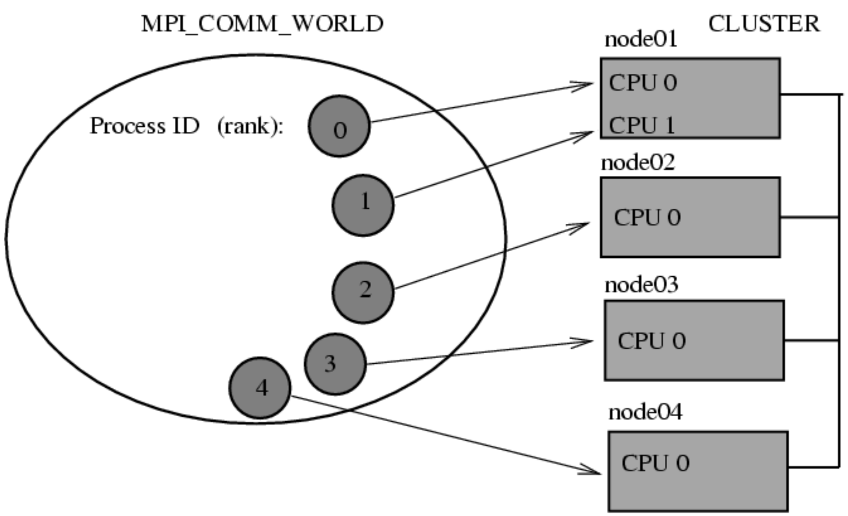
- **MTTF on ASCI Red = 876000h / 9632 cores ≈ 91h**
- **MTTF on Summit = 876000h / 2,414,592 cores ≈ 21m**
- **An example 48h execution**
  - **would need on average 1.69 executions on ASCI Red**
  - **would need on average $3{,}6 * 10^{59}$ executions on Summit.**

MPI’s countermeasures

- Message Passing Interface (MPI), the de-facto standard for intra-process communication.
- MPI provides efficient (low overhead) communication.
- Upon fault the status of the execution is undefined.
- Many efforts developed solutions to this problem.
- The User Level Fault Mitigation (ULFM) library is the most prominent one.

MPI basics

- **MPI communication is based on communicators.**
- **Each process within a communicator has a rank.**
- **Ranks go from 0 to size-1.**



ULFM basics

- **ULFM introduces new functionalities able to bring back the execution to a consistent state after fault.**

- **It introduces functions able to:**
  - ➢ **Get which processes failed;**
  - ➢ **Propagate errors on the network;**
  - ➢ **Eliminate faulty processes from the network;**
  - ➢ **Let all the non-failed processes agree on a value.**

- **May be integrated in future MPI versions.**

All-in-one frameworks drawbacks

- **Many frameworks combined ULFM with Checkpoint/Restart (C/R) to produce all-in-one frameworks for fault tolerance (Fenix, CPPC, CRAFT, LFLR).**

- **But…**
  - ➢ **The integration needs code changes in the application;**
  - ➢ **C/R overhead can be non-negligible.**

Legio strong points

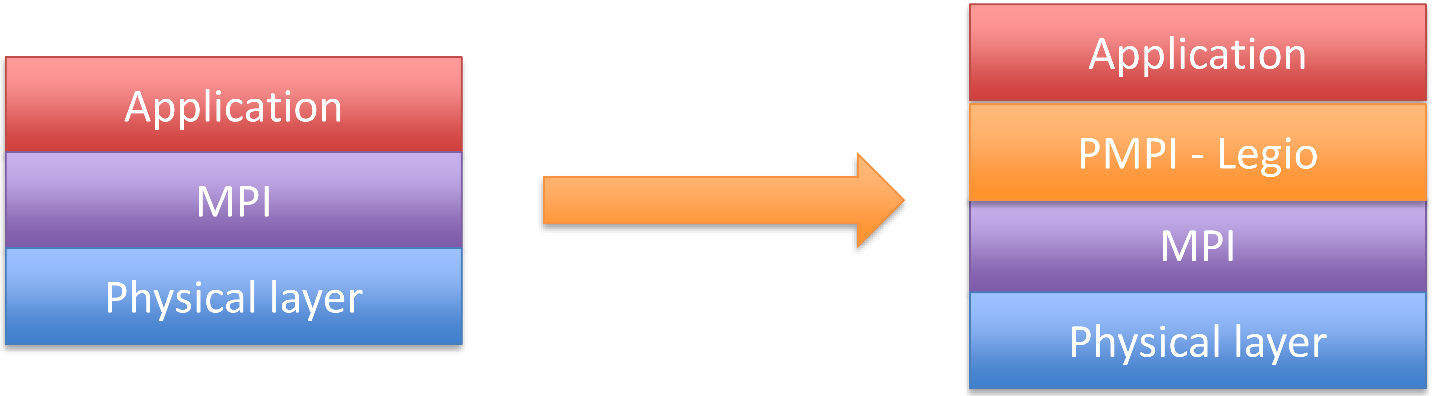
- **Transparency: no code changes needed in the application.**

- **Fault resiliency: execution continues only with the non-failed processes.**

- **Embarrassingly parallel applications: they solve a problem that is intrinsically parallel, little dependency, simple communication structure.**

Legio core idea

- **Transparently substitute the MPI structures used by the application with others handled by the framework.**
- **Upon fault, the structures handled by the framework are substituted, and the fault is masked to the application.**
- **Seamless integration using PMPI.**



Legio details

- **Ranks in the communicator handled by the framework may be different from the ones in the application communicator.**

Original rank	0	1	2	3	4	5	6
Rank in substitute	0	1	2	3	4	5	6
Rank in substitute after 3 failed	0	1	2	X	3	4	5
Rank in substitute after 6 failed	0	1	2	X	3	4	X
Rank in substitute after 0 failed	X	0	1	X	2	3	X

Legio details

- Multiple structures can be used, each one must have its own substitute.

- What to do in case of fault?
- ➢ Repair and repeat the operation.

- File and windows are not supported by ULFM.

Legio details

- **Some operations change behaviour when using the structures handled by the framework.**
  - **Like scatter and gather.**

Original rank	0	1	2	3	4	5	6
Rank in substitute after 3 failed	0	1	2	X	3	4	5
Scatter result if no faults	A	B	C	D	E	F	G
Scatter result using substitute	A	B	C	X	D	E	F
Scatter result correct	A	B	C	X	E	F	G

Legio weak points

- The framework transparently introduces fault resiliency in an embarrassingly parallel application.

- But…
  - The repair procedure needs the participation of all the processes;
  - The shrink operation, on which the repair procedure bases, should scale worse than linearly.

Hierarchical Legio principles

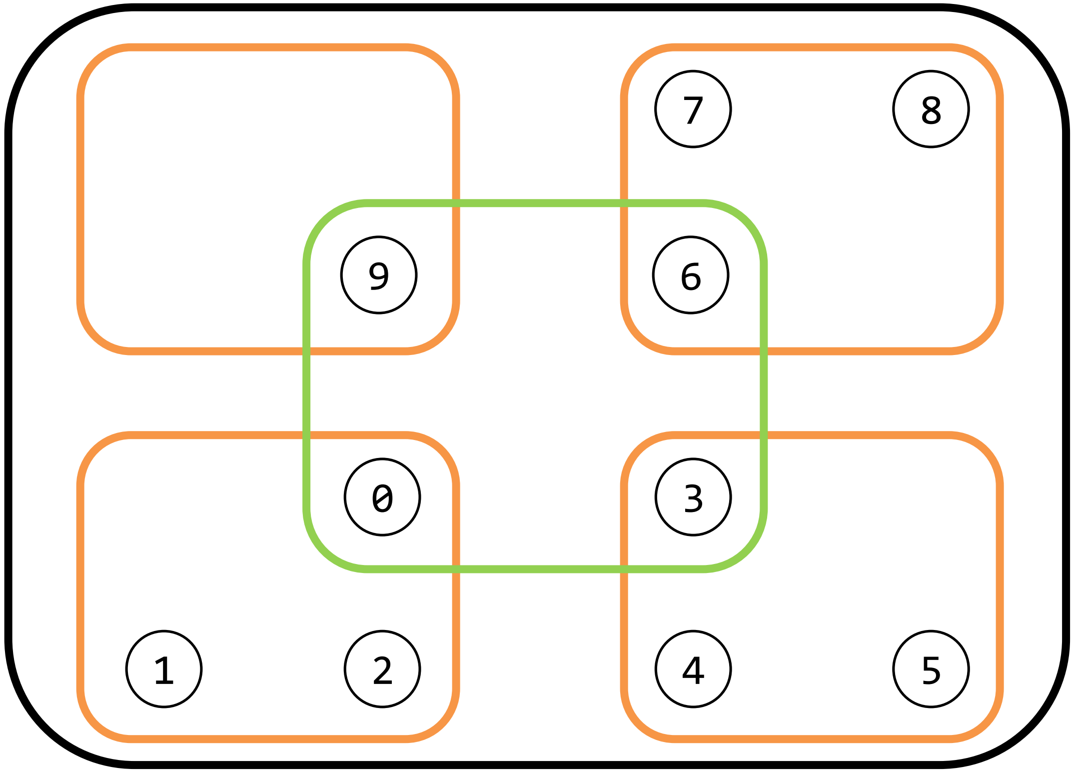
HONOURS PROGRAMME HP-SR
in Information Technology

- **Build a networking layer transparent to the application, which will reduce the impact of a fault.**
- **Upon fault, only the processes that directly communicate with the failed process have to participate in the repair procedure.**
- **Some processes can proceed without repairing…**
- **… at the cost of some communication overhead.**

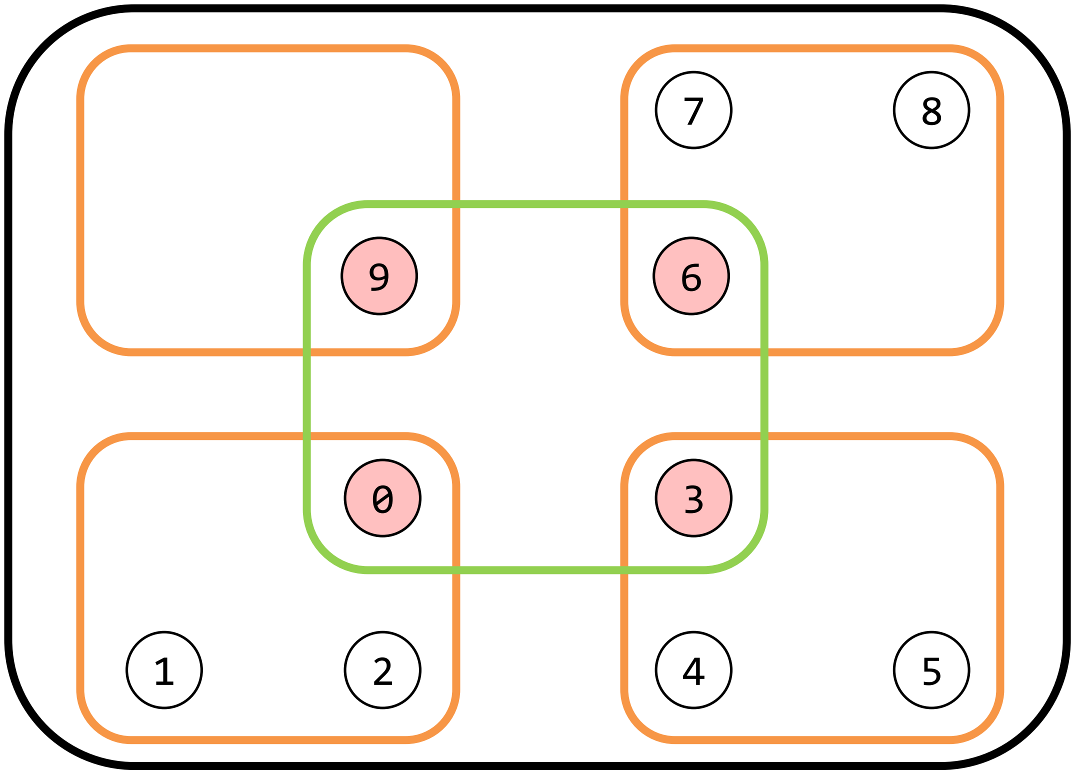
Hierarchical Legio principles

- Entire comm
- Local_comm
- Global_comm

- Linear # of comm
- Connected
- Single shortest path

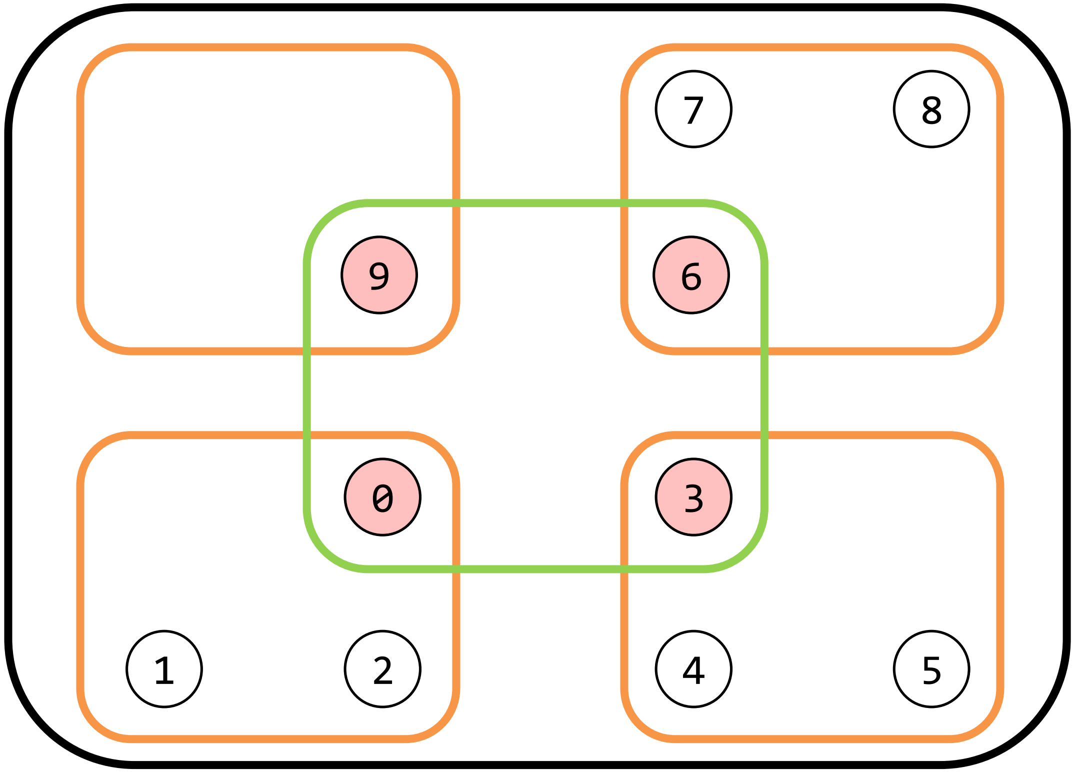


Hierarchical Legio communication



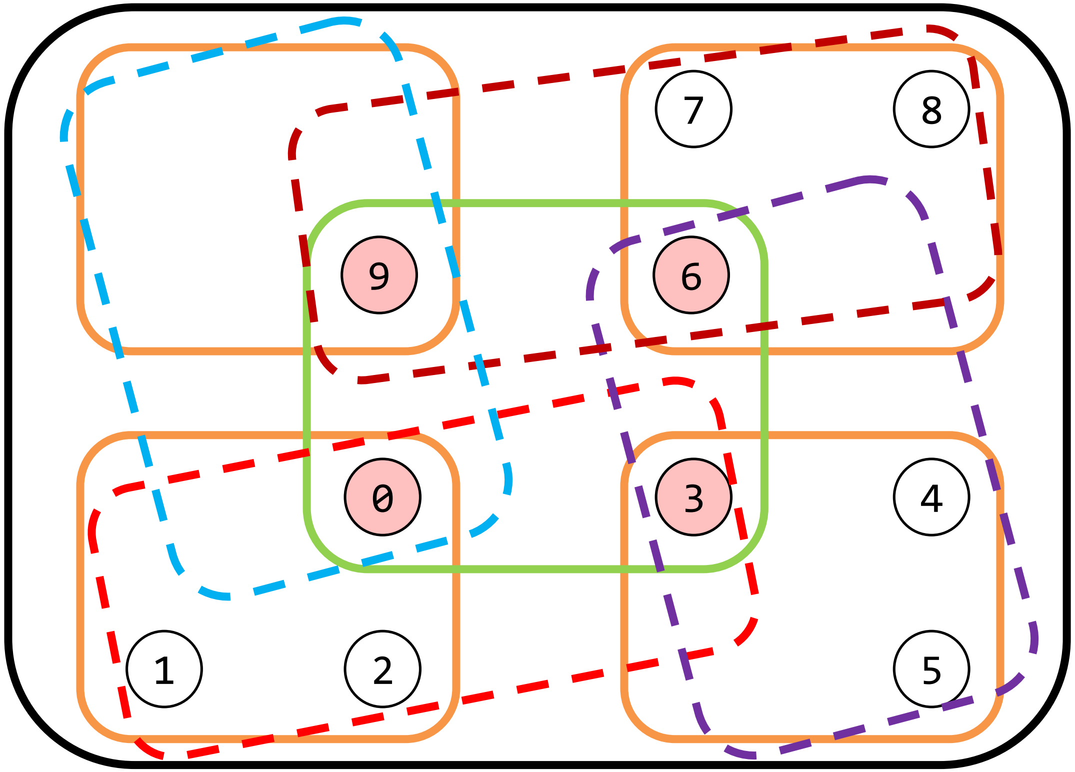
- One-to-one
- One-to-all
- All-to-one
- All-to-all
- Comm-creator
- File op
- Local_only

Hierarchical Legio repair procedure



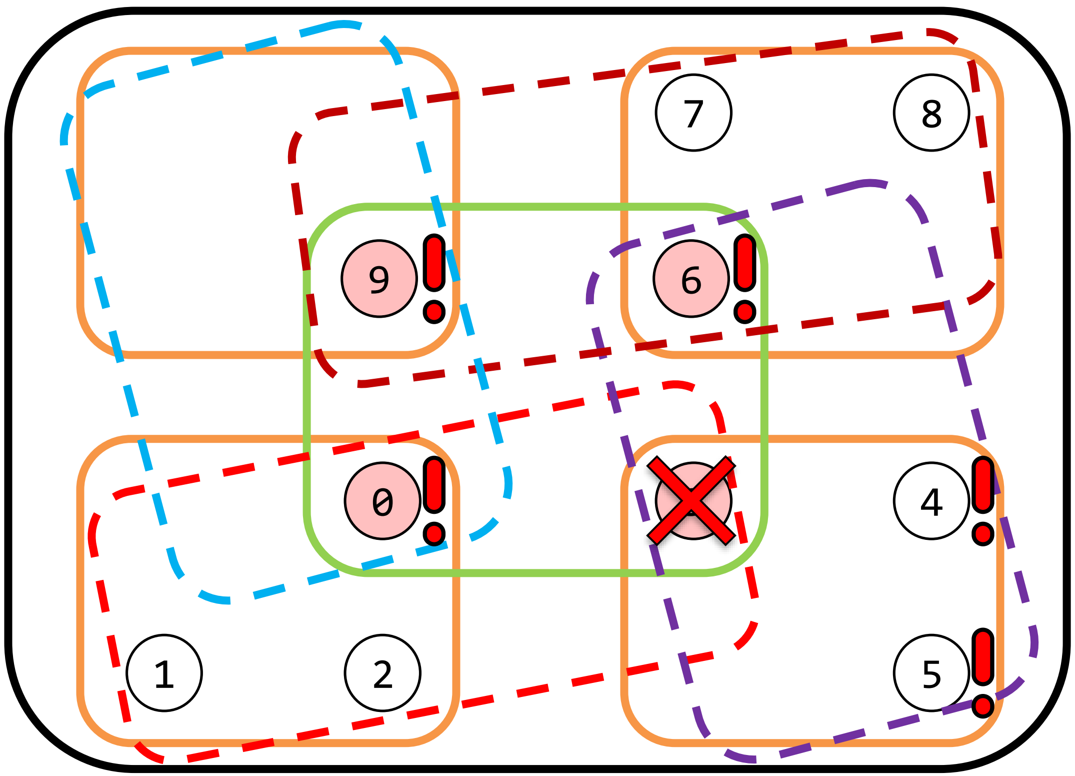
- **Non-master faults are trivial, repairing the local_comm is enough.**

Hierarchical Legio repair procedure



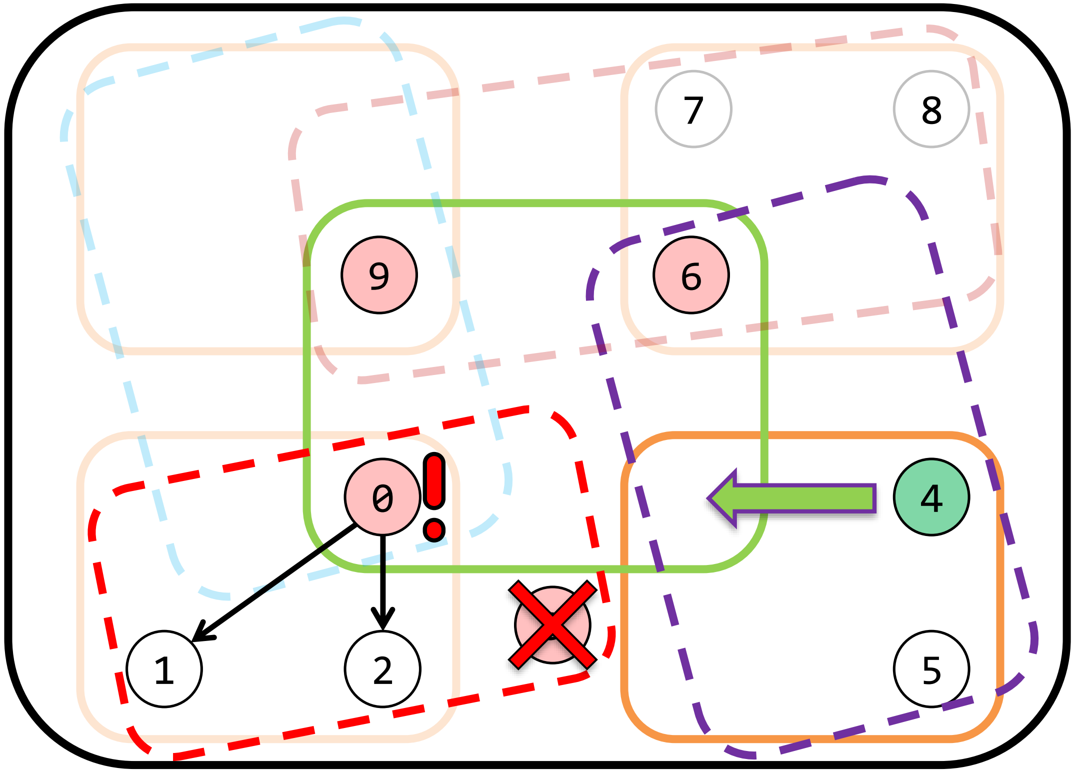
- To deal with the faults of the master processes, we need additional communicators.
- POV comms

Hierarchical Legio repair procedure



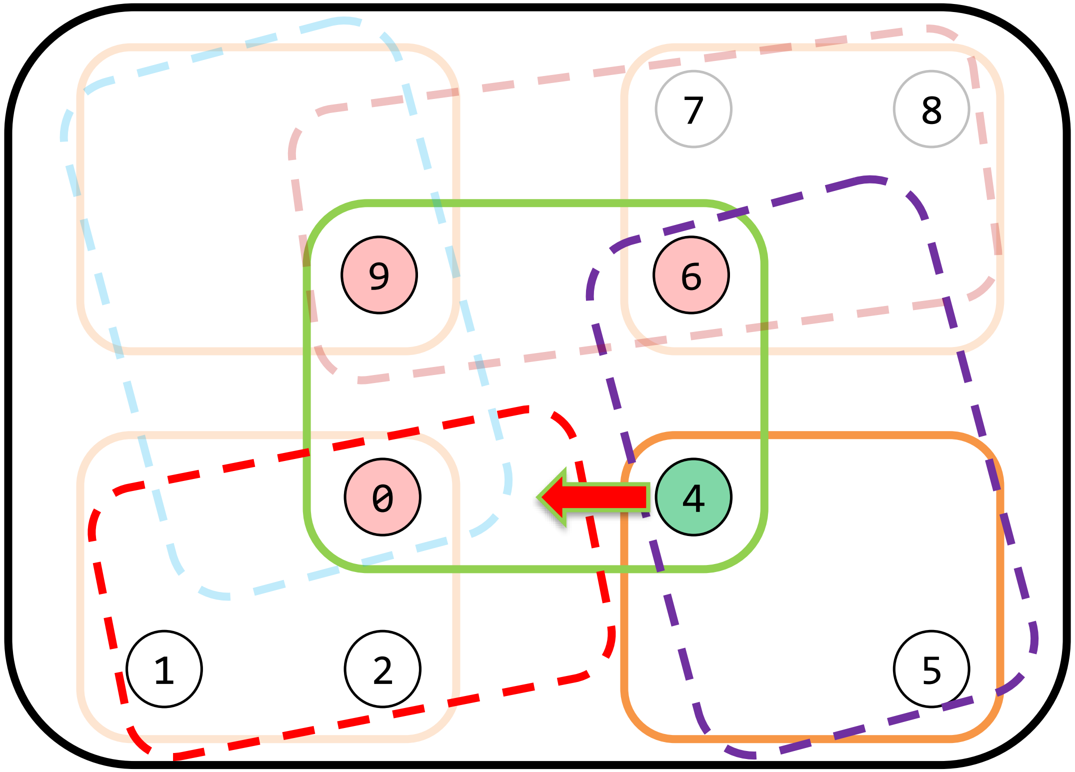
- 3 fails
- 4 comms need repair
- All the processes within local and global notice the fault
- 1 and 2 do not

Hierarchical Legio repair procedure



- **4 new master**
- **Using purple, 4 gets into global**
- **Meanwhile 0 propagates the notification to 1 and 2, which can shrink**

Hierarchical Legio repair procedure



- **4 joins red using green**

Hierarchical Legio repair complexity

- **In the hierarchical case, the shrink complexity can be written as follows:**

$$R_H(s,k) = \begin{cases} S(k) + 2S(k+1) + S\left(\frac{s}{k}\right) \\ S(k) \end{cases}$$

depending if a master or non-master failed.

- **If S(.) scales linearly or worse with the number of processes involved, then we proved that:**

$$\exists s_0 (\forall s > s_0 (\exists k | R_H(s,k) < S(s)))$$

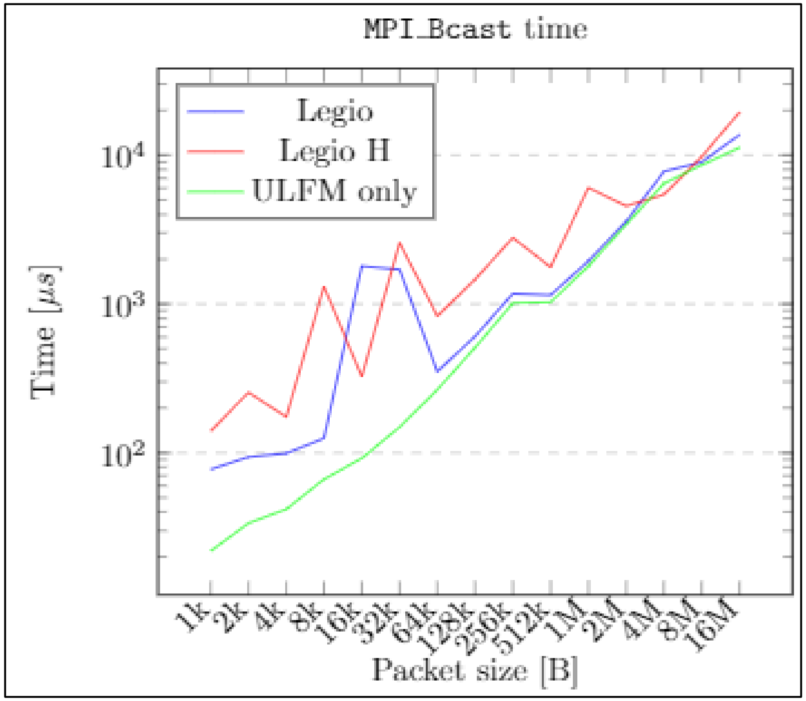
Experimental campaign

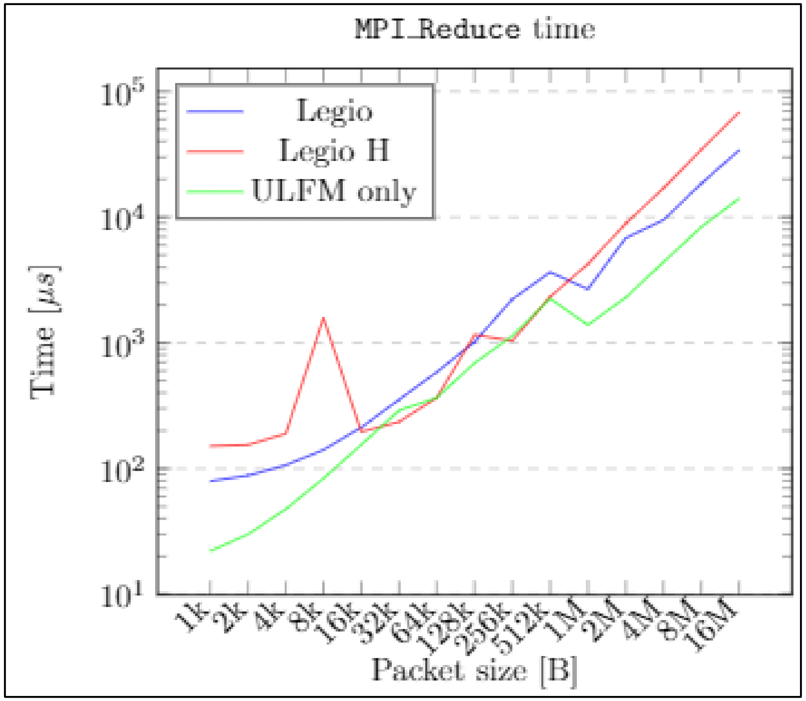
- **We used the Marconi100 cluster (11th most powerful in the world), 32 processes per node.**

- **Two types of experiments:**
  - **Per-operation overhead measurement;**
  - **Application impact measurement.**

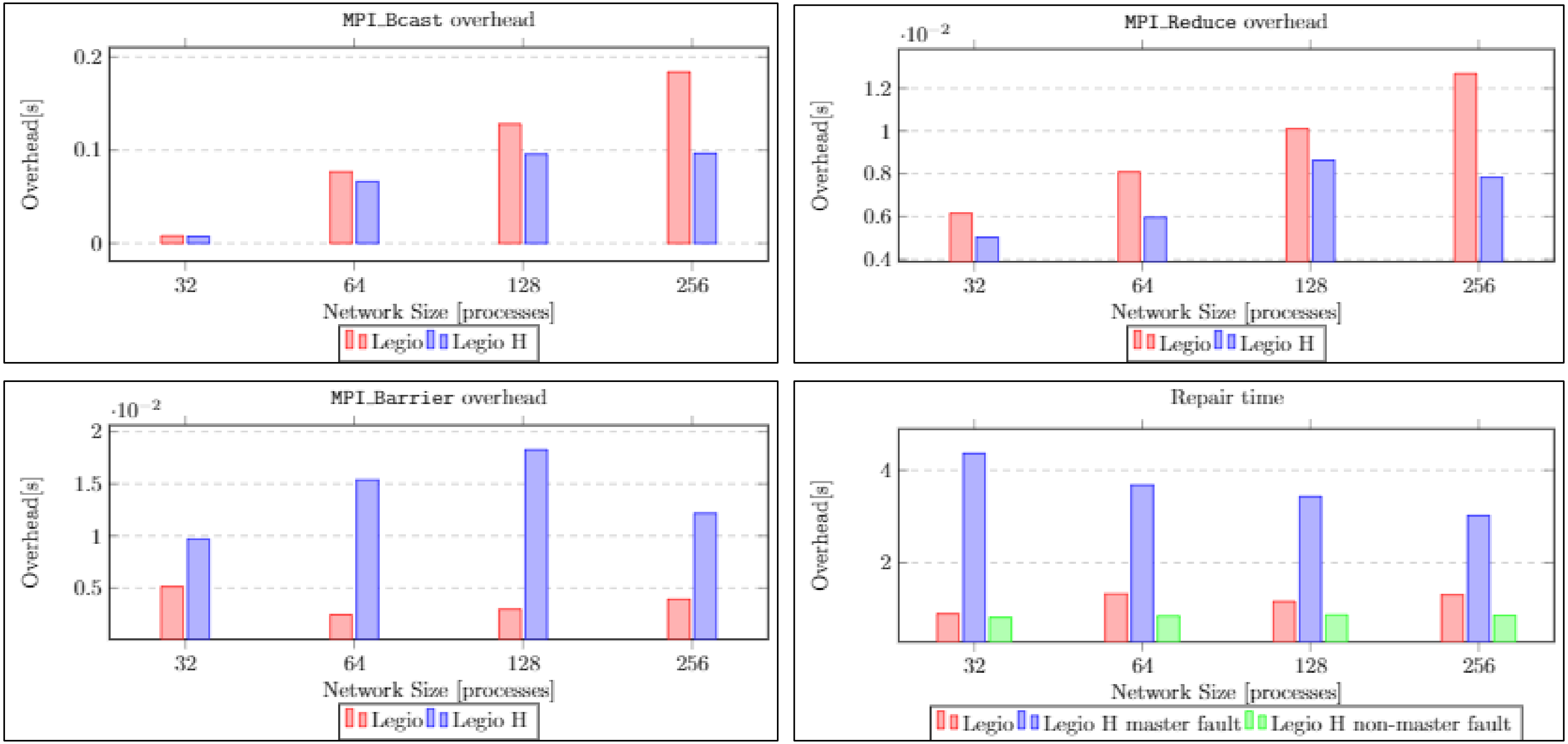
- **For each type, we considered two different experiments.**

Experimental campaign: mpiBench

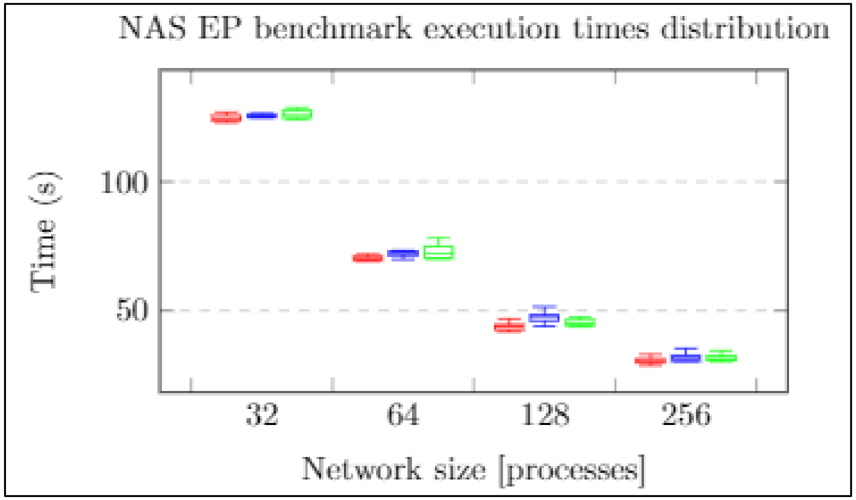


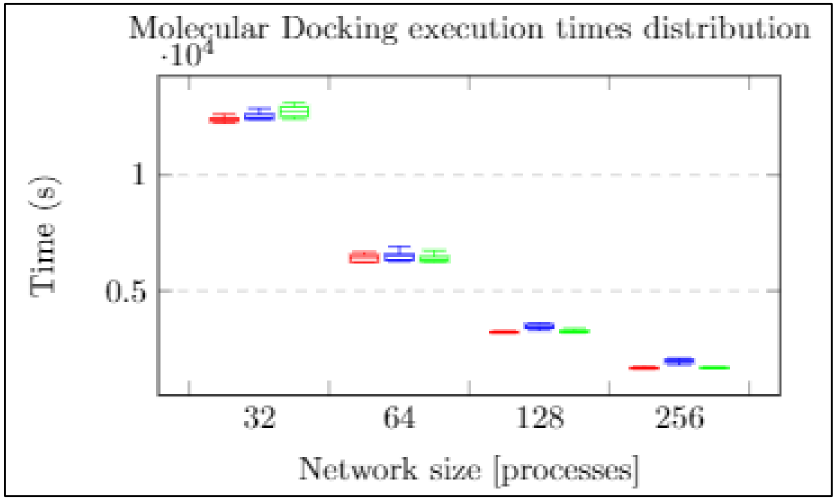


Experimental campaign: ad-hoc code



Experimental campaign: EP applications





Conclusions

- ✓ Experiments showed the effectiveness of the developed Legio framework. The low overhead is a key feature.
- ✓ The evolution, despite performing more operations, showed comparable results, and can be relevant especially in big executions.
- ✓ The transparency is a desirable property not present in similar frameworks.
- ✓ The problem will be increasingly relevant, since the size of HPC architectures will continue to grow.