Fault tolerance for a data flow model

Improve classical fault tolerance protocols using the application knowledge given by its data flow representation

Xavier Besseron
xavier.besseron@imag.fr

PhD supervisor: Thierry Gautier

Laboratory of Informatics of Grenoble (LIG) – INRIA
MOAIS Project

March 2010

Outline

Outline

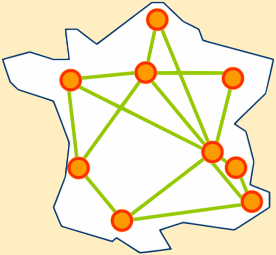
Grid computing

What are grids?

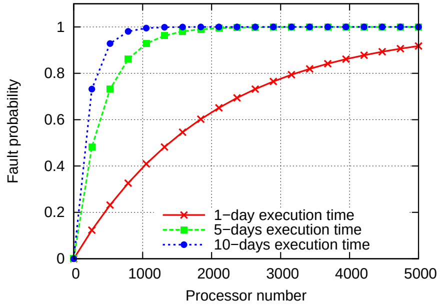
- Clusters are computers connected by a LAN
- Grids are clusters connected by a WAN
- Heterogeneous (processors, networks, ...)
- Dynamic (failures, reservations, ...)

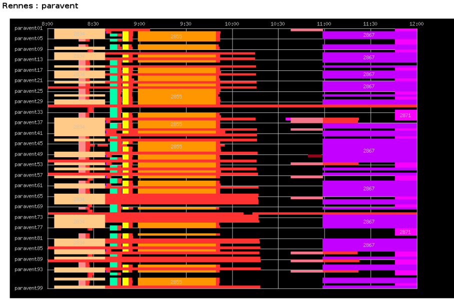
Aladdin – Grid'5000

- French experimental grid platform
- More than 4800 cores
- 9 sites in France
- 1 site in Brazil
- 1 site in Luxembourg



Fault tolerance





Why fault tolerance?

- Fault probability is high on a grid
- Split a large computation in shorter separated computations
- Capture application state and reconfigure it dynamically

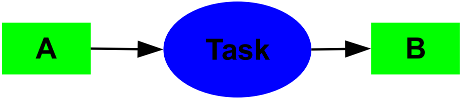
Outline

Kaapi's data flow model

Data flow model

- Shared Data = object in a global memory
- Task = function call, accessing shared data
- Access mode = constraint on shared data access (read, write, ...)

```
Shared<Matrix> A;
Shared<double> B;
Fork<Task>() (A,B);
```

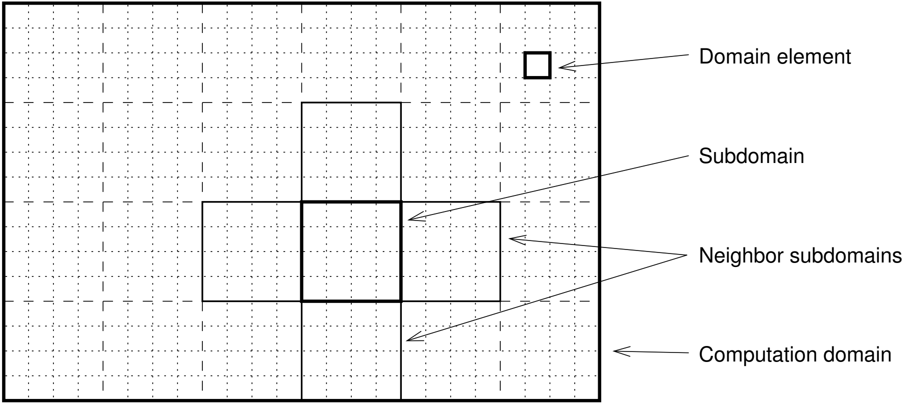


Application example: Jacobi3D

- Solve a Poisson problem
- Domain decomposition parallelization
- Jacobi iterative method

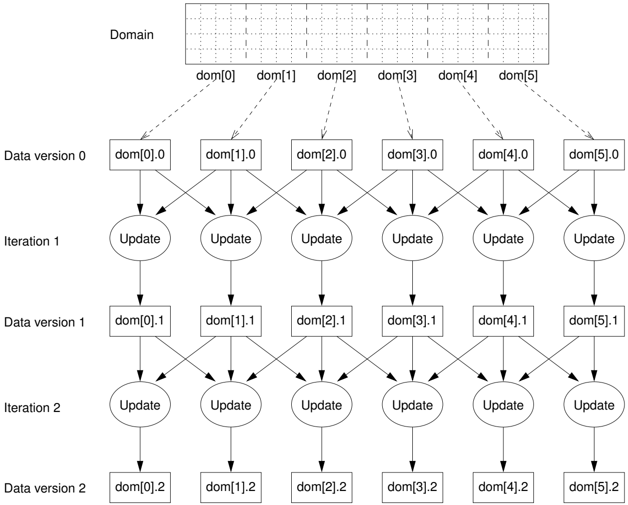
Jacobi3D: Domain decomposition

Example with a 2D domain



Subdomain ⟷ Shared data in the data flow graph

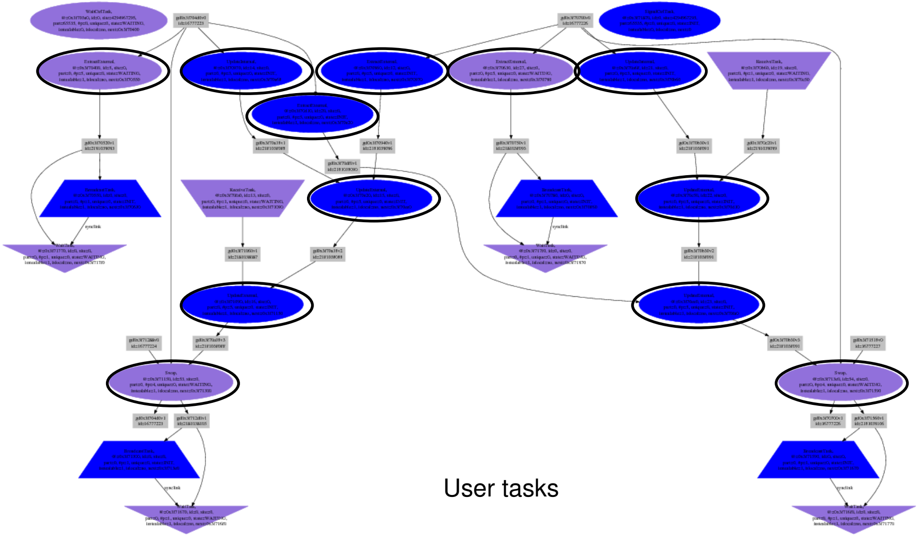
Jacobi3D: Domain decomposition & iterations



- Tasks are deterministic, ie same input $\Rightarrow$ same output
- Execution order respects the data flow constraints

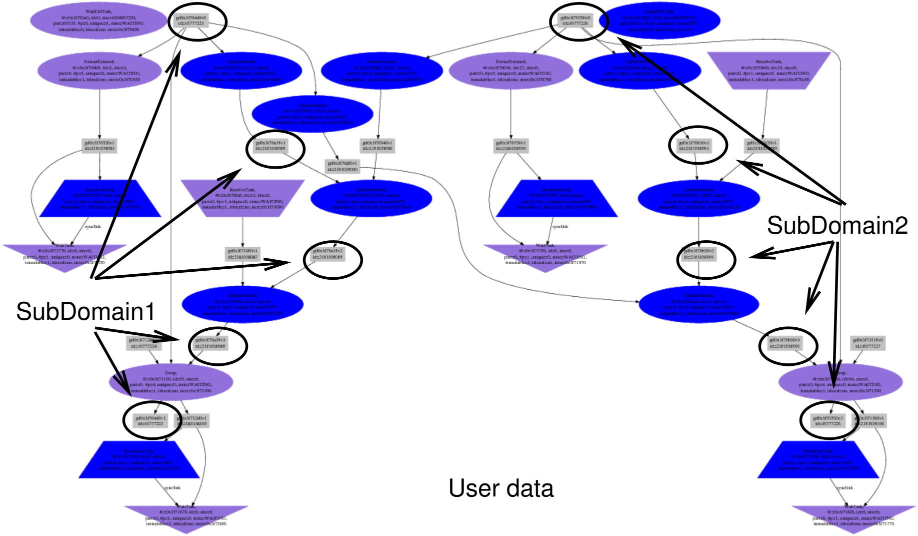
Jacobi3D: Real data flow graph

Data flow graph generated by Kaapi for processor *N*



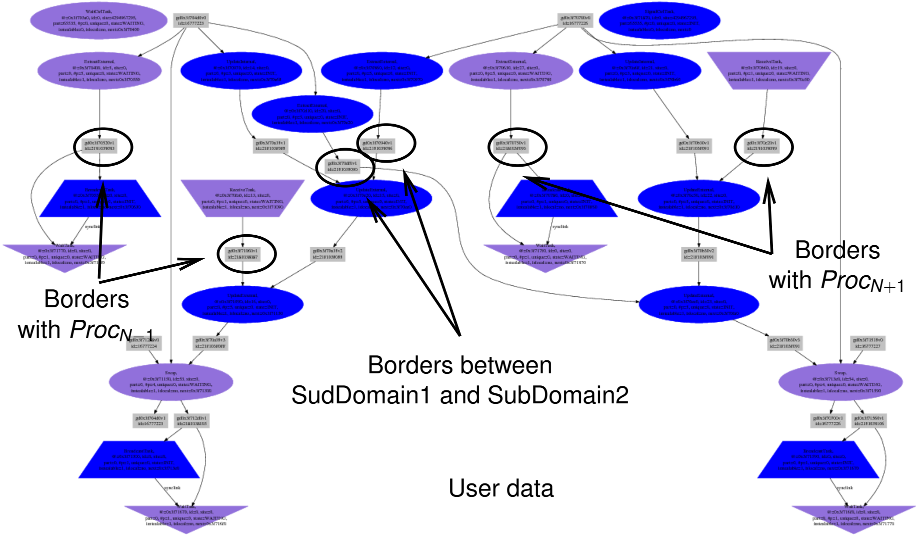
Jacobi3D: Real data flow graph

Data flow graph generated by Kaapi for processor *N*



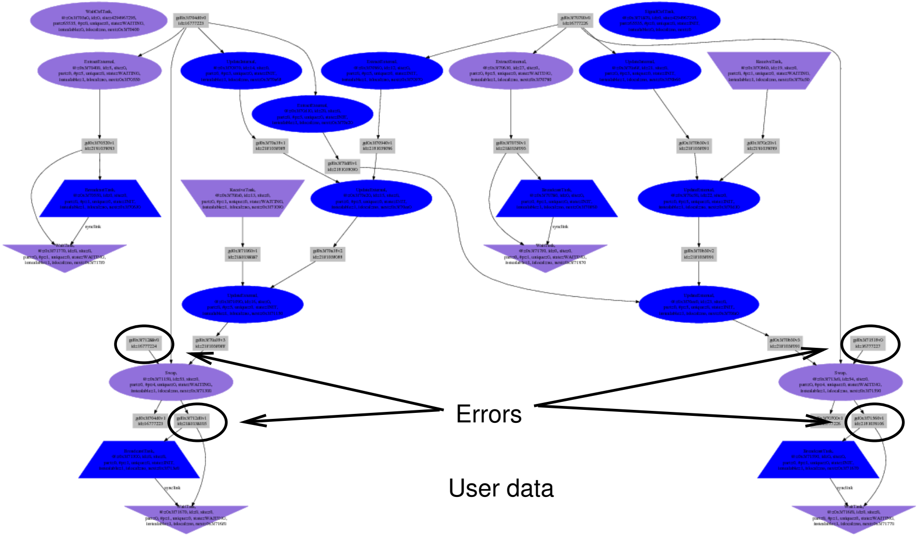
Jacobi3D: Real data flow graph

Data flow graph generated by Kaapi for processor N



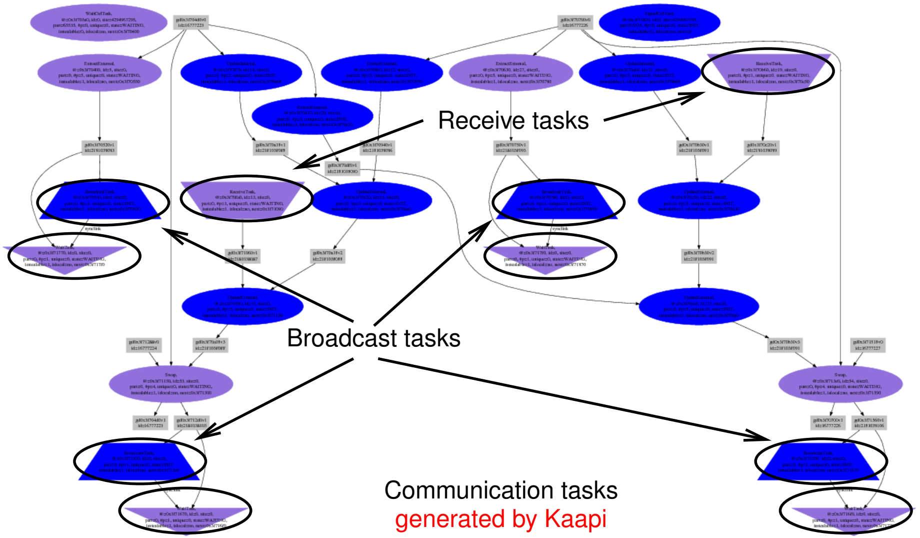
Jacobi3D: Real data flow graph

Data flow graph generated by Kaapi for processor *N*



Jacobi3D: Real data flow graph

Data flow graph generated by Kaapi for processor *N*



Outline

Coordinated checkpoint

Principle

Take a consistent snapshot of an application:

- Coordinate all the processes to ensure a consistent global state
- Save the processes snapshots on a stable memory

Issues

- Coordination cost at large scale
- Data transfert time for large application state

References

- Coordinated checkpoint/rollback protocol: blocking[Tamir84], non-bloblocking[Chandy85]
- Implementations: CoCheck [Stellner96], MPICH-V [Coti06], Charm++ [Zheng04], OpenMPI [Hursey07], ...

Improving coordination step

Classical coordination step

Save a consistent global snapshot:

- requires to send a message on all communication channels

Without knowledge of communication pattern, this coordination may require message exchange from all processes to all processes.

$\Rightarrow$ Number of exchanged messages is $O(N^2)$ (N = process number).

Coordinated Checkpointing in Kaapi

Equivalent to a blocking coordinated checkpoint, but

- Checkpointing a process = Saving the data flow graph and its input data
- Based on the reconfiguration mechanism of Kaapi (*see next slide*)
- Reduce the number of exchanged messages during coordination

$\Rightarrow$ Number of exchanged messages is $O(kN)$ (with $k \ll N$).

Dynamic reconfiguration mechanism in Kaapi

Allows to safely reconfigure a distributed set of objects by ensuring a mutually consistent view of the objects

Find the neighbor processes

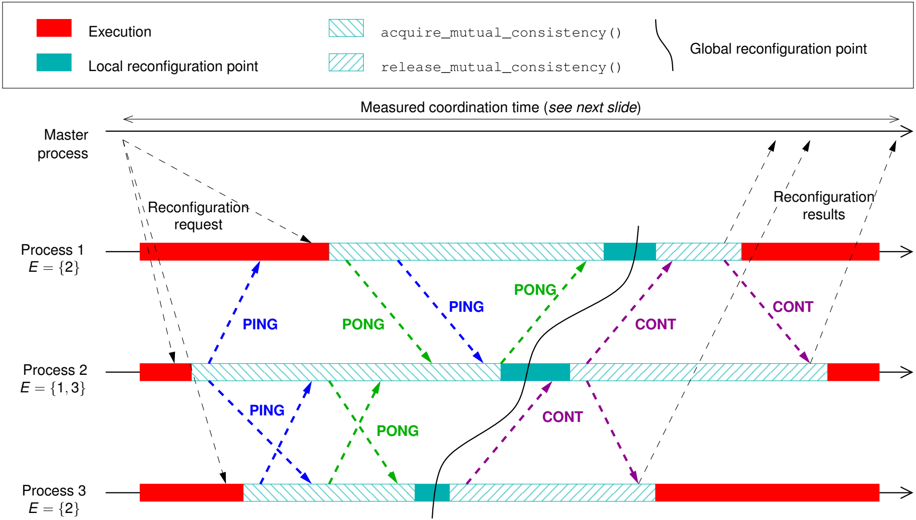
Data flow graph allows to know the future communications

- Neighbors processes are processes that can emit message to the considered process
- Identify tasks that generate communications
- Only flush channels with the neighbor processes

Properties

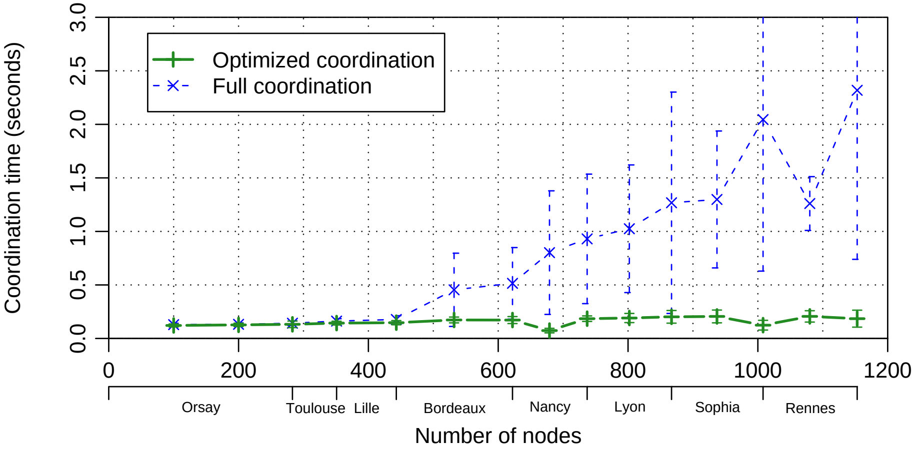
- Ensure consistency and accessibility of the application
- k is the average number of neighbors processes
  - application and scheduling dependent
  - for N-Queens application with work-stealing scheduling: $k < 2$
  - for Jacobi3D application with graph partitioning: $k \approx 7$

Mutual consistency protocol in Kaapi



Experimental results: Coordination time

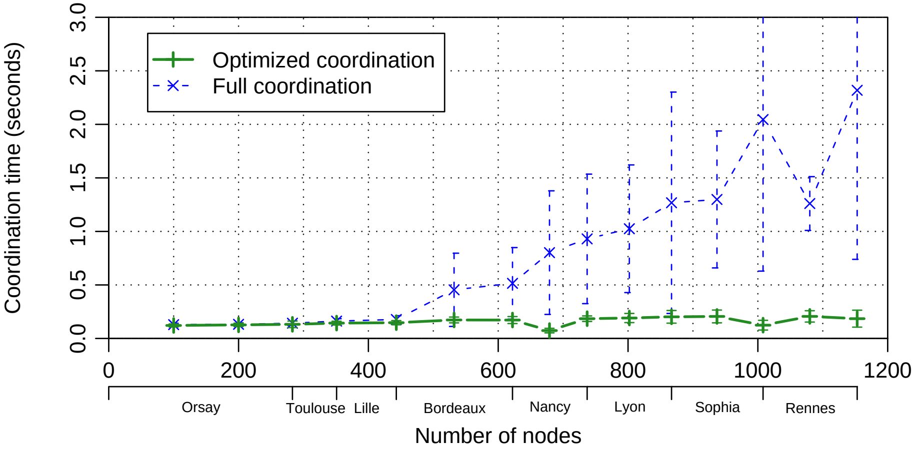
N-Queens application using work-stealing scheduler
No checkpoint, only coordination



But for large application state, coordination time is small compared to data transfert.

Experimental results: Coordination time

N-Queens application using work-stealing scheduler
No checkpoint, only coordination



But for large application state, coordination time is small compared to data transfert.

Outline

Global rollback

Principle

- Checkpointed states are consistent global states
- All processes rollback to the last checkpointed state

Good performances after global rollback require either

- Spare nodes to replace the failed ones
  - reserve spare nodes that could be used for another computation
  - wait for others nodes to be available or for failed nodes to be fixed
- or Load balancing algorithms
  - using over-decomposition, ie placing many subdomains per processor

Question: What is the influence of over-decomposition on the execution time?

- after failure of f nodes
- without spare nodes

XPs: over-decomposition influence

Experience 1: influence on the execution time

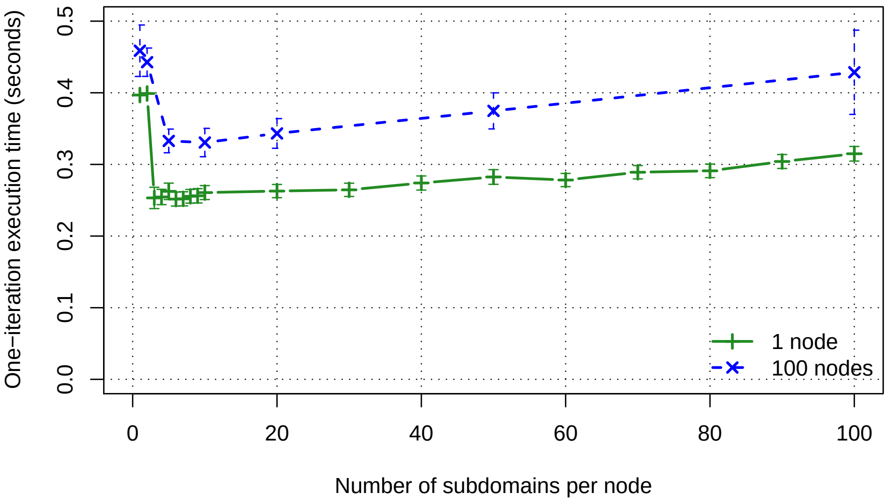
- Execution time in function of the decomposition *d*, ie the number of subdomains
- 3D domain, constant size per node: 10^7 `double`-type reals
  - On 1 node: 10^7 reals, ie $\approx$ 76 MB
  - On 100 nodes: 100×10^7 reals, ie $\approx$ 7.6 GB
- Nancy cluster of Grid'5000

Experience 2 : influence on the execution time after global recovery

- Execution time in function of the decomposition *d* and of the number of failed nodes *f*
- 3D domain: 100×10^7 reals with type `double` ($\approx$ 7.6 GB)
- Using 100 nodes of the Nancy cluster of Grid'5000
- Execution on $100 - f$ nodes

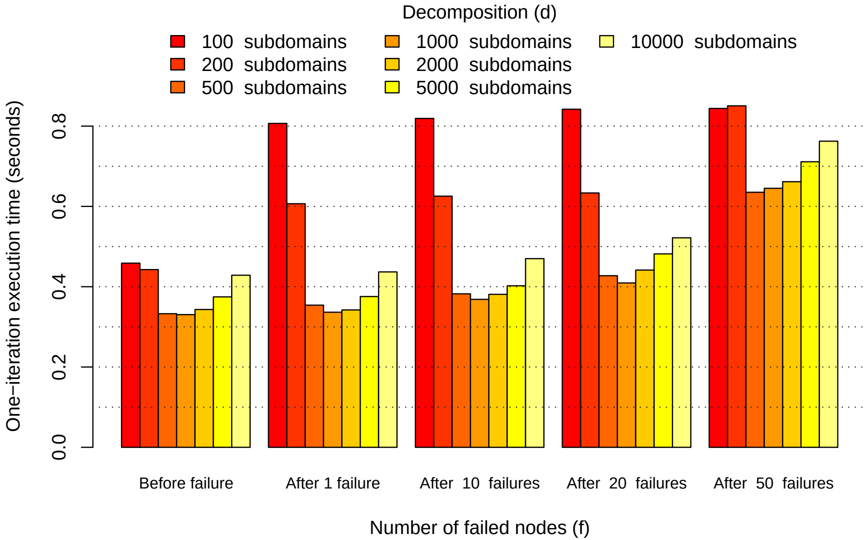
XPs: over-decomposition influence

Experience 1: Execution time



XPs: over-decomposition influence

Experience 2: Execution time after global recovery



Outline

Partial rollback

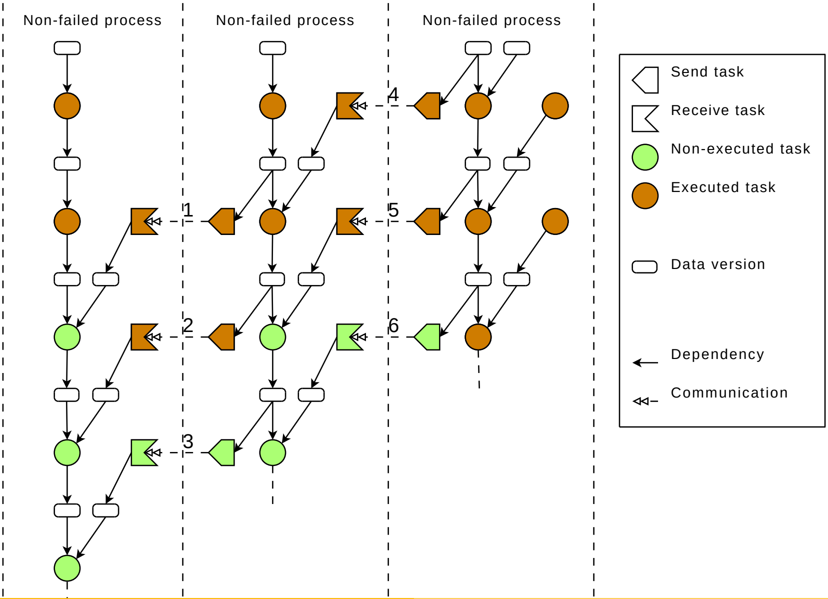
Principle

- Restart failed processes from last checkpoint
- Replay communications to the restarted processes
  - no message logging
  - re-execute tasks that produced the communications

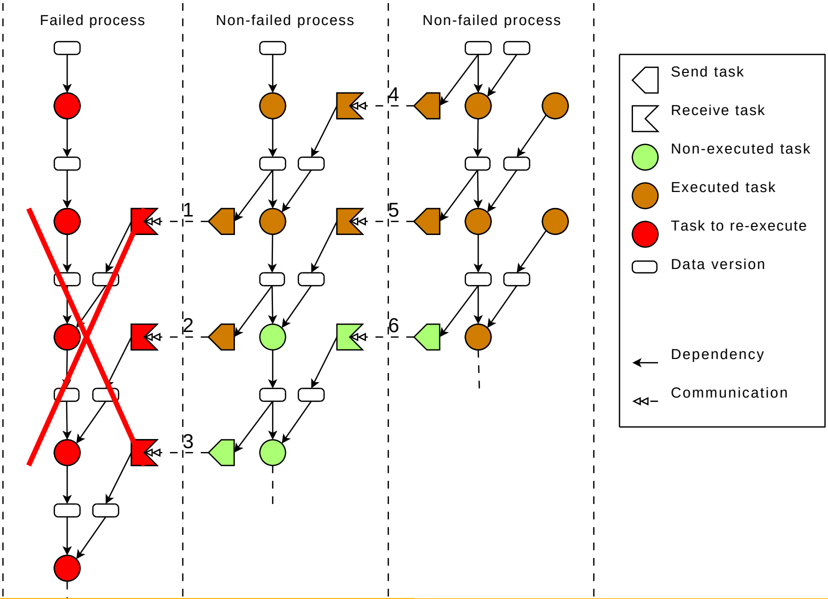
Two aspects

- Find the set of tasks required for restarting
  - this represents the lost work
- Schedule the lost work
  - in order to reduce the overhead induced by the failure

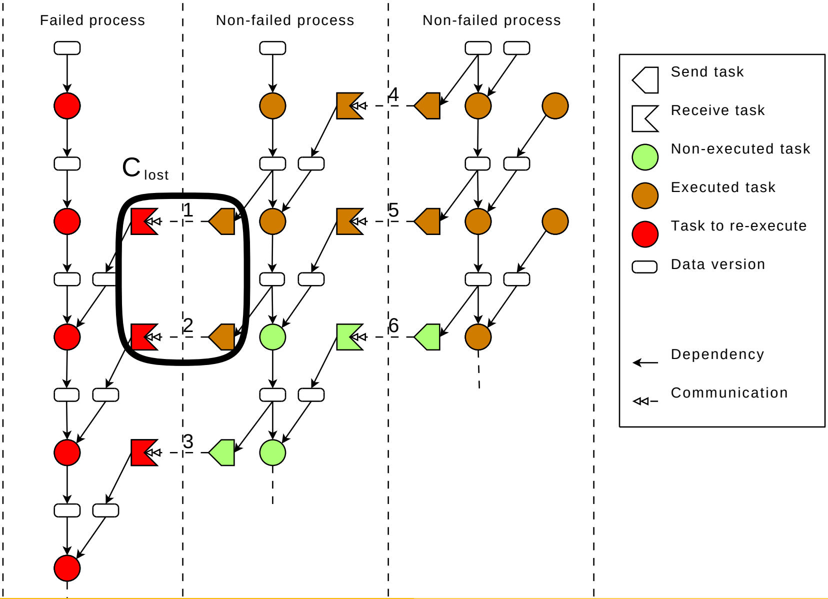
Partial rollback principle: Execution



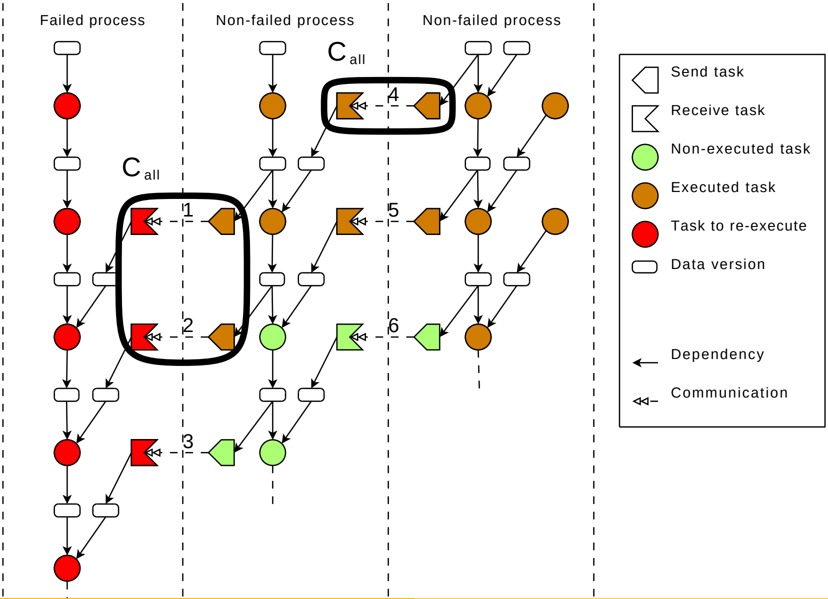
Partial rollback principle: Failure



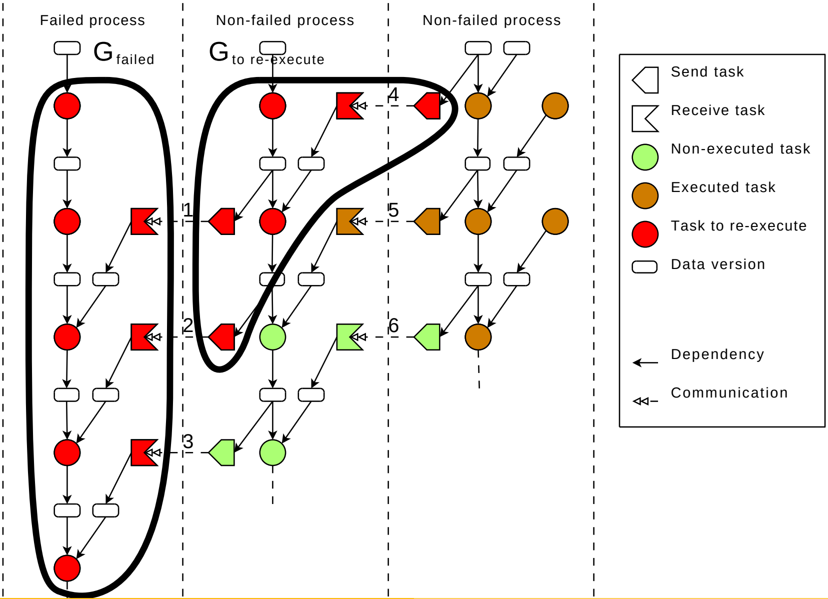
Partial rollback principle: Lost communications



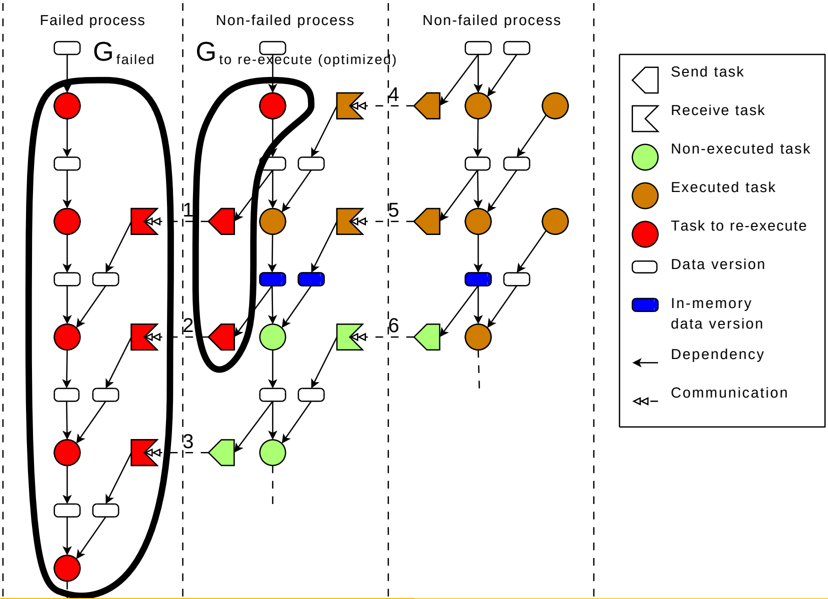
Partial rollback principle: Communications to replay



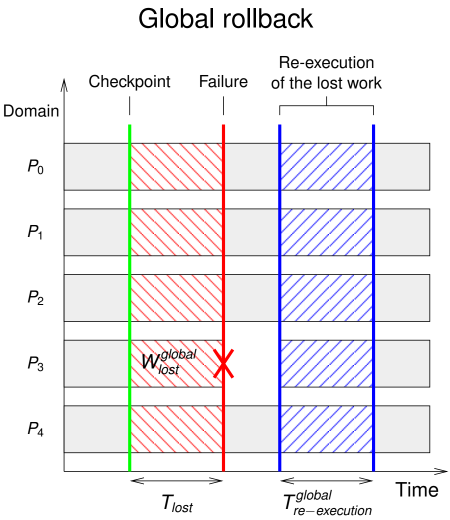
Partial rollback principle: Tasks to re-execute

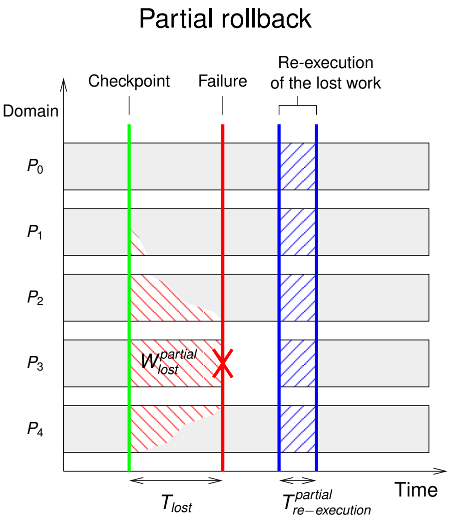


Partial rollback principle: In-memory data



Global vs partial rollback: Reexecution of the lost work

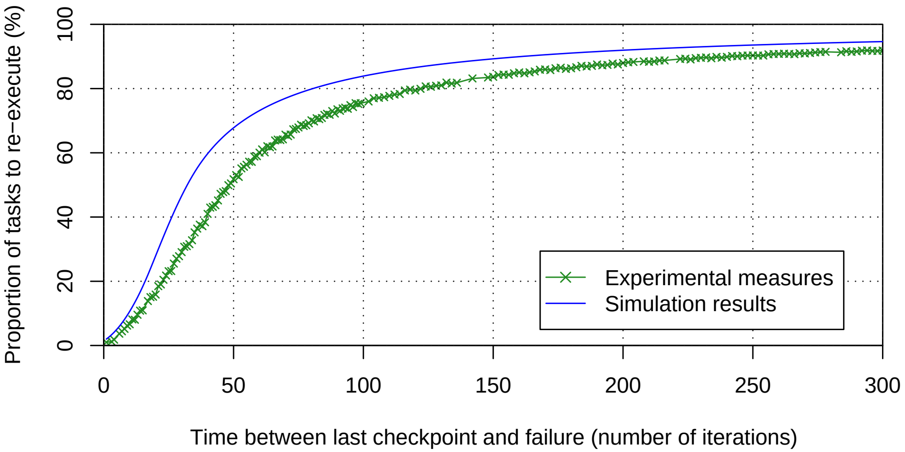




Thanks to over-decomposition,
the lost work can be parallelized !

Partial rollback: Proportion of tasks to re-execute

- Jacobi3D executed on 100 nodes
- 40 × 40 × 1 subdomains, ie 16 subdomains per node
- Failure of 1 fixed node



Partial rollback: Time to re-execute the lost work

Experimental conditions

- 100 computation nodes, 10 checkpoint servers (Bordeaux cluster)
- Domain size = 76 MB, splitted in 1000 subdomains
- Failure of 1 fixed node
- Considering 2 grains:
  - 2 ms for a subdomain update
  - 50 ms for a subdomain update

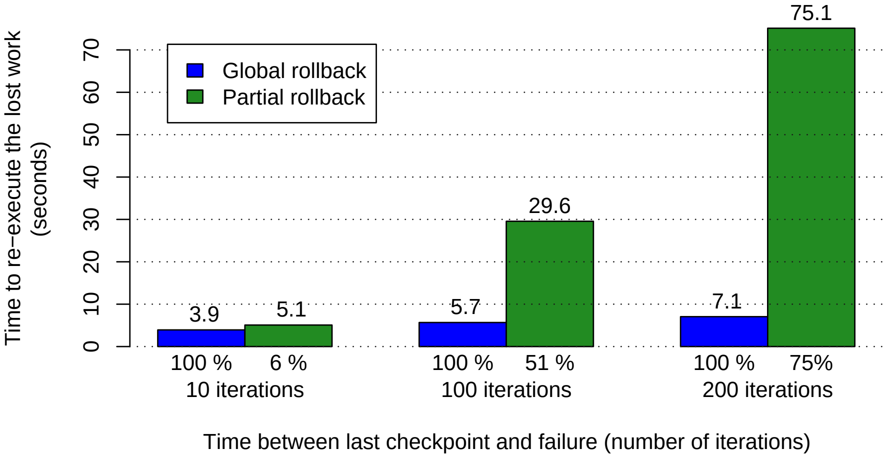
Measured value

- Time to re-execute the lost work:

Data redistribution + Computation

Partial rollback: Time to re-execute the lost work

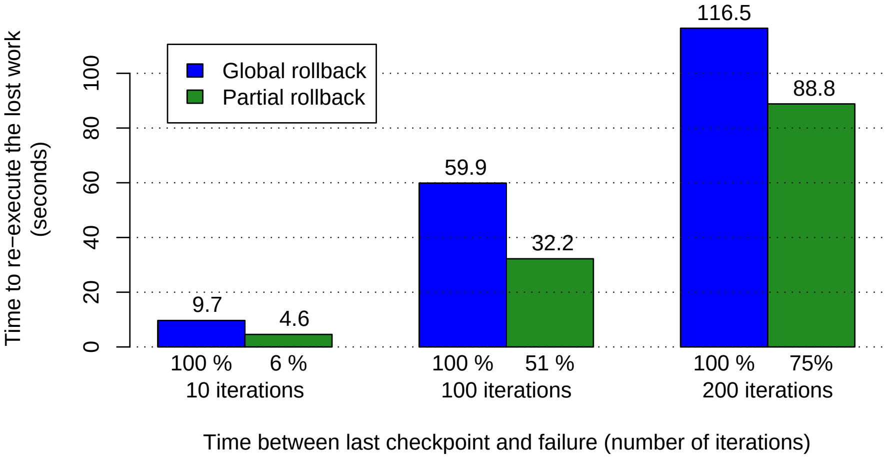
Time of a subdomain update $\approx$ 2 ms



$\Rightarrow$ Scheduling should take in consideration the previous data placement

Partial rollback: Time to re-execute the lost work

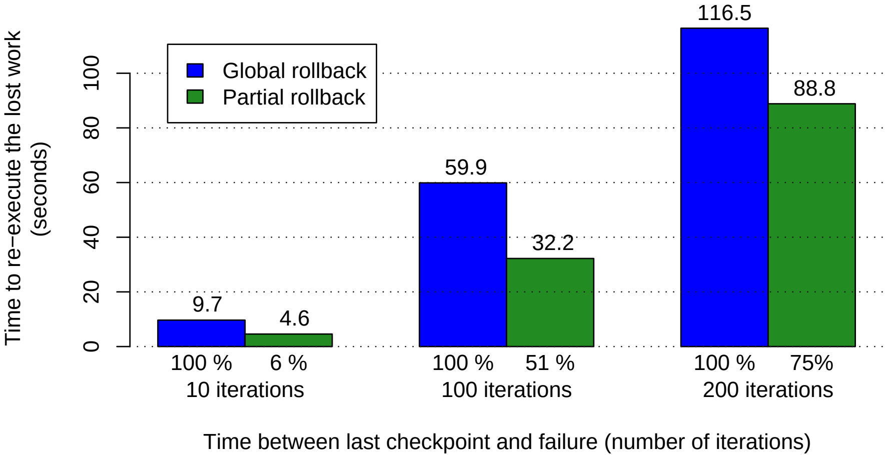
Time of a subdomain update ≈ 50 ms



⇒ Scheduling should take in consideration the previous data placement

Partial rollback: Time to re-execute the lost work

Time of a subdomain update ≈ 50 ms



⇒ Scheduling should take in consideration the previous data placement

Outline

Perspectives

Scheduling algorithms for partial recovery

Need to take in consideration the data placement and communication cost

- minimize makespan with communication ⇒ NP-hard
- find and try some heuristics

RDMA support in Kaapi

Currently communications in Kaapi are based on active messages

⇒ Data copy on reception

Optimization: Use RDMA (Remote Direct Memory Access) for data transfert

Reducing the data transfert cost during checkpoint and recovery step

- Incremental checkpoint for Kaapi (based on DFG)
- Placing checkpoint servers near the computation nodes
  - require to take in consideration the network topology

Other contributions

Dynamic reconfiguration

Allows dynamic change on the application while ensuring:

- Concurrency management
  - Concurrent & cooperative execution ⇒ X-Kaapi
- Mutual consistency
  - Consistent view of a distributed set of objects

Software development (mostly Kaapi)

Kaapi (≈ 100 000 lines of code)

- Authors: T. Gautier, V. Danjean, S. Jafar [TIC], D. Traoré [KaSTL], L. Pigeon, X. Besseron

My developments & contributions:

- Graph partitioning scheduling (≈ 10 000 lines of code)
- Fault tolerance support (≈ 10 000 lines of code)
- Large scale deployments & multi-grids computations (using TakTuk)

Thanks for your attention

Questions?

Outline

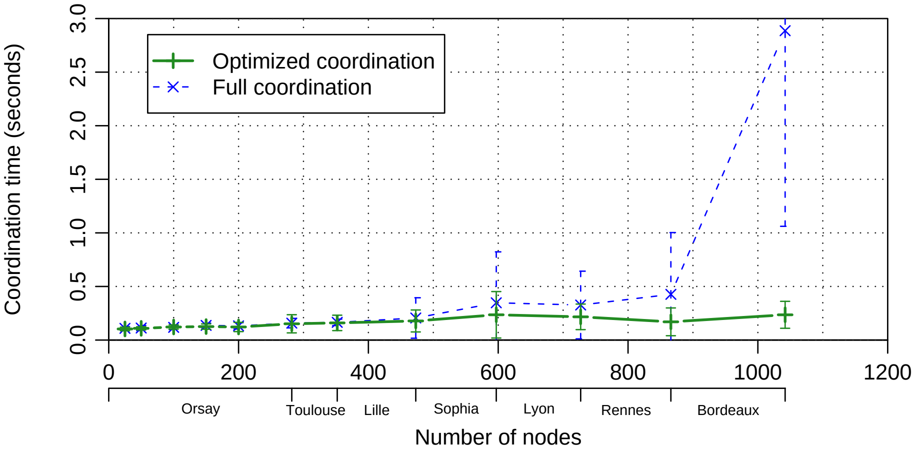
7 Coordination time

8 Placement of checkpoint servers

9 Over-decomposition

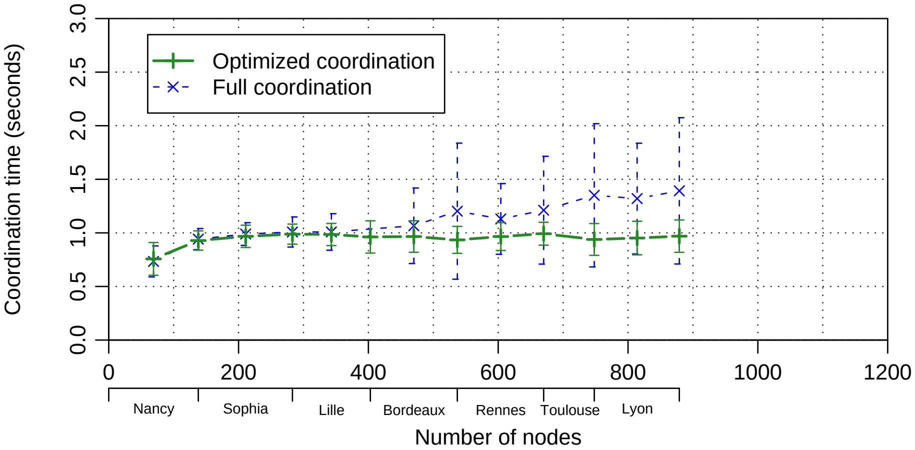
Experimental results: Coordination time

N-Queens application using work-stealing scheduler
No checkpoint, only coordination



Experimental results: Coordination time

N-Queens application using work-stealing scheduler
No checkpoint, only coordination



Outline

7 Coordination time

8 Placement of checkpoint servers

9 Over-decomposition

Placement of checkpoint servers

Idea

Reduce the checkpointing time by placing the checkpoint servers near the computation processes

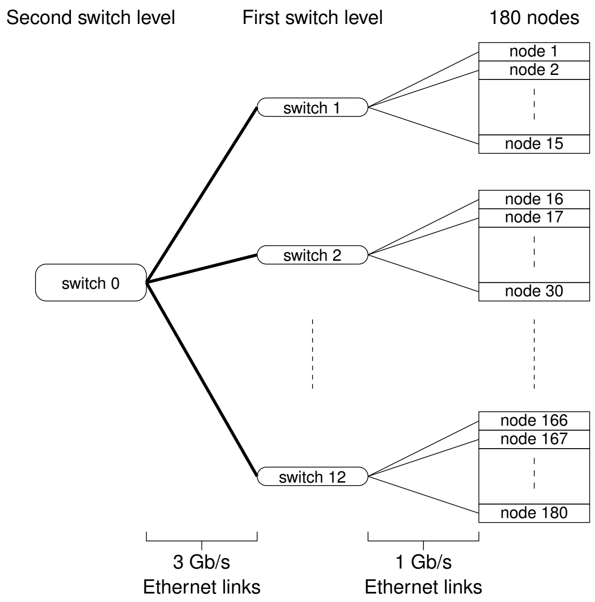
Practically, checkpoint servers can be:

- a dedicated node of the cluster
- another computation process (buddy-processor of Charm++)

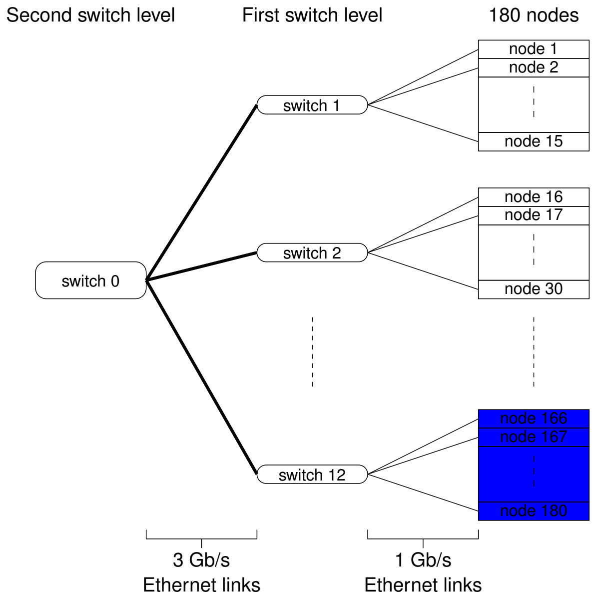
Experimental study

- 180 nodes of the Orsay cluster from Grid'5000
  - 120 nodes for computation
  - 12, 24 or 60 nodes for checkpoint servers
- Application state $\approx$ 20 GB, ie 169 MB per node
- Testing 3 placement methods: ordered, by-switch and random

Network topology of the Orsay cluster

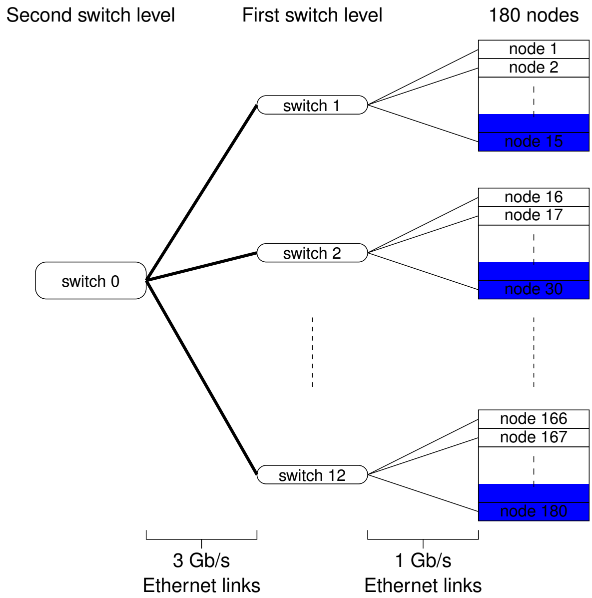


Network topology of the Orsay cluster



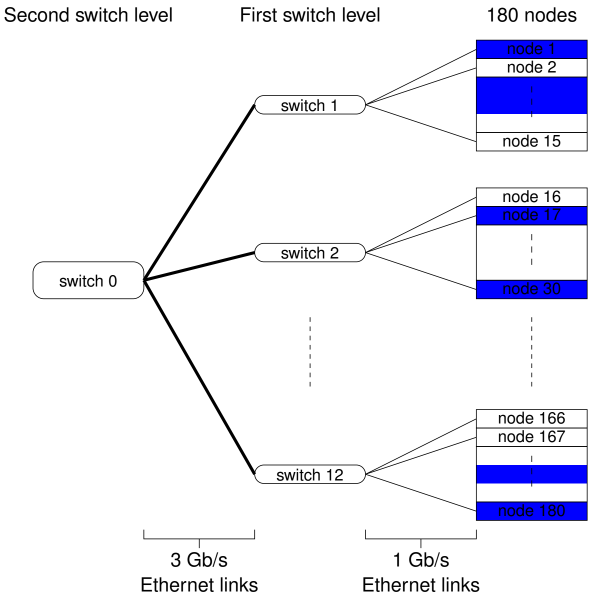
Checkpoint servers can be placed by following the node order

Network topology of the Orsay cluster



Checkpoint servers can be placed by switch

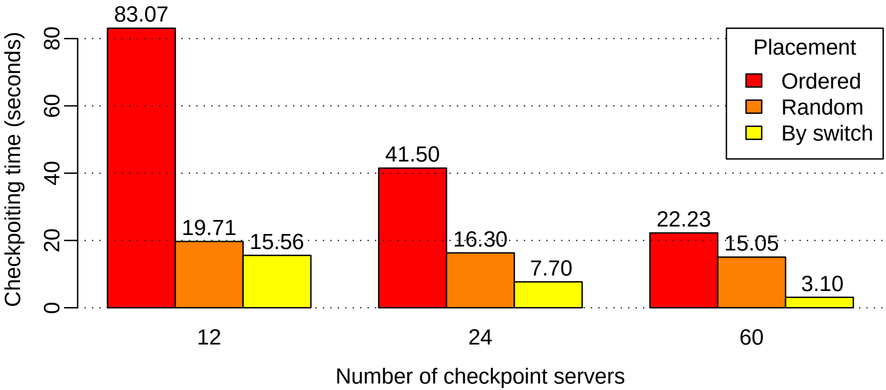
Network topology of the Orsay cluster



Checkpoint servers can be placed randomly

Placement of checkpoint servers in the Orsay cluster

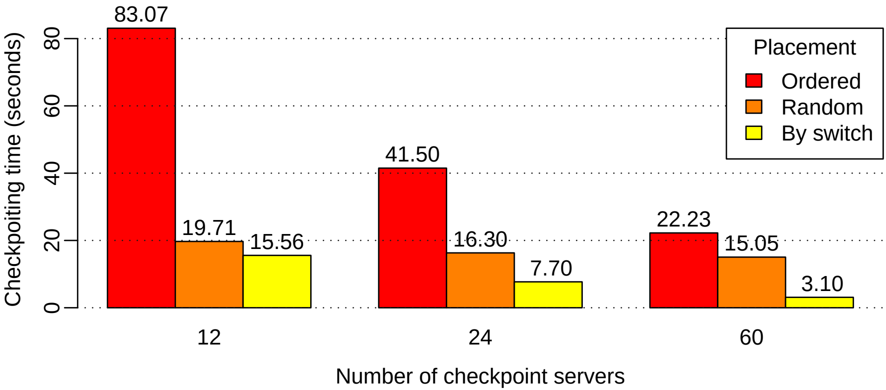
120 computation nodes
Application state ≈ 20 GB, ie 169 MB per node



⇒ Need to take in consideration the network topology
Could be done automatically (using Network Weather Service for example)

Placement of checkpoint servers in the Orsay cluster

120 computation nodes
Application state ≈ 20 GB, ie 169 MB per node



⇒ Need to take in consideration the network topology
Could be done automatically (using Network Weather Service for example)

Outline

Over-decomposition on Jacobi3D

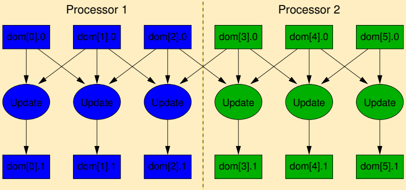
Number of nodes: n, number of subdomains: d

- Classical decomposition (MPI): $n = d$
- Over-decomposition: $d \gg n$

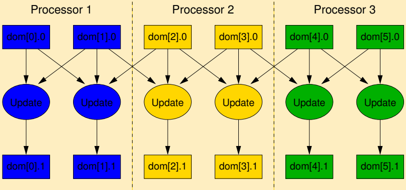
$\Rightarrow$ Over-decomposition allows to be independent of the processor number

Example: "Over"-decomposition in 6 subdomains

Distribution on 2 processors



Distribution on 3 processors



Over-decomposition influence: Modelization

Let T_n^d be the execution time of one iteration for

- a d-subdomains decomposition
- using n processors

- Execution time $T_n^d = \lceil \frac{d}{n} \rceil \times \frac{T_1^1}{d}$
- Optimal time T_n^n is for $d = n$
- Over-decomposition overhead is

$$T_n^d / T_n^n = \left\lceil \frac{d}{n} \right\rceil \times \frac{n}{d} \leq 1 + \frac{n}{d}$$

After the global recovery and load balancing

- f is the number of failed nodes
- After failures, over-decomposition overhead is

$$T_{n-f}^d / T_{n-f}^{n-f} = \left\lceil \frac{d}{n-f} \right\rceil \times \frac{n-f}{d} \leq 1 + \frac{n}{d}$$

Over-decomposition influence: Modelization

Simulating execution on $1000 - f$ processors