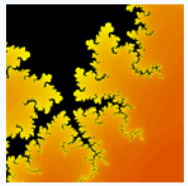
KERLABS

Error monitoring, processing and possible recovery strategies

Pascal Gallard, Matthieu Fertré, Louis Rilling

Outlines

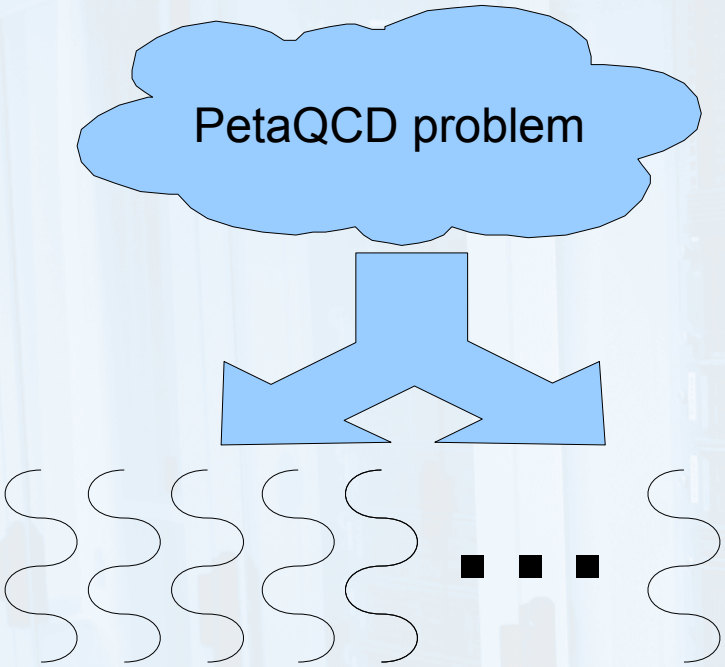


- PetaQCD constraints
  - Theoretical constraint
  - Software assumptions
  - Hardware assumptions
- Fault tolerance approaches
- Checkpoint granularity
- Fault tolerance targets

PetaQCD constraints: Theoretical view

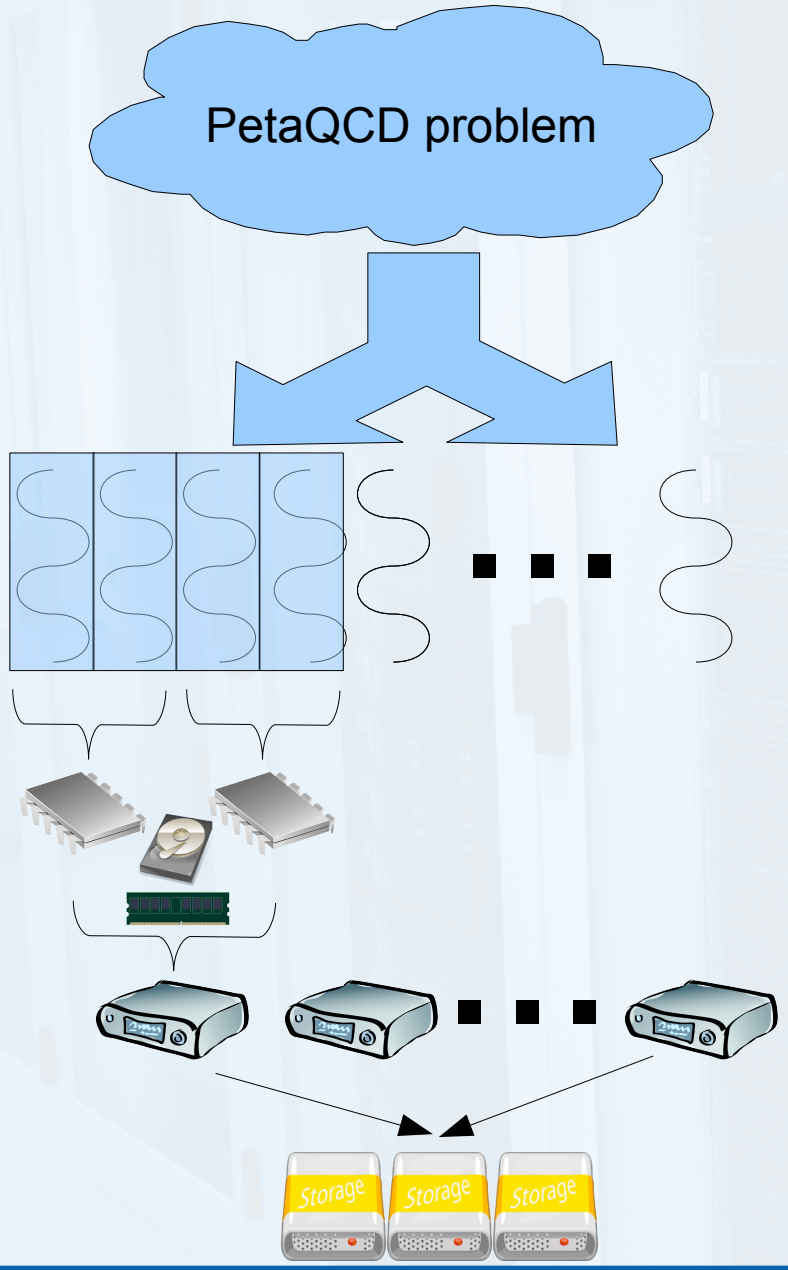
- Challenging application:
  - Very large memory to manage;
  - Very long execution time
  - Nearly every thing change between each computation step;
- Application based on convergent / divergent sequence
  - Convergence failure detector
- Questions are:
  - How can we store such huge running application ?
  - How often do we want to do that ?
  - How do we efficiently make such application fault-tolerant?

PetaQCD constraints: software view



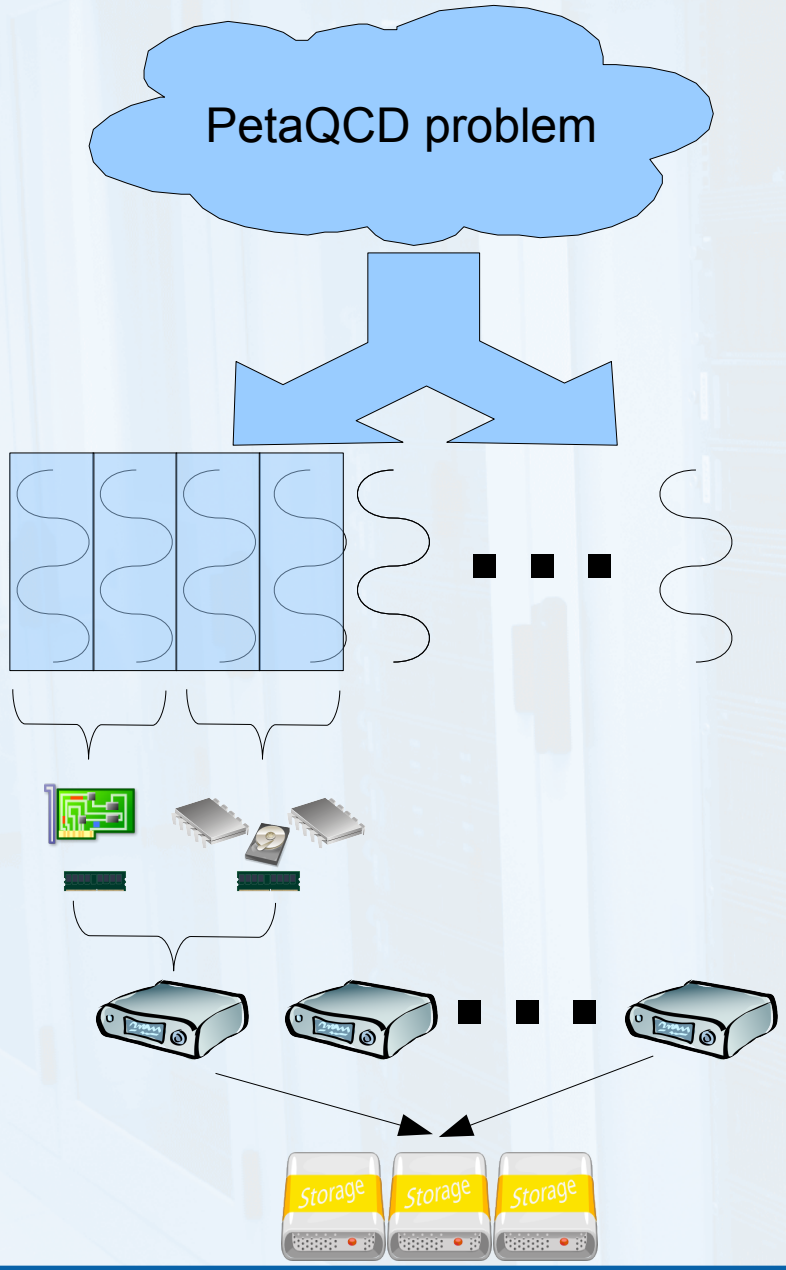
- Shorter execution time requires lot of computation units
- Need to make fault tolerant all these computation units
- Need to coordinate the fault tolerant mechanisms

PetaQCD constraint: Hardware view



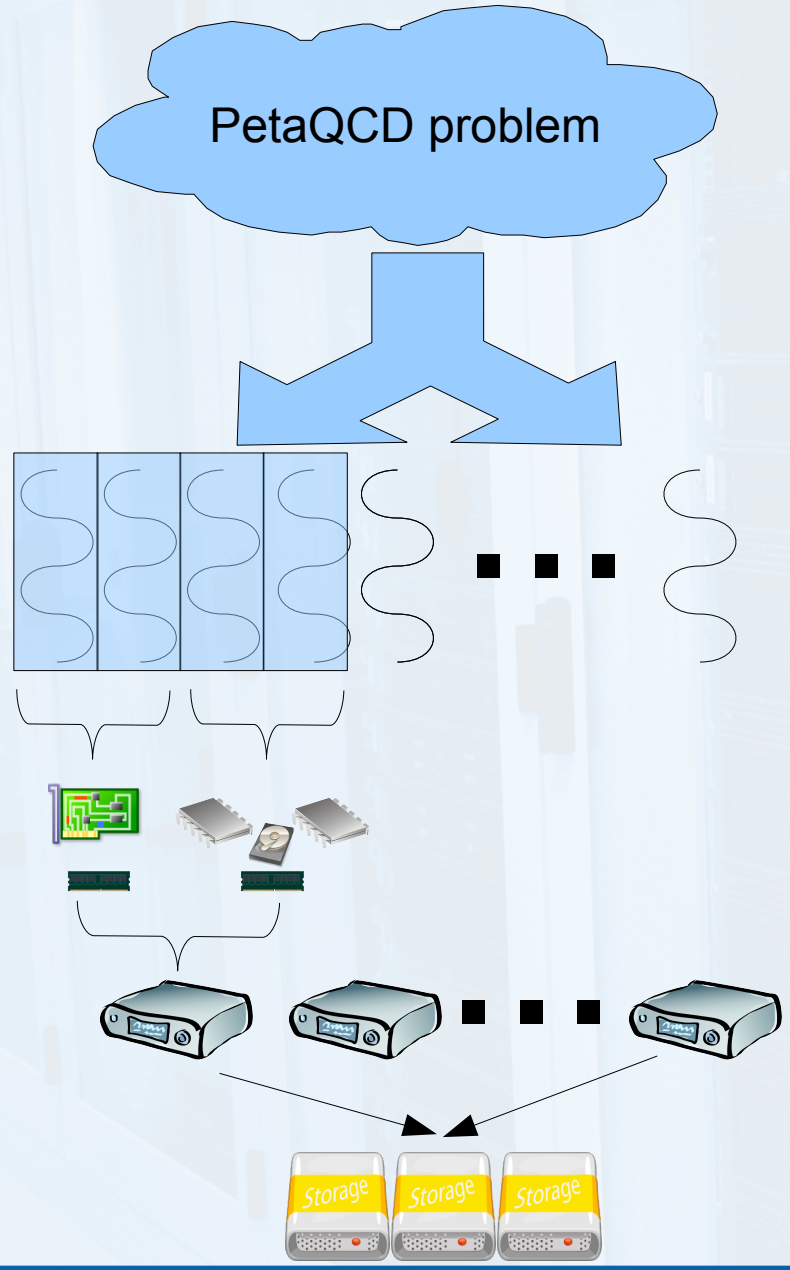
- Lot of computer resources
- Several hardware architectures
  - Regular clusters;

PetaQCD constraint: Hardware view



- Lot of computer resources
- Several hardware architectures
  - Regular clusters;
  - GPU clusters;

PetaQCD constraint: Hardware view



- Lot of computer resources
- Several hardware architectures
  - Regular clusters;
  - GPU clusters;
  - Cell clusters;
  - BlueGene;

Fault tolerance approaches

- Detection error
  - Application error, computation error, hardware failure, human action
- Fault tolerance policy:
  - Active policy
  - Coordinated checkpoint policy
  - Non coordinated checkpoint policy
  - Mixed approach

Fault tolerance approaches

- Detection error
  - Application error, computation error, hardware failure, human action
- Fault tolerance policy:
  - Active policy
    - Multiple run of a same computation
    - 2 runs: failure detector
    - At least 3 runs: failure correction
  - Coordinated checkpoint policy
  - Non coordinated checkpoint policy
  - Mixed approach

Fault tolerance approaches

- Detection error
  - Application error, computation error, hardware failure, human action
- Fault tolerance policy:
  - Active policy
  - Coordinated checkpoint policy
    - Freeze of all processes before the checkpoint
    - Easiest way to restart after a failure
    - All nodes must be restarted
    - Checkpoint network used in burst
  - Non coordinated checkpoint policy
  - Mixed approach

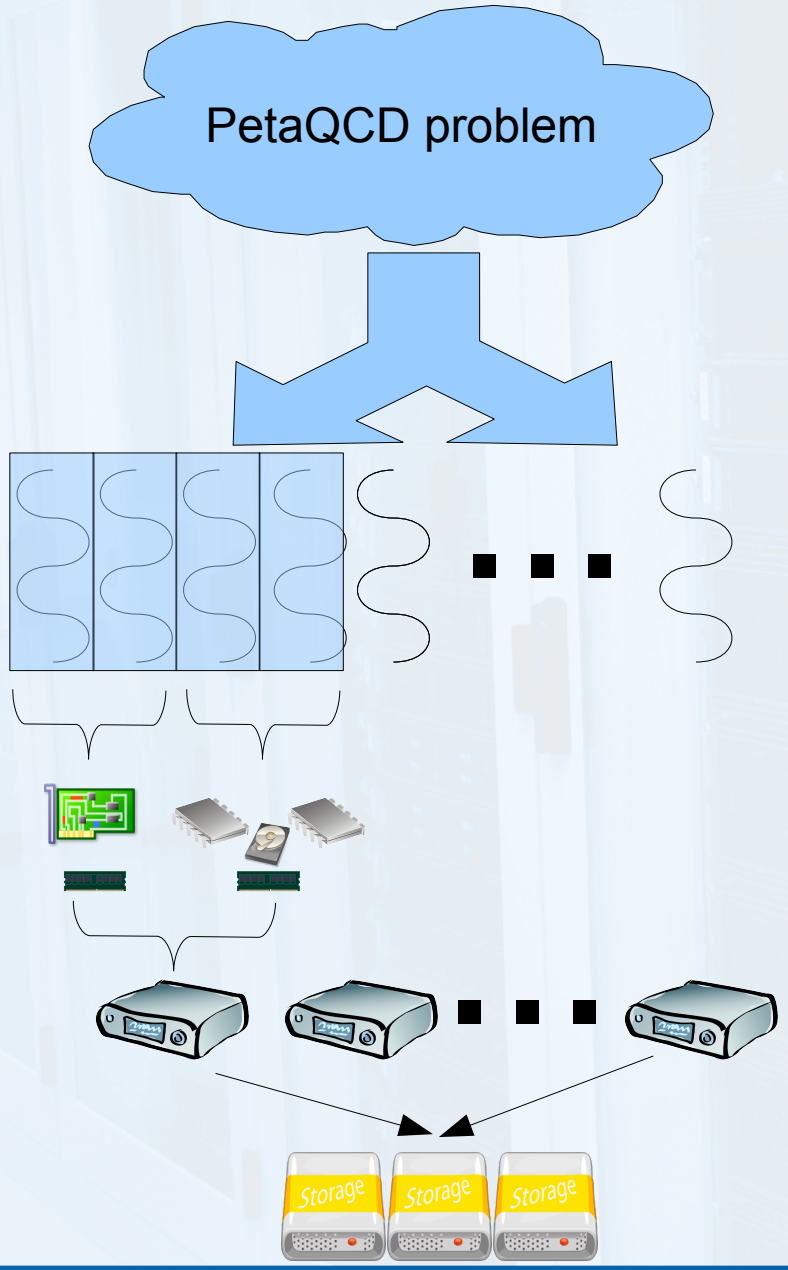
Fault tolerance approaches

- Detection error
  - Application error, computation error, hardware failure, human action
- Fault tolerance policy:
  - Active policy
  - Coordinated checkpoint policy
  - Non coordinated checkpoint policy
    - Checkpoint on a per-process decision
    - Must be able to recreate lost data and replay IOs
    - Uniform use of the checkpoint network
  - Mixed approach

Fault tolerance approaches

- Detection error
  - Application error, computation error, hardware failure, human action
- Fault tolerance policy:
  - Active policy
  - Coordinated checkpoint policy
  - Non coordinated checkpoint policy
  - Mixed approach

Checkpoint granularity



- Everything may be possible
- Fine grain Vs. complexity
- Huge amount of data to store and manage
- Checkpoint frequency ?
- Checkpoint “Time To Live” ?
- Checkpoint network Vs. Computation network ?

Fault tolerance targets

- Need a generic API in order to create, efficiently, the checkpoint of an application, independently from hardware
- Need a generic high availability API in order to allow parts of applications:
  - to continue their job without any interferences (maybe an API dedicated to active replication,
  - to restart and resynchronize their job without any human action.

Fault tolerance questions

- Application and hardware designs have an impact on fault tolerance choice
  - Should we consider fault tolerance as a property of the high level abstract language ?
  - Should we consider dedicated toolbox (per software and hardware environment) used by a code generator ?
- Can we do better than simple checkpoint/restart?
  - Add some active replication?
  - Save less data and rebuild missing one?