Distem: Evaluation of Fault Tolerance and Load Balancing Strategies in HPC Runtimes through Emulation

Cristian Ruiz, Joseph Emeras, Emmanuel Jeanvoine and Lucas Nussbaum
INRIA - France

May 18, 2016 – CCGRID 2016

HPC runtimes

- According to the IESP report a strong effort must be made on improving HPC software stacks
- One of the main parts of this stack is dedicated to HPC runtime
- HPC runtime enables the execution, managing and debugging of parallel applications
- OpenMPI, Charm++, CUDA, etc.

For this work we focus on studying HPC runtimes

Dongarra, Jack *et Al.*, *The International Exascale Software Project Roadmap*, International Journal of High Performance Computer Applications, 2011

Evaluating current HPC runtimes

Several properties to evaluate

- Programmability
- Scalability
- Fault tolerance
- Load balancing

We focus on

- Fault tolerance: more components $\Rightarrow$ shorter MTBF (Mean Time Between Failures)
- Load balancing: Cloud computing, Green computing, Data centers' policies

How people evaluate HPC runtimes?

Some examples

- Ad-hoc simulator and real application trace from IBM BG/Q

Harshitha Menon and L. V. Kalew, *A Distributed Dynamic Load Balancer for Iterative Applications*, SC'2013

- Leveraging DVFS processor capabilities

Osman Sarood *et Al.*, *A 'Cool' Way of Improving the Reliability of HPC Machines*, SC'2013

- For MPI based runtimes, most of the experiments are done using real platforms

Evaluating current HPC runtimes

- Carrying out evaluation under complex realistic conditions is hard
- Simulator:
  - simplified assumptions ☹
  - lower realism ☹
  - not possible to run a complete software stack ☹
- Real platform:
  - expensive ☹
  - lacks of reproducibility ☹

Outline

1. Distem
2. Experimental results
3. Conclusions

An emulator for distributed systems

Take your real application

Run it on a cluster

And use Distem to alter the platform
so it matches the experimental conditions you need

+ Distem →

Heterogeneous nodes
Long distance networks
Faults, perf. variations
Grid, Cloud, P2P features
...

Distem features

The features of Distem include:

- running many virtual nodes on each physical node
- emulation of CPU performance, network topologies, I/O speed

Distem uses modern Linux functionality:

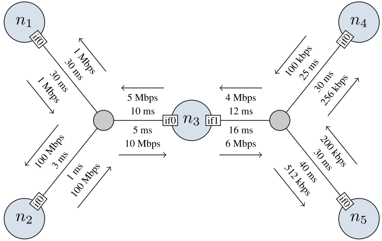
- Linux containers
- control groups
- CPU frequency scaling
- traffic control
- I/O throttling

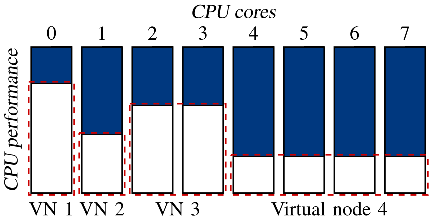
In this work

We integrated the following improvements in order to make possible the evaluation of HPC runtimes:

- Evolving experimental conditions
- Failure injection framework
- Event injection framework

Evolving experimental conditions





- Heterogeneous conditions can be created: CPU frequencies, different IO and network capabilities
- These features can be updated dynamically
- This is useful to achieve complex experiments where the platform is modified, like it could happened in reality

Failure injection framework

- We take into account failures that provoke a lost of the node (very common failures)
- Nodes can be lost in three different ways:
  - Graceful: the node is shut down cleanly, using an operating system command
  - Soft: the node is forced to shut down
  - Hard: the node failed abruptly
- We do not take into account byzantine failures

Event injection framework

- Increase the reproducibility of experiments
- Distem supports the following modifications for a given set of nodes:
    - CPU frequency
    - Network capabilities (latency and bandwidth)
    - Failures
- These modifications can be injected using a deterministic behavior or using a probabilistic distribution

Outline

Experiment setup

- We evaluate Charm++, OpenMPI and MPICH runtimes
- Charm++: Jacobi3D and Stencil3D
- MPI-based runtimes: NAS parallel benchmarks
- 3 Grid'5000 clusters located in two sites
- Experimental evaluation:
  - *Failure detection of HPC runtimes*
  - *Validity of fault injection mechanism*
  - *Evaluation of load balancing strategies in Charm++*

Failure detection of HPC runtimes

- We run an application on top of the HPC runtime
- We inject different types of faults and observe how the HPC runtime reacts

Failure	Runtime					
	Charm++		OpenMPI		MPICH	
	Detected	Action	Detected	Action	Detected	Action
Graceful	Yes	C	Yes	H	Yes	E
Soft	Yes	C	Yes	H	Yes	E
Hard	No	-	Yes	H	Yes	E

Table: Failure detection. C refers to the roll-back of the application to the previous checkpoint, H refers to the fact that processes hang, E refers to the termination of MPI processes

Validity of fault injection mechanism

- We run Jacobi3D application using 64 nodes, running 1 Charm++ process per node
- Different degrees of oversubscription
- We use the event injection framework to inject the same trace for all cases

Mechanism	% termination	Mean walltime (secs)
Charm++ Injection	100%	268.55
Real Injection	66%	267.19
Distem 1vn/node	56%	286.43
Distem 2vn/node	50%	287.05
Distem 4vn/node	56%	294.45

Table: Percentage of successful application executions

Evaluating load balancing strategies in Charm++

- We create a platform composed 128 vnodes distributed over 8 physical nodes.
- We experiment with two different scenarios:
  - Heterogeneous: half of the vnodes have a CPU clock reduced to 50 %
  - Dynamic: the available CPU power of a sub-part of the vnodes is dynamic.

The event injection framework was used to automate the creation of these scenarios

Evaluating load balancing strategies in Charm++

Running Stencil3D using 128 processes in the heterogeneous platform

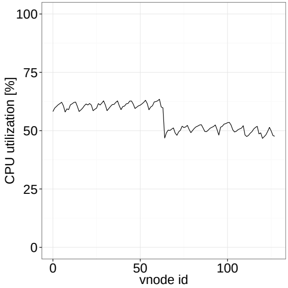


Figure: LBOff
Walltime: 341 secs

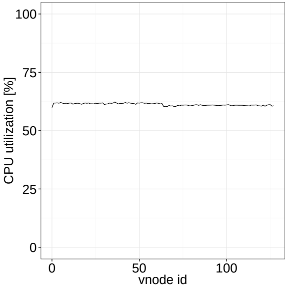


Figure: RefineLB
Walltime: 320 secs

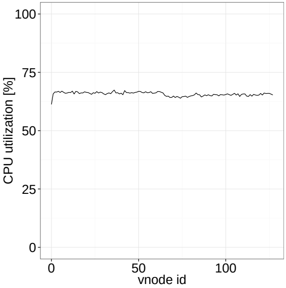


Figure: Hybrid
Walltime: 356 secs

Evaluating load balancing strategies in Charm++

Running Stencil3D using 128 processes in the dynamic platform

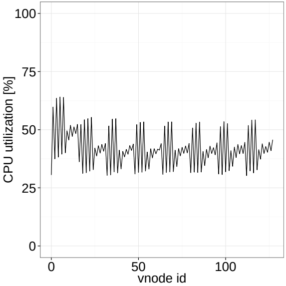


Figure: LBOff
Walltime: 347 secs

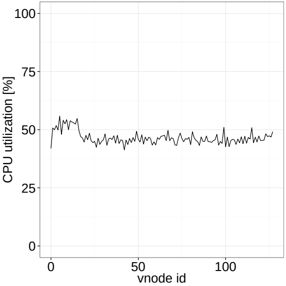


Figure: RefineLB
Walltime: 322 secs

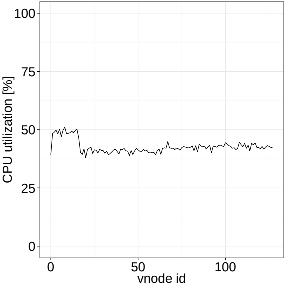


Figure: Hybrid
Walltime: 359 secs

Outline

Conclusions

- Being able to execute experiments on a large set of platform configurations in a repeatable way is a sound basis to design and improve the HPC runtimes in the future

- Distem:
  - offers realistic experimental conditions
  - simplified the uncovering of problems in the failure handling for widely used HPC runtimes
  - enables experimenters to easily simulate perturbations and heterogeneity of nodes