# ADT COSETTE


Cristian Ruiz

MADYNES Team


May 31, 2016 – young engineers seminar

# Validation in Computer Science

Two classical approaches for validation:

- Formal: equations, proofs, etc.
- Experimental, on a scientific instrument.

**In reality**

Given the size and complexity of distributed systems, it is difficult to carry out complete analytic studies. So, the experimental approach is commonly seen.

**Need of**

Reproducible research

# Grid'5000

A large-scale, shared testbed supporting high-quality, reproducible research on distributed systems:

- Highly configurable
- High Performance Computing, Grids, Peer-to-peer systems, Cloud computing

Another examples of testbeds: Chameleon, Cloudlab

# Goal of the ADT COSETTE

Conceive, consolidate and extend a set of tools aimed at experimenting with distributed systems (Cloud, Grid, HPC, P2P)

## Tasks

- Development of Ruby-Cute, a library that gathers useful procedures for experimenting with distributed systems
- Extend Distem to meet Cloud and HPC research requirements

## Supervised by

Lucas Nussbaum, Emmanuel Jeanvoine

# Outline

# Distem

## 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

# Outline

# 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

# Evaluating current HPC runtimes

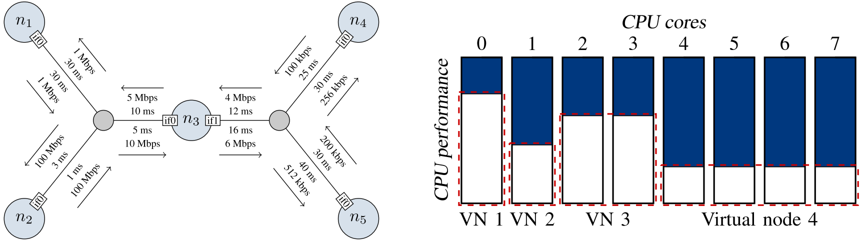
- Carrying out evaluation under complex realistic conditions is hard
- Simulator:
  - simplified assumptions ☹
  - lower realism ☹
  - not possible to run a complete software stack ☹
- Real platform:
  - expensive ☹
  - lacks of reproducibility ☹

# In this task

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

# 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*
  - *Evaluation of load balancing strategies in Charm++*
  - *Validity of fault injection mechanism*

# 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

| | Runtime | | | | | |
|---|---|---|---|---|---|---|
| **Failure** | **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

# 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

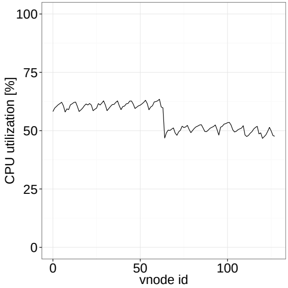



Figure: LBOff
Walltime: 341 secs

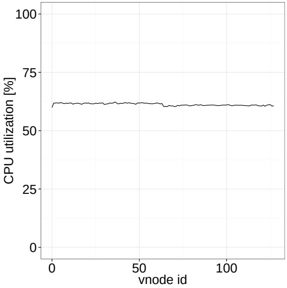



Figure: RefineLB
Walltime: 320 secs

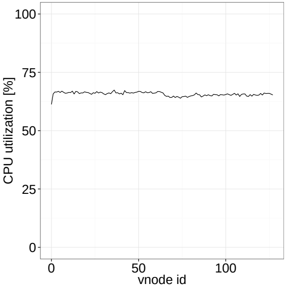



Figure: Hybrid
Walltime: 356 secs

# Evaluating load balancing strategies in Charm++

Running Stencil3D using 128 processes in the dynamic platform

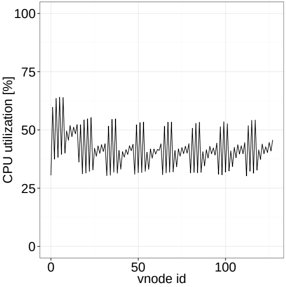



Figure: LBOff
Walltime: 347 secs

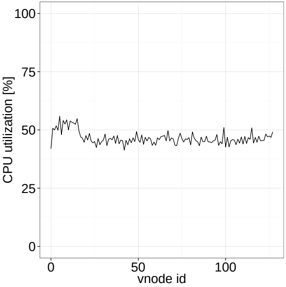



Figure: RefineLB
Walltime: 322 secs

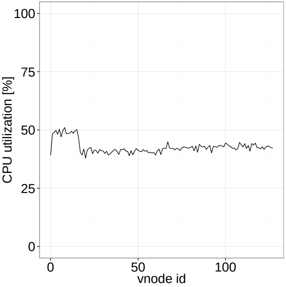
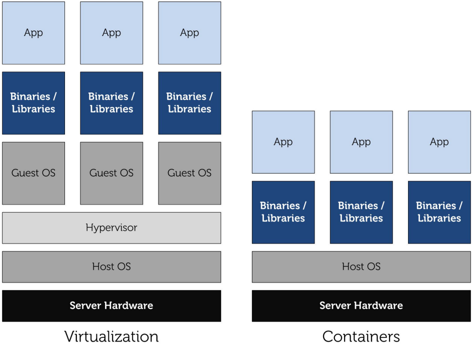



Figure: Hybrid
Walltime: 359 secs

# Outline

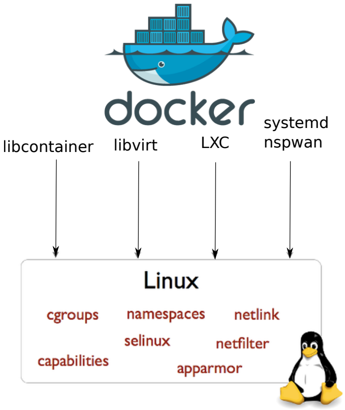
# Containers

Containers refers generally to Operating-system-level virtualization, where the kernel of an operating system allows for multiple isolated user-space instances.



# namespaces and cgroups

- Both features incorporated in Linux kernel since 2006 (Linux 2.6.24)
- Several container solutions: LXC, libvirt, libcontainer, systemd-nspawn, Docker



# In this task, we answer:

- What is the overhead of oversubscription using different versions of Linux kernel?
- What is the impact of running an HPC workload with several MPI processes inside containers?
- What is the impact of network technology ?

# Experimental setup

## Hardware

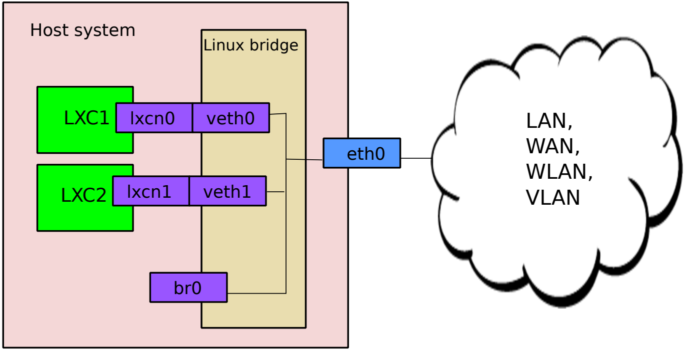
- Cluster in Grid'5000 Testbed where each node is equipped with two Intel Xeon E5-2630v3 processors (with 8 cores each), 128 GB of RAM and a 10 GbE adapter
- Our experimental setup included up to 64 machines

## Software

- Debian Jessie, Linux kernel versions: 3.2, 3.16 and 4.0, OpenMPI and NPB. We instrumented the benchmarks: LU, EP, CG, MG, FT, IS using TAU

# Network setup

- Veth pair + Linux bridge
- Veth pair + OpenvSwitch
- MACVLAN or SR-IOV
- Phys

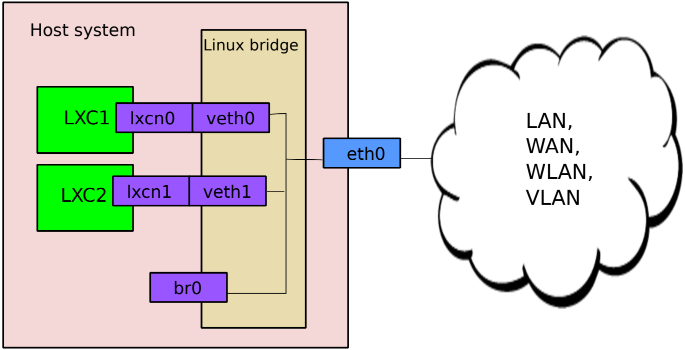


# What did we find?

- There is important performance degradation provoked by veth for Linux kernels $< 3.11$. Running 2 containers per physical machine:
  - 3.2: 1577.78%
  - 3.16: 22.67%
  - 4.0: 2.40%
- Overhead present in MPI communication
- Since Linux kernel version 3.11, TSO was enabled in veth
- Memory bound applications and application that use all to all MPI communication are the most affected by oversubscription
- Performance issues can appear only at certain scale (e.g. 180% overhead with 128 MPI processes for CG benchmark).

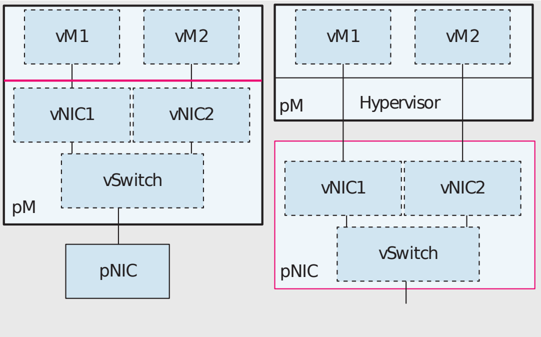
# Interconnection comparison

- Veth pair + Linux bridge
- Veth pair + OpenvSwitch
- SR-IOV
- Using Linux kernel 4.3



# SR-IOV explained

- Veth pair + Linux bridge
- Veth pair + OpenvSwitch
- SR-IOV

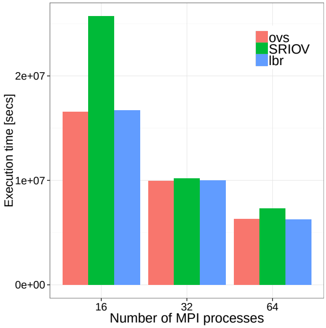



Network Virtualization and Software Defined Networking for Cloud Computing: A survey
Raj Jain and Subharthi Paul, Washington University

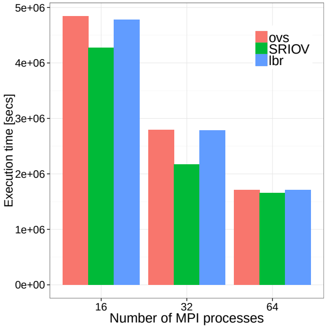
# Multi-machine (4 nodes)

- 4 containers per machine
- Each container configured with 4 cores

(a) LU Class B




(b) EP Class B



# What did we find?

- Using one machine there is a significant overhead of SR-IOV
- Multi-machine setups results varies from benchmark to benchmark there is no significant improvement of using SR-IOV
- Communication of container in the same machine are really affected using SR-IOV
- OpenVSwitch presents the best performance

# 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

# What did we find?

- There is important performance degradation provoked by veth for Linux kernels $< 3.11$
- Container placing plays an important role under oversubscription
- Memory bound applications and application that use all to all MPI communication are the most affected by oversubscription
- More details about the results:

Cristian Ruiz *et Al.*, *Performance Evaluation of Containers for HPC*,11th Workshop on Virtualization in High-Performance Cloud Computing, Vienna, Austria,2015