STRESS CLOUD
A VIRTUALIZED INFRASTRUCTURE EVALUATION TOOL

Guillaume Le Louët, Jean-Marc Menaud École des Mines de Nantes, INRIA, LINA
Huge increase of datacenters' numbers, size and power consumption
- 2011-2012: +22% invested. 2007-2011 +100%W consumption
- Services virtualization (VM): complex management
- Virtual Machine Manager (VMM) to help the administrator.
  - monitor the datacenter (virt-manager)
  - analyses the observations (VCOps, SolarWinds)
  - deduces actions (VMWare DPM, Entropy)
- IaaS level: infrastructure as a Service
- How an administrator can evaluate those VMMs?
EVALUATE ON SIMULATOR

• Simulator of data centers
  ■ Cloudsim (http://code.google.com/p/cloudsim/)
• Modeling more and more complex
  ■ Numerous and significant technological advances
    ○ (Content base sharing, ballooning, memory compression swap etc.)
  ■ Complex Systems: power, heat, requests response time
• Need for an evaluation on a real datacenter.
LOAD INJECTORS ON REAL DATACENTER

- Evaluate on real infrastructure with load injection?
  - RUBIs (http://rubis.ow2.org/)
  - The Grinder (http://sourceforge.net/projects/grinder/)
  - CLIF (http://clif.ow2.org/)
- The VMMs base their analyses only on systems metrics
  - No VM system resources load injector
  - StressCloud: a virtualized infrastructure evaluation tool
REQUIREMENT OF SYSTEM LOAD INJECTION

- Stress each VMs' system resources
  - CPU, storage, network
  - extensibility to other resources
- Observe stress execution.
  - effective activities modifications requests
  - requested activities' performances
REQUIREMENT OF IAAS LOAD INJECTION

- Lots of VMs to manipulate
  - Mass manipulation is mandatory
  - Registration of the VMs on one server
- Language to describe Load injection scenarios
  - Select scenario's VMs
  - set VMs activities
  - synchronize requested activities
  - deterministic execution of the scenario
STRESSCLOUD ARCHITECTURE

- In a IaaS center
- One main central server, the Registrar
- Each manipulated VM connects to the registrar
- The user executes the scenario on the registrar
- The registrar handles
  - transmission layer
  - loads synchronization
  - performances retrieval
SCRIPTING LANGUAGE

• Select the VMs using their virtual environment
• Manipulate VMs activities
  ■ usage in % of host has no meaning in IaaS.
  ■ Webserver loads are accesses per seconds.
  ■ HPC works are number of accesses to perform.
• Observe performances
  ■ list the requested works and loads
  ■ works present their execution speed
  ■ loads present their error rate
**WEBSERVER SCENARIO**

- VMs activities are loads specified in accesses to perform per second.
- The requested loads are synchronized using the time.

*web scenario sample:*

```python
# Example in Python

# Group 1
mem = 1024
cores = 3

# Group 1 activities
for instance in group1:
    instance.cpu = 1000
    group1.release()

# Group 2 activities
for instance in group2:
    instance.disk = 500
    group2.release()
```

```java
// Example in Java

// Group 1
mem = 1024
cores = 3

// Group 1 activities
for (Instance instance : group1) {
    instance.setCPU(1000);
    instance.release()
}

// Group 2 activities
for (Instance instance : group2) {
    instance.setDisk(500);
    instance.release()
}
```

```javascript
// Example in JavaScript

// Group 1
mem = 1024
cores = 3

// Group 1 activities
for (let instance of group1) {
    instance.setCPU(1000);
    instance.release()
}

// Group 2 activities
for (let instance of group2) {
    instance.setDisk(500);
    instance.release()
}
```

```c
// Example in C

// Group 1
mem = 1024
cores = 3

// Group 1 activities
for (Instance instance : group1) {
    instance.setCPU(1000);
    instance.release()
}

// Group 2 activities
for (Instance instance : group2) {
    instance.setDisk(500);
    instance.release()
}
```
HPC SCENARIO

- VMs activities are works specified as total amount of accesses to perform.
- The requested works are synchronized by sequencing beginning and end of works.

HPC scenario sample:

```plaintext
group1=require(null, 5)
last=require('mem==10000')

group1.each{it.send(last, 500)}
last.after(group1).cpu+30000
sync(last)
release()
```
STRESSERS IMPLEMENTATIONS

• Generic stresser implemented: CPU, storage, network.
• Periodic resources access loop.
• Scheduling errors observed to deduce error rates.
• Stressers evaluated outside of the framework: induced CPU load, error rate monitoring.
• Semantic issues for activities depending on the server's physical architecture.
LET’S SUM UP

- Framework to inject load in IaaS.
- Distributed load scenario executed in VMs.
- Extensible stressers model.
- Open-source (LGPL-v3)
- Main framework in Java, stressers in Java
- Language based on Groovy
FUTURE WORKS:

- Evaluate VM scaling.
- Implement other resources stressers: GPU, SAN.
- Enhance stressers (change language?)
- From activity traces to scenario?