Energy efficiency in OpenStack clouds

François Rossigneux
francois.rossigneux@inria.fr

January 28, 2013 - Université du Luxembourg
Summary

Context

Telemetry architecture

Scheduling / sleep modes (future works)
Summary

Context

Telemetry architecture

Scheduling / sleep modes (future works)
Context

XLcloud:
- HPC-as-a-Service (based on OpenStack)
- Funded by the "Fonds national pour la Société Numérique"
- Three-year long collaborative project
- Open source license

Some features:
- GPU virtualization
- Green scheduling
- Power consumption based billing
Context

Consortium:
Our team is working on energy topics:
- Telemetry (taking measurements)
- Scheduling (placing virtual machines)
- Turning off unused machines (sleep modes)
Summary

Context

Telemetry architecture

Scheduling / sleep modes (future works)
Telemetry architecture
OpenStack overview

OpenStack main components:
- Compute (Nova)
- Object Storage (Swift)
- Block Storage (Cinder)
- Networking (Quantum)
- Identity (Keystone)
- Dashboard (Horizon)

Recently added:
- Metering / billing (Ceilometer)

Incubation:
- Energy (Kwapi)
OpenStack main components:
- Compute (Nova)
- Object Storage (Swift)
- Block Storage (Cinder)
- Networking (Quantum)
- Identity (Keystone)
- Dashboard (Horizon)

Recently added:
- Metering / billing (Ceilometer)

Incubation:
- Energy (Kwapi)
Telemetry architecture
OpenStack overview

OpenStack main components:
- Compute (Nova)
- Object Storage (Swift)
- Block Storage (Cinder)
- Networking (Quantum)
- Identity (Keystone)
- Dashboard (Horizon)

Recently added:
- Metering / billing (Ceilometer)

Incubation:
- Energy (Kwapi)
Telemetry architecture
Datacenter overview

OpenStack (Ceilometer, Nova)
Kwapi API
Kwapi Drivers

Datacenter Paris

IPMI
Server 1
VM
VM

Datacenter Lyon

Kwapi Drivers

Server 2
VM
VM

Server 3
VM
VM

PDU

Datacenter Marseille

IPMI
Server 4
VM
VM

Power

internet

A
B
C
D
Telemetry architecture
Software layers

- OpenStack
  Nova scheduler

- Ceilometer

- Kwapi plugins
  Bus
  Kwapi drivers

- Wattmeters
Telemetry architecture
Software layers

- OpenStack Nova scheduler
- Ceilometer
- Web browser
- Ceilometer plugin
- Visualization plugin
- Bus
- Kwapi drivers
- Wattmeters
Telemetry architecture
Drivers layer

Hardware

Wattmeters
Wattsup
Wattsup
Omega

Servers
A
B
C
D
Telemetry architecture
Drivers layer

Drivers

Hardware

Wattmeters

Servers

USB

USB

SNMP

Thread

Thread

Thread

Wattsup

Wattsup

Omega

A

B

C

D
Telemetry architecture
Drivers layer

Drivers

ZMQ device

Threads

USB

USB

SNMP

Hardware

Wattmeters

Wattsup

Wattsup

Omega

Servers

A

B

C

D
Telemetry architecture
Drivers layer
Telemetry architecture
Bus frameworks

ZeroMQ (used in Kwapi):
- Very fast
- Small (1.6 Mo)
- Written in C++ (provide a Python wrapper)
- Socket types: inproc, ipc, tcp
- Reliable / preserves order of messages
- Simple to use design patterns (publish/subscribe, request/response, ...)
- Brokerless

RabbitMQ (used in OpenStack):
- Much more slower (10x)
- Require Erlang (70 Mo)
- Broker

Sockets (without framework):
- Why reinvent the wheel?
Telemetry architecture
ZeroMQ design pattern

Publish/subscribe design pattern

- Publishers
- Subscribers
- Plugin
- Plugin
- Driver thread

tcp://0.0.0.0:8000
Telemetry architecture
ZeroMQ design pattern

Publishers and subscribers need common endpoints
Telemetry architecture
ZeroMQ design pattern

Forwarding device:
- Subscribes to inproc://drivers
- Publishes all received packets on tcp://140.77.13.25:8000
Subscribers can listen multiple endpoints
Telemetry architecture
Bus messages format

Python dictionary:

<table>
<thead>
<tr>
<th>Probe ID</th>
<th>Payload (watts, volts, amperes...)</th>
<th>Signature</th>
</tr>
</thead>
</table>

Three mandatory fields:
- Probe ID
- Watts
- Signature

Signature based on a shared secret key
Telemetry architecture
Ceilometer overview

Nova scheduler

Ceilometer
- API
- Collector
- Central agent

Pollster

Ceilometer plugin
- API
- Collector (compute kWh)

ZeroMQ bus
Telemetry architecture
API plugin

Collector:
- Collects power consumption data
- Computes kWh and stores the last value (watts)

API (based on Flask):
/v1/probe-ids/ The list of probe ids
/v1/probes/ All detailed information about all probes
/v1/probes/A/ Detailed information about probe A
/v1/probes/A/kwh Energy consumed by probe A

Authentication:
- The pollster provides a token (X-Auth-Token)
- The plugin checks the token (Keystone request)
- If the token is valid, requested data are sent
Telemetry architecture
Ceilometer pollster

Pollster:
- Is run periodically by Ceilometer central agent
- Asks to Keystone the Ceilometer plugin address
- Retrieves data
- Publishes kWh and watts counters

Collector stores published counters

API is queried by the Nova Scheduler to make a placement decision
Telemetry architecture
Visualization plugin
Telemetry architecture
Visualization plugin

Writes power consumption into RRD files:
- Several archived periods with different resolutions
- RRD file size = 10 K\(\text{o}\) (1000 probes = 10 M\(\text{o}\))

Webpage based on Flask:
- Two visualization modes (per periods and per probes)
- Summary graphs
- Cache mechanism (rebuild graph only if outdated)
Telemetry architecture
Visualization plugin

API example

/graph/minute/A
Telemetry architecture
Visualization plugin

API example

/graph/day/
Summary

Context

Telemetry architecture

**Scheduling / sleep modes (future works)**
Scheduling
Choosing the greenest place

Where is the greenest place to run your job?

It depends on your job:
  CPU / GPU / memory / storage / network intensive ?
  Hard to estimate: vary over time, external events...

Approach: use a benchmark for efficiency rating.
Scheduling
Nova scheduler

Hosts chosen after filtering and sorted after weighting (here the best variant is Host 5, the worst – Host 6)
Scheduling
Nova scheduler

- Hosts from the pool of hosts
- Costs of the hosts capabilities relative to the request specifications
- Weights – sums of costs
- Sorted list of hosts

- Host 1: Cost x 3 + Cost x 4 = Weight 1 = 12
- Host 2: Cost x 2 + Cost x 3 + Cost x 2 = Weight 2 = 87
- Host 3: Cost x 3 + Cost x 2 + Cost x 2 = Weight 3 = 23
- Host 4: Cost x 2 + Cost x 2 = Weight 4 = 10
- Host 5: Cost x 3 + Cost x 2 = Weight 5 = 56
- Host 6: Cost x 3 + Cost x 2 + Cost x 3 = Weight 6 = 40

Hosts 2, 1, 3, 6, 5, 4
Using power saving modes:

- Which mode to choose?
  => Standby / hibernation

- How many machines should be turned off?
  => Anticipating demand and avoiding frequent shutdown / start-up cycles

- How much energy does it save? Is it profitable?
  => Peak start-up power

- Avoiding too frequent shutdown / start-up cycles
  => Sparing the old computers, but they are the least efficient ones
Conclusion

Telemetry:
- Writing more drivers
- Improving scalability

Scheduling and sleep modes:
- Implementing the strategies
- Live VM migration

Measuring energy with wattmeters is not all:
- What about the energy needed to build or recycle the servers?
- What about PUE (on a distributed architecture?)
Thank you for your attention