Cloudy systems

— Taking the most out of the HPC Cloud

Ander Astudillo <ander.astudillo@surfsara.nl>
Markus van Dijk <markus.vandijk@surfsara.nl>

Workshop at UvA
14th June 2018
Recap: defining cloud computing

Essential characteristics:
- On-demand **self-service**
- Broad **network access** (ubiquitous + convenient + on-demand)
- Resource **pooling**
- Rapid **elasticity**
- Measured **service**

Service models:
- Software as a Service (SaaS)
- Platform as a Service (PaaS)
- Infrastructure as a Service (IaaS)

Examples?
But why…?

...scaling
- Sequential run takes forever
- Not enough local resources (e.g.: memory)
- Analyse more data
- Achieve higher accuracy
- ...

...elasticity
- Booking fixed resources in advance is:
  - A waste
  - Too expensive
  - Unpredictable
  - ...

Examples?
Some programs are already parallel

- The end-user just needs to run them
- E.g.: Delft3D, XBeach, OpenFoam, Matlab…

Some problems are a matter of running the same thing (possibly) with different parameters

- You can simply run many of these runs independently at the same time on different computers
- E.g.: a Monte Carlo simulation
Agenda

1. - Scaling possibilities
2. - API overview
3. - Demo
Scaling possibilities
Your **application** may need more…

- **Scale up** vs. **Scale out**
The concept (II)

1. Scaling possibilities

*e.g.* transport people may need more room...

Scale up vs. Scale out
The concept (and III)

e.g. transport people may need more room...

Scale up AND Scale out
Some theory (I)

Meet: the CPU

1.- Scaling possibilities
Meet: **parallel** processing

How does this **scale** anything?
Dividing work (I)

Parallelism: **task** partitioning

- **work to do**
  - **compute**
  - **data processing**

1. Scaling possibilities

```
\[ \begin{array}{c}
\text{i/o} \\
\text{m} \\
\text{task 1} \\
\text{i/o} \\
\text{m} \\
\text{task 2} \\
\vdots \\
\text{i/o} \\
\text{m} \\
\text{task n}
\end{array} \]
```
Dividing work (II)

Parallelism: **data** partitioning

- **work to do**
- **compute**
- **data processing**

1. Scaling possibilities

- **dataset 1**
- **dataset 2**
- **dataset n**
Dividing work (and III)

Example: a possible parallel program (or workflow)

1. Scaling possibilities
Technique: shared memory

Processes only communicate through the shared memory

e.g.: OpenMP
Parallel programming (II)

Technique: message-passing

Processes can communicate directly

I/O

CPU

Mem

1.- Scaling possibilities
Combinations: shared memory and message passing
Parallel programming (and IV)

Combinations: multiple physical machines

Host 1

Host 2

Host 3

Host n

network

1.- Scaling possibilities
IaaS: Your place to run VMs

1.- Scaling possibilities

Images

- Data store
- Persistency
- ...

Template

- CPU
- RAM
- I/O
- Disks
- Network
- ...

VMs

Instantiate
IaaS: your interconnected VMs

1.- Scaling possibilities
Example: OpenMP

1.- Scaling possibilities
1.- Scaling possibilities
Example: MPI

1. Scaling possibilities

Private network
Some thoughts

Parallel programming can be tricky:

• Need to know your algorithm
• Need to know your data
• Need to know your architecture

Try to optimise:

• Identify sequential bottlenecks
• Strive for data locality
• Identify latencies
• Minimise communication
• Be wary of concurrency:
  • Deadlocks
  • Race conditions
• Prepare for failures: machines, networks, timeouts…

So… you may as well be better off using a naïve approach! 😊
API overview
Why automation?

pets vs. cattle

• Pets are given names like pussinboots.cern.ch
• They are unique, lovingly hand raised and cared for
• When they get ill, you nurse them back to health

• Cattle are given numbers like vm0042.cern.ch
• They are almost identical to other cattle
• When they get ill, you get another one

• Future application architectures should use Cattle but Pets with strong configuration management are viable and still needed

Borrowed from @randybias at Cloudscaling
http://www.slideshare.net/randdybias/the-cloud-revolution-cyber-press-forum-philippines

Gavin McCance, CERN
XML-RPC over http

- bindings for Java, Ruby (also Python, NodeJS…)

**Methods** like `one.<object>..<action>`
- e.g.: `one.vm.rename`
- *Pools*, like: `one.vmpool.info`

**Parameters**, position-based

**Output**, a 3-tuple (A, B, C) where:
- A: correct or error response
- B: returned info (if correct);
  - error message (if error)
- C: numeric error code

Operate/query on:
- Images
- Templates
- Virtual Machines
- Quotas
- …
Demo
Example (I)

```python
class VmList:
    """A simple list of my VMs"""
    ONE_ENDPOINT = 'http://ui.hpccloud.surfsara.nl:2633/RPC2'
    ONE_USER = 'username'  # replace this with yours
    ONE_PASS = 'pass'  # replace this with yours
    def __init__(self):
        self.client = oca.Client(
            self.ONE_USER + ':' + self.ONE_PASS, self.ONE_ENDPOINT)
    def fetch_vms(self):
        xml_string = self.client.call('vmpool.info', -3, -1, -1, -2)
        root = ET.fromstring(xml_string)
        return root

if __name__ == '__main__':
    xml = VmList()
    .fetch_vms()
    print(XmlUtil.prettify(xml))
```

Example (and II)

```
<VM_POOL>
  <VM>
    <ID>164</ID>
    <UID>247</UID>
    <GID>108</GID>
    <UNAME>ander</UNAME>
    <GNAME>workshop</GNAME>
    <NAME>Ubuntu-15.04</NAME>
    ...
    <LCM_STATE>3</LCM_STATE>
    <TEMPLATE>
      <CPU>...</CPU>
      ...
    </TEMPLATE>
  </VM>
  <VM>...</VM>
  ...
</VM_POOL>
```
Credits

Images: Wikipedia, Science Park, RRZE icons, NIST, nVidia, Ceph, publicdomainpictures.net, publicdomainvectors.org, cs.unc.edu/~weicheng
Slides: SURFsara colleagues, CERN

Ander Astudillo
<ander.astudillo@surfsara.nl>

Markus van Dijk
<markus.vandijk@surfsara.nl>

Request: https://e-infra.surfsara.nl
UI: https://ui.hpccloud.surfsara.nl
Doc: https://doc.hpccloud.surfsara.nl
Amdahl’s law

\[ T(s) = (1 - p)T + \frac{p}{s}T. \]

- \( T(s) \): running time after an improvement of \( s \)
- \( s \): speedup factor of parallel part
- \( p \): % of the program that is parallel
- \( T \): original running time
- \( W \): fixed workload

\[ S_{\text{latency}}(s) = \frac{TW}{T(s)W} = \frac{T}{T(s)} = \frac{1}{1 - p + \frac{p}{s}}. \]

It’s mainly the algorithm that defines speedup; rather than the amount of processors.

Speedup is limited by the serial part of the program. E.g., if 95% of the program can be parallelised, the theoretical maximum speedup using parallel computing would be 20 times.
Processes communicate over the fabric.