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
30th January 2019
Recap: defining cloud computing

Essential characteristics:

- On-demand **self-service**
- Broad **network** access (ubiquitous + convenient + on-demand)
- Resource **pooling**
- Rapid **elasticity** ⇒ 1. Scaling  
  2. API
- **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**
- **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
Some theory \((\text{and II})\)

Meet: \textbf{parallel} processing

- \textbf{I/O}
- \textbf{CPU}
- \textbf{Mem}
- \textbf{Mod}

How does this \textbf{scale} anything?
Dividing work (I)

Parallelism: **task** partitioning

1. Scaling possibilities
Dividing work (II)

Parallelism: **data** partitioning

![Diagram showing data partitioning](image)

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

1. Scaling possibilities

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

Example: a possible parallel program (or workflow)
Parallel programming

Technique: **shared memory**

- e.g.: OpenMP

Technique: **message-passing**

- e.g.: MPI

Very complex to do yourself!

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

1. Scaling possibilities

Images
- Data store
- Persistency
- ...

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

Instantiate

VMs
IaaS: your interconnected VMs

1. Scaling possibilities

Private network

Node 1

Node 2

Node n

Internet
IaaS: master-workers set-up

**e.g.: workers**

- Node 1
- Node 2
- Node n

**Private network**

**e.g.: master**

- Node n

Each *worker* receives work to do from the master and actually does the heavy lifting.

The *master* orchestrates work to/from each of the workers.

1. Scaling possibilities
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! 😊
Principal Component Analysis (PCA)
in a nutshell

We want to reduce the dimensionality of the data without losing variance.

- https://gmaclenn.github.io/articles/airport-pca-analysis/

1.- Scaling possibilities
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 @randyb in Cloudsc aling
http://www.slideshare.net/randybias/the-cloud-revolution-cyber-press-forum-philippines

Gavin McCance, CERN
Description

OpenNebula

**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
class VmList:
    """A simple list of my VMs"""
    ONE_ENDPOINT = 'http://ui.hpcccloud.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)

List my VMs (output)

```xml
<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>
```
Request: https://e-infra.surfsara.nl
UI: https://ui.hpccloud.surfsara.nl
Doc: https://doc.hpccloud.surfsara.nl

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>
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

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.