Self-Adaptive Multi-Cloud Applications

Brice Morin
SINTEF
Cloud and Big Data ecosystems are dynamic and complex

- Many cloud providers, with offers changing over the time
  - Location of data
  - Pricing models
- Many services "doing the same things"
  - With different QoS
  - At different prices
- Many types of data sources
  - Social networks (typically "fuzzy")
  - Sensors (typically "binary")
- Many dependencies
  - Some services might require some other services
  - Some services might require to run on a given cloud provider
- VMs or services might fail
Challenges

• Not tractable to try to accommodate this complexity at design-time
  – Combinatorial explosion of the number of possible configurations (assemblies of services) for real multi-cloud applications

• Not satisfactory to just crash at runtime when facing a situation not validated at design-time

• Not sustainable to pre-book a large number of VMs and pre-deploy a large number of alternative services
A simple example

- SilverOrbi
- SmallClusteredAS
- SmallMonoliticDB
+ GoldenOrbi
+ LargeElasticAS
+ LargeClusteredDB
One component failing $\rightarrow$ 6 operations to fix the system

- What if the system operator just changed the failing component
  - From MediumClusteredAP to LargeElasticAP
  - $\rightarrow$ yields an invalid configuration, the system is likely not to work properly

- We need tools to
  - Reason about failures (occurred or **impending**)
  - Automatically enact decisions
Abstractions to support reasoning (1/2)

High-level context information:
- not everything you can possibly monitor
- but rather what makes to support reasoning
Can be obtained by monitoring infrastructure and services
+ Complex Event Processing (CEP)

Context information can be used to describe basic/reflex rules
Abstractions to support reasoning (2/2)

QoS properties

When to optimize what

Impact of services on QoS

<table>
<thead>
<tr>
<th>Service</th>
<th>Cost</th>
<th>Usability</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAP Service (MAP_S)</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>Google Map (GMAP_S)</td>
<td>MEDIUM</td>
<td>High</td>
</tr>
<tr>
<td>Bing Map (BMAP_S)</td>
<td>LOW</td>
<td>High</td>
</tr>
<tr>
<td>Virtual Machine (VM)</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>Large (L)</td>
<td>HIGH</td>
<td>High</td>
</tr>
<tr>
<td>Medium (M)</td>
<td>MEDIUM</td>
<td>MEDIUM</td>
</tr>
<tr>
<td>Small (S)</td>
<td>LOW</td>
<td>MEDIUM</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rule</th>
<th>ID</th>
<th>context</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low</td>
<td>LOW</td>
<td>CPU = HI or RAM = HI</td>
</tr>
<tr>
<td>High</td>
<td>HIGH</td>
<td>CPU = LOW and RAM = LOW</td>
</tr>
<tr>
<td>Med</td>
<td>MED</td>
<td>CPU = MED or RAM = MED</td>
</tr>
<tr>
<td>Unstable</td>
<td>UNSTABLE</td>
<td>GMAP = NOK or BMAP = NOK or GMAP = FAIL or BMAP = FAIL</td>
</tr>
</tbody>
</table>
Optimization problem

In a real system, the problem is not limited to a 2-axis optimization problem
Notes

• Can be interpreted
  – Gives more flexibility e.g. new rules can be added at runtime, transparently for the reasoner
  – Can be slow in some use cases

• Can be compiled
  – First into a state machine model, then into Java, JS or C code (based on ThingML.org)
  – Faster
  – Less flexible as we need to regenerate/re-compile code (need to embed code generators and compilers at runtime)
  – Less scalable as we need to (automatically) enumerate all possible configurations
Abstractions to support enactment: CloudML

- Two main components:
  1. A **modelling environment** with a tool-supported domain-specific modelling language (DSML) to model the provisioning and deployment of multi-cloud systems
  2. A **models@run-time environment** for enacting the provisioning, deployment and adaptation of these systems
How both languages fit together?

Atomic actions for the first (high level) language, e.g. SmallClusteredAS, can translate into complex scripts for the second language, e.g. including the provisioning of new nodes, the deployment of components, etc.

Combining several of these complex script is not trivial
Conclusion

- Cloud applications are not just big, monolithic binaries that execute in the cloud
- Cloud applications should be able to adapt to their context, both
  - Short term, e.g. how to react if memory get saturated
  - Long term, e.g. how to accommodate changes in Amazon's terms and conditions, or pricing
- Abstraction and tool support is the key
  - Should be human readable, to include operators in the loop when needed
  - Should be machine processable
Credits

MODA Clouds

REMiCS

Broker@Cloud

PAA SAGE