Promise Theory: A Promising Approach to Convergent Management of Virtual Machines

Dong Wu

Oslo University College

June 9, 2010
Outline

1. Introduction
2. Background
3. Convergence Principle
4. Promises of Virtualization
5. Model Scenarios
6. Discussion
7. Conclusion
Motivation and Problem Statement

1 Motivation

- Popularity of Virtualization
- Challenge of Management

2 Problem Statement

"In order to manage VMs automatically, how do we find a convergent model for virtual machine management with Promise Theory? " 
Methodology

- Literature Research
- Comparison of Tools
- Approach of Convergence
- Scenarios Modelling
Virtual Machine Management Tools

- Libvirt
- Manage Large Networks (MLN)
- VMware vCenter Server
- Open Source Cluster Application Resources (OSCAR)
- Cfengine (In future)
To find a model, we start from bottom up.

Atomic Operation with VMs:
- Create, Start, Shutdown, Suspend, Resume, Delete, Reboot,
- Backup, Restore, Update, Migrate.

This set of atomic operations are:
- Mechanized
- Non-intelligent
What’s the convergence?

- **Convergence:**

  An operation is convergent if it always brings the VM closer to its ideal state and has no effect if the VM is already in that state.

- **The meta-rules:**

  incorrect state → correct state
  correct state → correct state
Convergent State

- A life-cycle of VM consists of a set of states: nonexistent, created, running, suspended, shutdown.
- Whatever state the VM is in, the convergent operations can lead the VM close to its ideal state:
Convergence Formula

This principle can be summarized by the following formula:

\[
\text{operation} \cdot \text{state} = \text{ideal-state} \\
\text{operation} \cdot \text{ideal-state} = \text{ideal-state}
\]

To simplify the formula above, we can use operator \( \hat{O} \) as operation, \( q \) as an arbitrary state of VM and \( q_0 \) as ideal-state of VM:

\[
\hat{O} \cdot q = q_0 \\
\hat{O} \cdot q_0 = q_0
\]
In order to illustrate the transitions between the states and actions, we can use finite state machine (FSM) to express. For example:

According to convergence formula:

\[
\text{converge} : \\
\text{create } \cdot \text{nonexistent state} = \text{created state} \\
\text{create } \cdot \text{created state} = \text{created state}
\]
In order to make primitive atomic operations convergent, we can add "guards" to them. The "guards" is defined as:

```plaintext
guards : if (VM is already in ideal-state) {
    do nothing;
} else {
    execute convergent operations to lead VM close to its ideal state;
}
```
Convergent and Non-Convergent Operations

1 Convergent operations
   For Example:
   - Start: *Start a vm. Check if it is running before starting it. If it is up, then do nothing, else start it.*

2 Non-Convergent operations
   For Example:
   - Reboot: *Reboot a VM. It always changes state of VM, so this is not convergent operation.*
Convergent properties of different tools

- The "√" means we can guarantee the operation convergent;
- The "×" means the operation is not convergent, or we can not guarantee it convergent.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Libvirt</th>
<th>MLN</th>
<th>VMware</th>
<th>Cfengine</th>
</tr>
</thead>
<tbody>
<tr>
<td>create</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>√</td>
</tr>
<tr>
<td>start</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>√</td>
</tr>
<tr>
<td>shutdown</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>√</td>
</tr>
<tr>
<td>suspend</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>√</td>
</tr>
<tr>
<td>resume</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>√</td>
</tr>
<tr>
<td>delete</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>√</td>
</tr>
<tr>
<td>reboot</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>backup</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>restore</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>migrate</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>√</td>
</tr>
<tr>
<td>update</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
</tbody>
</table>
Distributed cooperation

How system components work together to maintain their convergent state and ideal behaviour?

- Distributed cooperation has to emerge from independent agents;
- Look to ways of understanding our systems better and reduce their complexity before automating;
- Maintenance of convergence is just keeping promise;
- Promise Theory is a way of seeing how cooperation has to work.
What is Promise Theory?

- Promise Theory is a model of voluntary cooperation between individual, autonomous agents who publish their intentions to one another in the form of promises.

- It is a graph theoretical framework for understanding complex relationships in networks, where many constraints have to be met.

- Promise Theory is a strategy for Knowledge Management. This is the essence of Promise Theory.
Promise notation

Promise is like an arow, made by a promiser to a promisee

- a : the promiser that make a promise;
- b : the promisee that receive the promise;
- c : the promise body that describe subject of the promise;
- +c: a declaration to ”give” behaviour from one agent to another;
- -c : a declaration to ”use” behaviour by one agent from another.
What is virtualization really?

Virtualization is Knowledge Management. It is packaging:

- An interface to a technical implementation;
- A layer that makes appearance independent of the underlying implementation;
- The "containment" of implementation that isolates the details from direct contact with users or other virtual machines.
Any system may be partitioned into an internal environment and an external environment, where there is an interface where promises can be made at the boundary between each other.
Building an internal environment is like building a house:

<table>
<thead>
<tr>
<th>Properties</th>
<th>House</th>
<th>Environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unique Id</td>
<td>No.27</td>
<td>15</td>
</tr>
<tr>
<td>Name</td>
<td>Hio</td>
<td>Host</td>
</tr>
<tr>
<td>Space</td>
<td>200m²</td>
<td>20G</td>
</tr>
<tr>
<td>Voltage(CPU)</td>
<td>220V</td>
<td>6</td>
</tr>
<tr>
<td>System (OS)</td>
<td>Educational</td>
<td>Debian</td>
</tr>
<tr>
<td>Floors(Memory)</td>
<td>6</td>
<td>2G</td>
</tr>
<tr>
<td>State</td>
<td>open</td>
<td>running</td>
</tr>
<tr>
<td>Address(location)</td>
<td>Oslo</td>
<td>Master Server</td>
</tr>
</tbody>
</table>
The external environment make the promise about the properties of its internal environment:

```json
{  
    Environments:  
        physical_host::  
            "Virtual_host_id"  
                Hostname ==> host 1  
                Platform ==> xen  
                Disk ==> 20G  
                Memory ==> 2G  
                Cpu ==> 6  
                Network ==> 10.0.0.3  
                Location ==> master server  
                OS ==> debian  
                Sate ==> running  
}
```
The internal environment promises to manage its "virtual resources": files, processes, data:

```
{
    files:
        virtual_host::
            "/path/file"
            perms => p("612");
            owner => root
}  
```
What qualifies as a promise model?

- The properties of environments could also be represented by XML notation or any domain specific language;
- Libvirt describe the properties of virtual machine in XML Format;
- Cfengine is able to integrate directly with libvirt to formulate a convergent model.
Model Design: cfengine-libvirt model

This Model aims to use cfengine, integrated with libvirt, to build and manage internal environments with a convergent, Promise Theory approach.
Scenario 1: State Convergent Management

A user shuts down the running VM accidentally, then cfengine brings it up:
Scenario 2: Resources Convergent Management

A user changes the resources of VM, then cfengine roll it back:
Scenario 3: Files Convergent Management

A user changes the permissions of files in VM, then cfengine changes it back:
Discussion

- Evaluate approach of convergence
- Evaluate cfengine-libvirt model
- Some issues of this project
Evaluate approach of convergence

Efficiency = \frac{\text{Maintainance Time}}{\text{The number of VMs}}
Evaluate approach of convergence

**Scalability** = \( \frac{\text{The number of VMs}}{\text{The number of operations}} \)

![Maintenance Scalability graph](image-url)
Promise Theory: A Promising Approach to Convergent Management of Virtual Machines

Dong Wu

Introduction
Background
Convergence Principle
Promises of Virtualization
Model Scenarios
Discussion
Conclusion

Evaluate approach of convergence

Reliability

Goal alignment

Percentage of the number of VM on convergent state (%)

Time (Minutes)

Cfengine
Libvirt
MLN
Evaluate approach of convergence

Predictability

- we manage the process or changes with Libvirt and MLN;
- we manage the knowledge with cfengine, which are the intended outcome of the promiser’s actions.
- cfengine has better ”assurance of state” .
Feature comparison of Cfengine, Libvirt and MLN

- "√" means the tool have this feature.
- "×" means the tool don’t have this feature.

<table>
<thead>
<tr>
<th>Tool</th>
<th>Libvirt</th>
<th>MLN</th>
<th>Cfengine</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level</td>
<td>low</td>
<td>high</td>
<td>high</td>
</tr>
<tr>
<td>Efficiency</td>
<td>low</td>
<td>medium</td>
<td>high</td>
</tr>
<tr>
<td>Scalability</td>
<td>low</td>
<td>medium</td>
<td>high</td>
</tr>
<tr>
<td>Reliability</td>
<td>low</td>
<td>medium</td>
<td>high</td>
</tr>
<tr>
<td>Predictability</td>
<td>low</td>
<td>medium</td>
<td>high</td>
</tr>
<tr>
<td>Knowledge Management</td>
<td>×</td>
<td>×</td>
<td>√</td>
</tr>
<tr>
<td>Self-repairing</td>
<td>×</td>
<td>×</td>
<td>√</td>
</tr>
<tr>
<td>Human Intervention</td>
<td>√</td>
<td>√</td>
<td>×</td>
</tr>
<tr>
<td>Convergence</td>
<td>×</td>
<td>×</td>
<td>√</td>
</tr>
</tbody>
</table>
Evaluate Cfengine-Libvirt Model

- Validity: meet the requirements of a promise model
- Reproducibility: technical implementation
- Reliability: trust and reliability come from keeping promises
- Strategic value: the desired-state or promised state is our goal
- Business value: there is a large marketing potential
The issues of this project

- Unique Id of VMs
- The live migration with cfengine
- The reliability of the results
Conclusion

To answer the research question:

1. Convergence Principle was introduced: lead VMs closer to ideal state.

2. Promise Theory help us understand our systems better and reduce their complexity:
   - Promise Theory clearly result in a model of System Knowledge.
   - Virtualization is Knowledge Management.

3. We find a convergent model to present a new platform for building and maintaining a number of virtualized environments automatically.
Future Work

- Testing the convergent model in cfengine
- Set up experiments to compare cfengine with the other tools
- Knowledge Management in Cloud Computing
Thank you for listening!

"Progress is made by lazy men looking for easier ways to do things"

— Robert A. Heinlein

Questions?