Configuring Infrastructure for the Cloud
Automated planning & agents

Paul Anderson
<dcspaul@ed.ac.uk>

KES AMSTA 2011
Background

- Several different communities have an interest in configuring some aspect of computing infrastructures -
  - “System configuration”, GRID, Network configuration, Application configuration ...

- Although the approaches have been slightly different, there is a lot of commonality -
  - Specification languages & policy, deployment, federated specification, security, robustness ...

- The Cloud is different only in emphasis ..
  - Less predictable, more devolved control, more opaque
Configuration Evolution

- Manual configuration
  - doesn’t scale, error prone, ...

- Imperative scripts
  - scalable
  - but difficult to prove properties of resulting configuration

- Declarative specifications
  - guarantees properties of resulting configuration
  - but essentially “random” order of changes

- Stored change plans
  - declarative specifications & controlled change order
  - inflexible, unlikely to cover all requirements
Change Planning
An Example Reconfiguration

A (up)  B (down)

“current” state

A (down)  B (up)

“goal” state

constraint: C is always attached to a server which is “up”
Possible Plans

1. **A down, B up, C.server=B** ✗

2. **A down, C.server=B, B up** ✗

3. **B up, A down, C.server=B** ✗

4. **B up, C.server=B, A down** ✓

5. **C.server=B, A down, B up** ✗

6. **C.server=B, B up, A down** ✗
“Cloudburst”

- Perhaps we need to change the DNS for the server ...
- Maybe the server needs to access internal services ...
Automated Planning

- Fixed plans cannot cover every eventuality
- We need to prove that any manual plans
  - always reach the desired goal state
  - preserve the necessary constraints during the workflow
- The environment is a constant state of flux
  - how can we be sure that the stored plans remain correct when the environment has changed?
- Automated planning solves these problems
  - but introduces others ...
Herry’s Prototype

- Current state and goal state input to planner together with a database of possible actions
- Planner (LPG) creates workflow
- Plan implemented with “Controltier” & “Puppet”
Some Issues

- **Usability (most important!)**
  - administrators are relinquishing control
  - automatic systems can often find “creative” but inappropriate solutions if some constraint is missing

- **Plan repair**
  - reconfigurations often occur in response to failures or overload, so the environment is unreliable

- **Goals are often “soft”**
  - there may be more than one acceptable goal state - usually with different levels of desirability
  - eg. “low execution time” or “least change”

- **Centralised control has problems ....**
Decentralised Configuration
Decentralised Configuration

- Centralised configuration
  - allows a global view with complete knowledge

- But ...
  - it is not scalable
  - it is not robust against communication failures
  - federated environments have no obvious centre
  - different security policies may apply to different subsystems

- The challenge ...
  - devolve control to an appropriately low level
  - but allow high-level policies to determine the behaviour
Distributed configuration with centralised policy

Subsystem-specific mechanisms
“OpenKnowledge” & LCC

- Agents execute “interaction models”
- Written in a “lightweight coordination calculus” (LCC)
- This provides a very general mechanism for doing distributed configuration
- Policy is determined by the interaction models themselves which can be managed and distributed from a central point of control
- The choice of interaction model and the decision to participate in a particular “role” remains with the individual peer
  - and hence, the management authority
A Simple LCC Example

\textbf{a(buyer, B) ::}
\begin{align*}
\text{ask}(X) & \Rightarrow \text{a(shopkeeper, S)} \text{ then} \\
\text{price}(X, P) & \Leftarrow \text{a(shopkeeper, S)} \text{ then} \\
\text{buy}(X, P) & \Rightarrow \text{a(shopkeeper, S)} \\
& \Leftarrow \text{afford}(X, P) \text{ then} \\
\text{sold}(X, P) & \Leftarrow \text{a(shopkeeper, S)}
\end{align*}

\textbf{a(shopkeeper, S) ::}
\begin{align*}
\text{ask}(X) & \Leftarrow \text{a(buyer, B)} \text{ then} \\
\text{price}(X, P) & \Rightarrow \text{a(buyer, B)} \\
& \Leftarrow \text{in\_stock}(X, P) \text{ then} \\
\text{buy}(X, P) & \Leftarrow \text{a(buyer, B)} \text{ then} \\
\text{sold}(X, P) & \Rightarrow \text{a(buyer, B)}
\end{align*}
An Example: VM Allocation

- **Policy 1 - power saving**
  - pack VMs onto the minimum number of physical machines

- **Policy 2 - agility**
  - maintain an even loading across the physical machines

Discovery service

role: overloaded

role: underloaded

migrate

Friday, 1 July 2011
Distributed Planning for Configuration Changes
Behvioural Signatures

- Blue transitions are only enabled when the connected component is in the appropriate state
  - simple plans execute autonomously
- The plan executes in a distributed way
- The components are currently connected manually
  - and the behaviour needs to be proven correct in all cases
Planning with BSigs
(Herry’s current Phd work)

- If we have ...
  - a set of components whose behaviour is described by behavioural signatures
  - a “current” and a “goal” state

- We can use an automated planner to generate a network of components to execute a plan which will transition between the required states

- Some interesting possibilities
  - this can be structured hierarchically
  - the plans may not be fixed
    ie. they could handle some conditionals and errors
Configuring Infrastructure for the Cloud
Automated planning & agents

- Paul Anderson <dcspaul@ed.ac.uk>
- Herry <h.herry@sms.ed.ac.uk>
Sponsored by HP Research Innovation Grant