Automatic Undo for Cloud Management via AI Planning

Ingo Weber, Hiroshi Wada, Alan Fekete, Anna Liu and Len Bass

Based on presentation at USENIX HotDep ‘12
NICTA in Brief

- Australia’s National Centre of Excellence in Information and Communication Technology
- Five Research Labs:
  - ATP: Australian Technology Park, Sydney
  - NRL: UNSW, Sydney
  - CRL: ANU, Canberra
  - VRL: Uni. Melbourne
  - QRL: Uni. Queensland and QUT
- 700 staff including 270 PhD students
- Budget: ~$90M/yr from Fed/State Gov and industry
- ~600 research papers/year, ~150 patents total
Yuruware can help…

<table>
<thead>
<tr>
<th>Yuruware can help…</th>
<th>Bolt</th>
<th>Monitor</th>
<th>Compare</th>
<th>Clip</th>
</tr>
</thead>
<tbody>
<tr>
<td>Build business continuity</td>
<td></td>
<td></td>
<td>✔️</td>
<td></td>
</tr>
<tr>
<td>Compare providers</td>
<td></td>
<td>✔️</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Monitor multiple clouds</td>
<td></td>
<td>✔️</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Find idle resources</td>
<td></td>
<td></td>
<td>✔️</td>
<td></td>
</tr>
</tbody>
</table>
Motivation of this work

- **Yuruware Bolt: site-replication across regions**
  - Executes long-running operations - create, delete and update variety resources via AWS API

- **Issues we face**
  - High cost of writing unit tests
    - Preparing a test bed, reset after each test, and error recovery
    - “Why there is no DBUnit for cloud!”
  - High cost of error handling
    - Resources often get stuck in an unexpected state
    - Well-coordinated and tailored clean up for each case
Our Goal

- Provide an “undo” button to cloud users
  - Allow for rolling back to a previous state
  - e.g., undelete deleted resources and reconstruct the relations among resources

- We’re users - cannot alter API or implementation
  - Some operations are undoable, e.g., detach a VIP
  - Some operations are semi-undoable, e.g., stop a VM
  - Some operations are not undoable, e.g., delete a disk

- Less invasive
  - Minimum changes in existing code or scripts
Status quo

- Administrators and/or scripts talk to cloud via API
Goal

- Provide the ability to go back to a checkpoint
- No change in scripts except checkpoint and commit
Our approach

• “undo one by one in reverse order“ does not always work
  – No undo action is available
    • e.g., no “undeleteing a deleted resource“
  – Undo requires a different sequence of API calls
    • e.g., “deleteing an autoscaling group“ does not work
  – Simple undo could results in a different state
    • e.g., Undo “stopping an instance with a VIP“
  – Undo operation may fail
    • e.g., detaching a volume could fail. Need an alternative.
  – Not optimal
    • e.g., Bolt‘s operations (creating many temporary resources)

• Tedious to hand-code for all possible cases!
Pseudo-delete for not-undoable ops

- Execute API calls if they are undoable
- Defer the execution of non-undoable calls until commit
AI planning for the rest of cases

- Changes made by (semi-)undoable API calls are compensated by an AI planner
- AI planner finds ways to handle errors potentially occur during undo as well
AI Planning 101

• Given the initial state of the world, goal state, and a set of available actions, find a sequence of actions that leads from the initial to the goal


• Highly optimized heuristics tame the problem for practical purposes
Planning under uncertainty

• In our problem
  – Initial state: state at rollback
  – Goal state: state at checkpoint
  – Actions: AWS APIs

• We use FF [*] with an extension to handle actions with alternative outcomes

• Finds “maximal“ contingency plans
  – e.g., if detaching a volume fails, stop the attached instance if possible. If a planner cannot solve, ask human intervention

Domain model: example

- Action to delete a disk volume in PDDL

```pddl
(:action Delete-Volume
 :parameters (?vol - tVolume)
 :precondition
 (and
  (volumeAvailable ?vol)
  (not (unrecoverableFailure ?vol)))
 :effect
 (oneof
  (and
   (deleted ?vol)
   (not (volumeAvailable ?vol)))
   (unrecoverableFailure ?vol)))
```
## Domain model: actions

<table>
<thead>
<tr>
<th>Resource type</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtual machine</td>
<td>launch, terminate, start, stop, change VM size</td>
</tr>
<tr>
<td>Disk volume</td>
<td>create, delete, create-from-snapshot, attach, detach</td>
</tr>
<tr>
<td>Disk snapshot</td>
<td>create, delete</td>
</tr>
<tr>
<td>Elastic IP address</td>
<td>allocate, release, associate, disassociate</td>
</tr>
<tr>
<td>Security group</td>
<td>create, delete</td>
</tr>
<tr>
<td>Auto-scaling group</td>
<td>create, delete, change-sizes, change-launch-config</td>
</tr>
<tr>
<td>Auto-scaling launch config</td>
<td>create, delete</td>
</tr>
<tr>
<td>Tag</td>
<td>create, delete</td>
</tr>
<tr>
<td>Others</td>
<td>AZ, cluster online/offline, ...</td>
</tr>
</tbody>
</table>
Evaluation

• Scalability of the planner based on an internally released prototype
  – AWS cmd line tool replacement
• Use cases: ~70 different planning settings tested
• Evaluation 1: against plan length
• Evaluation 2: against # of unrelated resources
Evaluation 1: vs plan length

20 length is the maximum we need in our problem. Executing a plan with 10 steps takes ~145 sec.
**Evaluation 2: vs # of unrelated resources**

Basis: most difficult problem from previous slide
Planner’s cost is small unless having 1000s of resources

- + 500 unrelated instances and volumes
- ~50 related resources

Seconds

Facts + Actions (in 1000s)
Conclusion & future work

- Rollback in cloud using AI planner
  - Formalized part of AWS APIs in a planning domain
  - Planning execution time is marginal for practical system sizes

- Future work
  - Extending checkpoints to capture internal resource state
  - Parallelizing plans
  - Finding forward plans with constraints
Questions?