Supporting Undoability in System Operations

System operation could go wrong

- Operation errors as the largest contributor to system failures [1]

- Configuration errors (28%) and human errors (13%) cause service disruptions [2]

- System operation plays a large role in overall system reliability

Cloud increases the problem

- More and more services operate on API-controlled infrastructures
  - e.g., IaaS cloud
  - Enabled frequent changes - increase chance of error

- Activities are performed without a safety net
  - Deleting a virtual disk results in a total loss
  - Stopping a VM changes relations to other resources.

Not obvious how to fix it.
Project Goal and Scope

• Provide an "undo button" on Amazon Web Service
  – Facility to rollback changes is valuable support for dependability in various areas
  – Introduce "atomicity" in system operation

• Allow for restoring the state of cloud resources such as VM, disk volume, IP address and the structure
Challenges

- Not possible to "copy back snapshot" directly
  - Can only manipulate through given API
  - Need to find a right sequence of operations
- "Revert one-by-one" may not work
  - Constraints among operations/resources
  - Error-prone operations require "backup" plans
- Completely non reversible operations
  - e.g., deleting resources
- Partly reversible operations
  - Not all resource properties can be restorable, e.g., id, creation timestamp, ...
Our Approach: Checkpoint

Introduce "checkpoint" operation to record the state of resources

No change in given API operations

No change in "provider"
Our Approach: Rollback by AI Planning

"Rollback" command triggers system restoration

Leverage AI planning to find a sequence of operations to restore the state at the checkpoint
Why AI Planning?

• Traditional techniques to rollback long-running transactions do not apply or are sub-optimal:
  – There may be no direct reverse operation
  – More than one operations required to reverse one
  – Hand-coding for all possible cases is tedious

• AI Planning:
  – Given start state, goal state, set of actions, searches a solution in the state of possible plans
  – Our variant finds maximal contingency plans

  • If one action fails, but the goal is still reachable, a backup plan will be executed
# Domain Model: Supported Resources

<table>
<thead>
<tr>
<th>Resource Type in AWS</th>
<th>API operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtual machine</td>
<td>launch, terminate, start, stop, change 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>Autoscaling group</td>
<td>create, delete, change size, change config, create config, delete config</td>
</tr>
<tr>
<td>Tag</td>
<td>create, delete</td>
</tr>
</tbody>
</table>
Our Approach: Wrap non reversible ops

Replace completely non reversible operations with "pseudo-do" operation

"Commit" command applies changes

Partly reversible operations

• Many operations seem to be undoable - but not

• e.g., starting a VM is NOT reverse of stopping it
  – Properties cannot be restorable: resource id, dynamic IP address, or creation timestamp

• Restoring ALL properties is not feasible for many scenarios; however, it may not be required
  – e.g., undoing "change the size of the web layer" can safely ignore changes to the creation timestamps
Our Approach: Undoability Checking

• Facility to examine whether a collection of operations to execute are undoable

List properties need to be restorable

Tool provider

Tool user (e.g., sys admin)

Define

Operation(s) to execute (e.g., script, command)

Feedback

• Undoability (yes/no)
• List of causes if not undoable

Define

Resources and properties required restorable on undo

Apply Projection

Generate

Projection Specification

Per operation: Generate pre and post-states

Check undoability per pre-post state pair

Full domain model (e.g., AWS)

Generate

Projected domain model

"subset" of the domain

algorithm and proof at undo.research.nicta.com.au

Check if operations to execute are undoable
Two Usage Models of Checker

• Offline: check the undoability of all operations to execute under any situations
  – Much stronger than needed since it checks situations that may not occur
    • e.g., delete a volume if resource X is in state Z

• Online: check the undoability of an operation before execution under the current situation
  – Check the undoability exactly as needed
  – Performance penalty due to state sensing
Evaluation

• Evaluate prototype
  – Undo engine released at undo.research.nicta.com.au

• Execution time to search undo plans
  – Use FF [3] with an extension for planning

• Execution time to check undoability of a domain (offline)

• Validation with real-world scenarios

Evaluation: planning performance

- 20 length is the maximum we needed for practical use cases
- Executing a plan with 10 steps takes ~145 sec

on Intel i7 (3.4GHz) with 4GB RAM

Evaluation: undoability check

• Checking a full domain (35 actions)
  – Projection: remove unrecoverable errors and few properties, and adding pseudo-delete and undelete
  – 11.0 seconds to check 1330 planning problems
  – Result can be reusable unless the projection changes

• Examine our day-to-day manual tasks
  – e.g., adding a slave to a database server, expand the disk size, scaling up/down web layer, and upgrading app layer
  – Confirmed the undoability given a domain projection
Conclusion

- Facility to offer "undo" and check undoability
  - Provide rollback on a set of API operations
  - Check if operations are undoable under in certain context (projected domain)
- Prototype shows the execution time is marginal and the algorithm scales well

- Future work
  - Capture internal resource state
  - Parallelizing plans to speed up execution
Appendix
Domain Model: Example

- Formally captures operations in domain
  - e.g., a set of operations that AWS supports
- Action to delete a disk volume in PDDL

```
(:action Delete-Volume
 :parameters (?vol - tVolume)

:precondition
(and
  (volumeAvailable ?vol)
  (not (unrecoverableFailure ?vol)))

:effect
(oneof
  (and
    (deleted ?vol)
    (not (volumeAvailable ?vol)))
  (unrecoverableFailure ?vol)))
```

Volume to be deleted must be available
Volume will become deleted and unavailable, or failure
Domain Projection and Undoability Check

• Domain projection
  – Remove properties not required to be restorable
  – Add pseudo-delete for each delete operations

• Undoability checking algorithm

  foreach operation op to execute:
  foreach pre-state:
    find possible post-states by applying op:
    foreach post-state:
      let AI planner finds a plan from pre to post-state
      if no plan found
        op is not undoable

  – pre-states is an infinite set. We obtain sufficient pre-states from domain model