ABSTRACT
A large number of cloud application failures happen during sporadic operations such as upgrade, deployment reconfiguration, migration and scaling-out/in. Most of them are caused by operator and process errors [1]. From a cloud consumer’s perspective, recovery from these failures relies on the limited control and visibility provided by the cloud providers. In addition, a large-scale system often has multiple operation processes happening simultaneously, which exacerbates the problem during error diagnosis and recovery. Existing built-in or infrastructure-based recovery mechanisms often assume random component failures and use checkpoint-based rollback, compensation actions [2], redundancy and rejuvenation to handle recovery [3]. These recovery mechanisms do not consider the characteristics of a specific operation process that consists of a set of steps carried out by scripts and humans interacting with fragile cloud infrastructure APIs and uncertain resources [4]. Other approaches such as FATE/DESTINI [5] look at the process implied by a system’s internal protocols and rely on built-in recovery protocol to detect bugs. The problem we target is at a different level related to the external sporadic activities operating on a hosted cloud application.

Our overall approach is to explicitly model and analyze an operation as a process. We divide the process into sections consisting of steps. The division criteria can be different for different purposes, such as error diagnosis, conformance checking or recovery. For recovery, our initial division criteria include: 1) atomicity to achieve all-or-nothing for a group of actions making recovery easier; 2) idempotence to enable the same or parameterized actions to be re-executed for recovery; 3) fine-granularity to allow higher-level of reuse of existing steps during recovery; 4) alternatives-friendly to allow alternative actions to be executed to reach the same expected result during recovery. We specify a set of assertions representing the expected outcomes produced by the execution of each section. We use our run-time assertion evaluation and monitoring system [7] to help detect errors at the end of each section and trigger recovery actions if necessary.

We applied our approach to a typical AMI-driven rolling upgrade process for AWS-hosted applications. Netflix has a well-known tool Asgard [6] supporting this process. We divided Asgard’s internal recovery and error-handling mechanisms using our division criteria. We also enabled log-triggered assertion evaluation at the end of some sections. We were able to detect errors earlier and catch errors that would be missed by Asgard. However, we found that Asgard was not designed with the granularity and idempotence required for re-execution. So we could not conduct re-execution based recovery using selected pieces of Asgard’s code. We did develop external alternative actions for recovering from some subtle errors. For example, upon detection of some unexpectedly stopped instances, our alternative actions either restarted these instances or killed them and relied on the auto-scaling group to restart them. Our recovery significantly shortens the time that Asgard would have taken otherwise.

At the moment, we are developing patterns and frameworks that can allow operators to easily design and instrument their scripts for marking atomic actions, re-execution blocks and alternative actions for recovery purposes. We are also integrating this with our assertion evaluation and monitoring framework [7].

Keywords
process-oriented; idempotence; assertion evaluation;

References
1. Cloud Failures Recovery from a Consumer’s Perspective
Cloud recovery can be provided either by cloud providers or by cloud consumers. Our research is focusing on the recovery from a consumer’s perspective.

1.1. Challenges for Recovery
- Limited visibility control
- Fragile cloud infrastructure APIs and uncertain resources
- Simultaneous multi-processes

1.2. Existing Recovery Mechanisms
- Checkpoint-based Rollback, Compensation, Redundancy and Rejuvenation
- Ignoring the recovery (from a consumer’s perspective) for operation processes on cloud applications
- FATE/DESTINI approach [1]
  - Looking at the process implied by a system’s internal protocols
  - Relying on built-in recovery to detect bugs

2. Our Approach for Recovery
- Modelling and analysing an operation as a process
- Process division based on the division criteria
- Assertion evaluation for certain sections of process
- Errors detection using our assertion evaluation and monitoring system [2]
- Recovery alternative actions triggering if necessary

See Figure 1.

2.1. Process Division Criteria
1) Atomicity to achieve all-or-nothing for a group of actions making recovery easier;
2) Idempotence to enable the same or parameterized actions to be re-executed for recovery;
3) Fine-granularity to allow higher-level of reuse of existing steps during recovery;
4) Alternatives-friendly to allow alternative actions to be executed to reach the same expected result during recovery.

2.2. Process Division
- AMI-driven rolling upgrade
- Asgard [3] utilized to provide this upgrade process
- Based on our division criteria
See Figure 2.

2.3. Assertion Evaluation
- Log-triggered assertion evaluation at the end of some sections
See Figure 2.

2.4. Initial Recovery Alternative Actions
- Rewind & Replay
- Compensation
See Figure 2.

2.5. Initial Evaluation for Our Approach
- Recovery Alternative Actions Handler
- Handling some subtle errors such as unexpectedly stopped instances
- Significantly shortening the time compared to Asgard [3]

See Figure 3 shows our evaluation environment.

Figure 1: Overview of Our Approach
Figure 2: Process Division & Monitoring
Figure 3: Evaluation Environment

3. Our current and Future work
- Developing patterns and frameworks for marking atomic actions, re-execution blocks and recovery alternative actions
- Integration with our assertion evaluation and monitoring framework [2]

4. References