A Recoverability-Oriented Analysis for Operations on Cloud Applications

Min Fu, Liming Zhu, Len Bass, Xiwei Xu
Software Systems Research Group, NICTA, Sydney, Australia
School of Computer Science and Engineering, University of New South Wales, Sydney, Australia

Abstract—Consumer-initiated sporadic operations on cloud applications, such as deployment, upgrade and reconfiguration, may fail because of the inherent uncertainty of operating in a cloud environment. For example, if a VM is not able to start, operations dependent on that VM will fail. In this paper we propose an approach for analyzing sporadic operations on cloud applications to facilitate recovery. We do this by first creating a process model of the sporadic cloud operation. The process may not be suitable for recovery operations or the application’s deployment architecture since an individual step may be at the wrong level of granularity or a failure may unnecessarily affect multiple steps. We then propose a set of process division criteria for revising an operation process to support recovery. This set of criteria contains four aspects: 1) Atomicity to support the imposition of all-or-nothing transactions on portions of the process; 2) Idempotence to allow for re-execution of a failed portion; 3) Granularity to allow reuse of existing steps; 4) Recovery Actions Identifiable to allow proper recovery actions to be taken to recover from the errors occurring during the operation. We demonstrate the feasibility of imposing these criteria by using the rolling upgrade operation in Asgard - a popular cloud management tool for AWS EC2.

Keywords—cloud application; consumer-initiated; process model; recovery operations; process division criteria

I. INTRODUCTION

Past experience shows that the majority of cloud application failures are due to the errors in the operations on cloud applications[1][2]. However, reducing errors during operations is not an easy task and not always feasible due to the inherent uncertainty of operating in a cloud environment[5][7][8][9]. As such, one solution can be enabling recoverability for the errors during the operations. For example, when creating a virtual machine fails during a cloud operation, we can take the recovery action of recreating this virtual machine to recover from the error. To facilitate the recoverability of an operation, analyzing the operation in a recoverability-oriented way is imperative. Our research has two main contributions: 1) we propose new process division criteria to facilitate recovery; 2) we illustrate how to make use of the process division criteria.

Second, we define a set of recovery-oriented process division criteria to divide the operation process. In order to analyze the recoverability of the operation process, the process should be analyzed based on the recovery-oriented process division criteria while considering the deployment architecture of the application. The division criteria contains four aspects: 1) Atomicity to support the imposition of all-or-nothing transactions on portions of the process; 2) Idempotence to enable the same or parameterized actions to be re-executed; 3) Granularity to allow high level reuse of existing steps; 4) Recovery Actions Identifiable to allow proper recovery actions to be executed.

Finally, we divide the operation process based on the above-defined process division criteria. The details for how to apply the criteria to dividing an operation process are discussed in section IV.

Our research has two main contributions: 1) we propose new process division criteria to facilitate recovery; 2) we illustrate how to make use of the process division criteria.

II. RELATED WORK

One aspect of our process division criteria takes similarity to the atomicity in transaction. A transaction is a group of operations that have the following properties: atomic, consistent, isolated, and durable (ACID) [12]. For an atomic transaction, either this transaction must succeed, or the effects of this must rolling back [12]. We have looked into two types of transactions: database transactions (DB transactions) and web service transactions (WS transactions). Atomicity for a DB transaction means that this transaction provides an "all-or-nothing" proposition, stating that all the work units performed in a database must either complete in their entirety or have no effect whatsoever[6]. WS transaction is an OASIS standard[13]. Atomicity for WS transaction is to achieve all-or-nothing property for a group of services, and it defines three protocols (completion, volatile two-phase commit, and durable two-phase commit) as well as a set of services[13]. These protocols and services together ensure automatic activation, registration, propagation and atomic termination of Web Services.

For the atomicity aspect of our process criteria, we were also inspired by atomic operations in concurrent programming[16] defined as program operations that run completely independently of any other processes[14]. For example, in OpsCode[18] scripts language such as Chef[18], exclusive file locks mechanism[18][19] is used to achieve operation atomicity when two chef clients are operating on the same file. Likewise, the aspect of atomicity in our process

...
division criteria should ensure that an atomic section will not be interrupted by other sections, especially when there are concurrent operations.

Sometimes atomic groups may need to rollback and this rollback is realized by undoing certain operation steps. Before undoing certain operation steps, their undoability should be checked first. We rely on an undoability checking mechanism[17] to check to what extent each operation is undoable.

### III. OPERATION ANALYSIS APPROACH

Fig. 1 shows the overview of our approach for operation analysis. First, we model and analyze the operation as a process to create the process model. Second, we define a set of recovery-oriented process division criteria to divide the process. Finally, we divide the process based on the process division criteria. The details of how to divide the process are exemplified in section IV.

![Fig. 1. Overview of Process Analysis Approach](image)

#### A. Recovery-Oriented Process Division Criteria

1. Ensuring **Atomicity**: Similar to the atomicity in WS transactions[13] and DB transactions[6], atomicity in our division criteria means to achieve all-or-nothing for a group of actions. For instance, if removing an old AMI from an ASG (auto-scaling group) or attaching a new AMI to this ASG fails, the ASG will fail to be updated, so they must both complete successfully and should be put into an atomic group. Atomicity can help maintain the process in a consistent state when rolling back certain steps for the recovery. Dividing by Atomicity makes the process into several atomic action-groups, and this is the first step in our process division.

2. Ensuring **Idempotence**: In our process division criteria, Idempotence means to enable the same or parameterized actions to be re-executed without changing the result. After the operation process is divided into several action-groups, some of them should be combined as a section in order to make this section idempotent. If a section is idempotent, it means that this section, no matter parameterized or non-parameterized, and no matter being executed how many times, will always return the same execution result. For example, when using Chef Script to remove an old AMI from the existing LC (launch configuration) and attach a new AMI to the existing LC, the result will always be the same (new AMI attached to the LC), regardless of execution times. Idempotent sections, in some cases, can be recovered by re-executing them. Since Idempotence can ensure that the execution result won’t change, Idempotence should be included in the criteria.

3. Adjusting **Granularity**: Granularity in our criteria means to allow high level reuse of existing steps in the operation process. By including this aspect in our criteria, recovery actions can be taken with high reusability (re-executing the same recovery functions or steps). Moreover, after making certain sections reusable, they can also be reused by other operations when designing steps for those other operations in OpsCode Scripts. As such, our process division criteria should include Granularity as one aspect.

4. Ensuring **Recovery Actions Identifiable**: Recovery Actions Identifiable means to allow recovery actions to be taken for recovery. When dividing the operation process into a recoverability-oriented way, the key point is to make sure each section can be recoverable by certain recovery action(s). Our recovery actions mainly have two categories: Undo/Redo[20] and Compensation[3]. For example, after the step of updating existing ASG, if the ASG is wrongly reattached to old LC by another team, we can recover it by recalling the API function of “UpdateAutoScalingGroup”[4]. Since our goal is to facilitate recoverability, the aspect of Recovery Actions Identifiable should be necessarily included into the process division criteria.

#### B. Detailed Operation Process Division

First, Atomicity is used to break the process into atomic action-groups, followed by Idempotence to combine certain action-groups into idempotent sections. Next, we further combine some of the action-groups to make highly reusable sections with Granularity. Finally, we have to make sure that for each section certain recovery actions can be identified to recover from the errors inside. The details are provided in the approach demonstration in section IV.

### IV. DEMONSTRATION FOR ANALYSIS APPROACH

We use the rolling upgrade operation in Asgard[11] for the demonstration, and Fig. 2 shows its original process. We divide it into sections by using the process division criteria.

![Fig. 2. Original Rolling Upgrade Operation Process in Asgard](image)

This original operation process is mined from corresponding Asgard source code. Due to the paper length limitation, we don’t show the details of how to mine this procedure by using Asgard source code and relevant logs generated.
Firstly, we divide the process by Atomicity. Some steps, such as step 1 and 2 (serve as creating new LC), should be combined as an atomic group. Steps 6, 7, 8 serve as removing instance from ELB so they should also be combined into an atomic section. The division result is illustrated in Fig. 3.

![Fig. 3. Process Division after Ensuring Atomicity](image)

Then, based on the result in Fig. 3, we further divide the process by Idempotence. Specifically, we check if the action-groups are idempotent. If some action-groups are not idempotent, we try to combine some of them into several idempotent sections. The result is shown in Fig. 4.

![Fig. 4. Process Division after Ensuring Idempotence](image)

Next, the process is further revised by Granularity. For instance, step 1, 2 and 3 can be merged since they serve as updating the ASG (reusable). The result is shown in Fig. 5.

![Fig. 5. Process Division after Adjusting Granularity](image)

Finally, the process is further revised by Recovery Actions Identifiable. We notice that, if the recovery action for step 9 is to re-terminate the old instance, the ASG will auto-launch a new instance (step 10), so this recovery action actually involves step 9 and step 10. As such, step 9 and 10 are combined as a section. The final division output is shown in Fig. 6.

![Fig. 6. Process Division after Ensuring Recovery Actions Identifiable](image)
Now the rolling upgrade process is revised into six sections (as shown in Fig. 6). For the possible failures in the first section, the recovery action can be recreating the new LC using the new AMI or reattaching the new LC to the ASG, for example. For the second section, if instance killing order is incorrect, we can reset the instance killing order as the recovery action. For the third section, if the instance fails to be removed from ELB, the cloud API function of removing instance from ELB can be recalled (e.g. DeregisterInstancesFromLoadBalancer in Java). For the fourth section, if the error is that the old instance cannot be terminated and new instance cannot be created, the recovery action can be replaying this section. For the fifth section, if the instance cannot become ready, the recovery action is to restart the instance by calling “StartInstances (StartRequest)” function in java. For the sixth section, one possible failure is that the new instance is not added into ELB, and the feasible recovery action can be re-adding the instance into the load balancer by calling the SDK function of “RegisterInstancesWithLoadBalancer(InstanceList, LoadBalancerName)” in Java.

V. DISCUSSION

Although we only demonstrate our approach by using rolling upgrade operation in Asgard, the approach is also supposed to be applied on the other operations on cloud applications to facilitate recoverability for example, deployment process on cloud applications[21] can be revised based on our criteria and recovery can be enabled after doing analysis for each section of the deployment process.

The recovery-oriented process division criteria is not only applicable for revising and analyzing the operations on cloud applications to facilitate recovery, but also the criteria might possibly provide guidance on how to design operations and deployment architectures on cloud applications. For example, when designing and writing the Chef scripts for a cloud upgrade operation, if developers make the function modules idempotent or highly reusable and make the architecture components less dependent on each other, the scripts might be better for recoverability. And by doing so it should also facilitate the exceptions detection and handling in the test-driven structured scripts[10].

VI. CONCLUSION & FUTURE WORK

The errors that happen during operations on cloud applications have become the main reason for cloud applications to fail. In order to recover from those errors, the operations should be analyzed in a recovery-oriented way to facilitate the recovery. Hence, we propose an approach for how to analyze an operation on cloud applications to facilitate recovery. Our approach first build a process model for the operation and then divide the process based on a set of recovery-oriented process division criteria. We demonstrate our approach by imposing the process division criteria on the rolling upgrade operation in Asgard[11] which is a popular cloud management tool for AWS EC2[4].

Our future work includes more systematic assessment of an application’s deployment architecture for its impact on sporadic operations and automated derivation of a more recoverable operation.

ACKNOWLEDGMENT

NICTA is funded by the Australian Government through the Department of Communications and the Australian Research Council through the ICT Centre of Excellence Program.

REFERENCES

[18] OpsCode official Website: http://www.opscode.com/chef/ (last access time: 20:06pm 27th Sep 2013).