Accurate and Efficient Rollback Recovery in SDN

Ying Zhang, Ravi Manghirmalani
Ericsson Research

I. Introduction

Checkpointing is a common and powerful approach to recover from transient errors in servers and distributed systems [1]. In general, a system periodically records its state during normal operation and stores it in non-volatile storage *i.e.,* checkpointing. Upon failure, it is restored to a previous state, and the system restarts the execution from this intermediate state, *i.e.,* rollback process, thereby reducing the lost computation. This technique is especially useful for long-running applications such as scientific computing and telecom applications, where restarting from the beginning can be costly. It is also commonly used for debugging and root cause analysis [2].

Despite its usefulness, checkpointing and rollback are rarely used in networking infrastructure for the following reasons. First, the network interacts with the outside world constantly, *i.e.,* receiving packets and sending them out, the outside world cannot be rolled back. Second, traditional checkpoint-recovery mechanism assumes a fail-stop system, *i.e.,* upon any fault or failure, the process or system terminates. This assumption does not always hold true in networking. For example, a loop may exist for a non-negligible amount of time before it is detected, even if it's hurting the performance. Third, the network equipment and applications are mostly blackboxes to the operators, making it impossible to instrument and reason. Fourth, the network state can be very huge, making the storage of checkpoints costly. Thus, in traditional network, the fault recovery process is still error-prone and notoriously hard. In contrast, we argue that the recently proposed Software-Defined Networks (SDNs) architecture [3] may make checkpointing and rollback feasible, thanks to its three key properties: simple abstraction, network wide visibility and direct control. Exploiting these properties of SDN, in this work, we explore a checkpointing and rollback framework for fault recovery in networks.

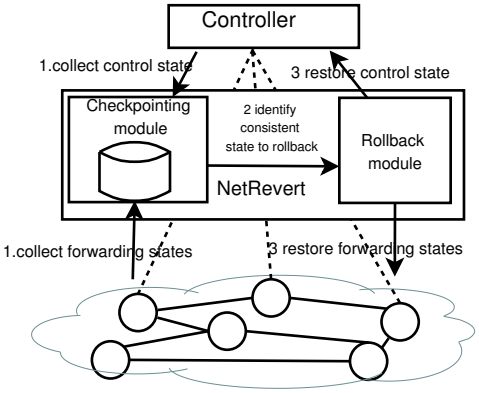

Fig. 1. Architecture

To recover from fault in a network, three principal steps are involved: detecting a fault, containing a fault, and rolling the necessary nodes of the network back to the last checkpoint in order to recover from the fault. In this work, our focus is to answer questions for the third step, *i.e.,* how to do checkpointing in SDN and how to perform rollback so that the system can function correctly. More precisely, we develop an efficient framework that allows the entire network state to roll back to a consistent and correct global state. First, we introduce and define a model for checkpointing and rollback in SDN and demonstrate its value. For checkpointing, we decide to keep not only the controller states but also the relevant states in switches to tolerate different types of failures. Second, using the model, we develop an algorithm to efficiently identify the consistent state with minimum amount of rollback, which effectively prevents losing too many states before the failure and prevents causing inconsistencies between controller and switches. Finally, unlike traditional rollback on hosts, certain external events that have happened cannot be rolled back, *e.g.,* link failure, which have to be fed into the rollback process. To accommodate this issue, we further propose a mechanism to selectively replay *hard* events during the rollback process.

II. Approach Overview

We propose a framework called NetRevert, which sits between the controller and the collection of OF switches as a software hypervisor, as is shown in Figure 1. This design allows it to passively log the OF messages exchanged between the controller and the network. It is composed of two modules, naturally, the checkpointing module and the rollback module. On the northbound, it takes full snapshots of the controller periodically and logs the messages between two consecutive checkpoints. On the southbound, it maintains the states of OF switches' flow tables, by reconstructing it from the OF messages. Periodically, it dumps the states to the non-volatile storage. This step enables recovery even under the failure of NetRevert itself. Upon failure, the rollback module runs an algorithm to determine the best state to be restored and perform the restoration. The checkpointing module also contains a garbage collector that automatically removes the checkpoints that will no longer be used in any rollback, in order to save the storage and search space. We briefly describe each module below.

A. Checkpointing module

Logically, the network is composed of one controller $CTRL$ and a set of switches $S_1, S_2 \ldots S_n$, which are defined as the nodes of the network. The set of messages exchanged between $CTRL$ and S_i is M_i, *e.g.,* the Openflow messages. Each

node takes a set of snapshots of its state periodically. For example, switch S_1 has a list of checkpoints $C_1^{S_1}, C_2^{S_1} \ldots C_t^{S_1}$. The messages are considered as the internal events of the entire network system, but external events to a particular node.

Checkpointing the controller is similar to check point any type of user level process. That is, a checkpoint of the controller includes the controller process's address space and the state of its registers. For recovery, the new process is spawned which initializes its address space from the checkpoint file and resets its registers. During the checkpointing, the operation is paused and the memory pages are copied into non-volatile storage.

Checkpointing the switch means taking snapshots of the flow tables in each switch. Since the rules are installed by the controller, the straightforward way is to let the controller keep track of the set of rules installed on each switch and then consider it as a part of the controller state. However, somewhat surprisingly, we found that none of the existing controllers maintains a copy of the flow tables for the switches, mainly because of the concerns of space and overhead [4]. Another possibility is to copy the memory pages (more precisely, the flow tables) from the switch, but it is even more expensive, as the control channel may be congested by the large amount of state transferred. Therefore, we take the approach to reconstruct the table entries on the NetRevert by intercepting the OF messages. To tolerate failures of NetRevert itself, we also store these states to disk periodically. Note that we choose to *NOT* record all the packets at the data plane, due to scalability concerns.

Incremental checkpointing is used to reduce the size of the logs. For example, if the memory for storing topology map hasn't been changed between the two consecutive full snapshots, then this part of the memory may not be saved duplicately. We use a simple heuristic to determine when to take a full snapshot: if the number of messages entering this node exceeds n, then we take a full checkpoint, which is a configurable parameter.

B. Rollback module

To recover from transient failures, the recovery mechanism restores system state to the last *complete* checkpoint (C_t^i) and then modifies it according to the incremental checkpoint data and the events (M_i). However, if the fault is caused by misconfiguration or software bugs, and if the root cause of the fault did not manifest itself until after the last checkpoint, the restored system is likely to fail again, *i.e.,* the error is latent through several checkpoints. Besides this issue, the message and relationship between nodes in the network complicate the rollback process because messages induce dependencies between states on different nodes. Upon a failure of one or more nodes in a system, these dependencies may force some nodes which did *NOT* fail to roll back. Moreover, the traditional checkpoint-recovery assumes that if there is a fault, the process or system terminates, which does not always hold true in the networking context. For example, it can be that the network violates a policy, which is notoriously hard to detect.

Identifying the maximum recoverable consistent state: Upon failure, we need to determine the maximum recoverable, network-wide consistent states among all the checkpoints in the history in an automated way. To achieve this, we model all the checkpoints as a graph called Checkpoint Graph or *CPG*, where each node represents a checkpoint and a directed edge shows the dependencies between the checkpoints. The edges can be either between checkpoints from the same switches/controllers or between checkpoints from different devices.

Selectively replaying events: After choosing the MRS and rolling back to it, incremental logging (M) can be replayed to bring the states forward to the state before failure. If the failure is due to misconfiguration or other software bugs, then the system will constantly fail if following these exact steps. We assume that the operator uses other means to identify the cause of the fault, and can provide them to our system. With this input, NetRevert will selectively replay the incremental loggings to avoid the fault again.

III. Preliminary Evaluation

For checkpointing, we use the libckpt [1], an open source in Unix environment with NOX controller. We emulate the network using Mininet [5] with the modified controller and 12 switches connected as the Abilene topology. NetRevert builds on top of FlowVisor to intercept OF messages. We first evaluate the impact of failure on the flows in the network. For transient failures on both controller and switches, the prolonged end-to-end delay caused by failure has been reduced from 1000ms to 100ms with the rollback recovery. Next, we directly evaluate the overhead of checkpointing by comparing the delay for handling the same amount of messages with different checkpointing configurations. We find that the time overhead of logging is small (at most 8%). The space overhead of logging is not significant, *i.e.,* maximum of 2GB per day during the experiment, thus we can tolerate to save the logs over a long period of time at low cost.

References

[1] J. S. Plank, M. Beck, G. Kingsley, and K. Li, "**Libckpt**: Transparent checkpointing under Unix," in *Usenix Winter Technical Conference*, January 1995.

[2] G. W. Dunlap, S. T. King, S. Cinar, M. A. Basrai, and P. M. Chen, "Revirt: enabling intrusion analysis through virtual-machine logging and replay," in *Proceedings of the 5th symposium on Operating systems design and implementation*, OSDI '02, 2002.

[3] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, pp. 69–74, March 2008.

[4] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "Nox: towards an operating system for networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, July 2008.

[5] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, 2010.