Automatic Software Upgrades for Distributed Systems

Sameer Ajmani and Barbara Liskov, MIT CSAIL
Liuba Shrira, Brandeis University
ajmani@csail.mit.edu • pmg.csail.mit.edu/upgrades

Goals

To support automatic software upgrades for *long-lived, large-scale* distributed systems, e.g., peer-to-peer networks, content distribution networks, server clusters, and sensor networks; and to enable those systems to provide service during upgrades.

Challenges

- Not all nodes can upgrade at the same time, so each upgrade must define a *schedule* for when nodes should upgrade.
- Nodes running different versions may need to interact, so we enable nodes to simulate multiple versions at once. However, perfect simulation is not always possible or practical.

Approach

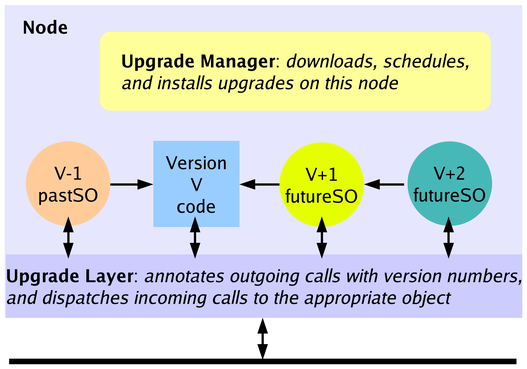
- The systems of interest are *robust* to node failures, so we model a node upgrade as a soft restart.
- Upgrades are rare, so we optimize for same-version interaction.
- Nodes recover from persistent state, so we execute state transforms on persistent state.

Model

- A system is a set of *objects* that interact via remote method calls
- Each object resides on a node and has an identity and state
- A single trusted party defines the software for an entire system
- Objects running the same version interoperate correctly
- A *system upgrade* defines a set of *class upgrades*, each of which defines how objects of a given old class upgrade to a new class

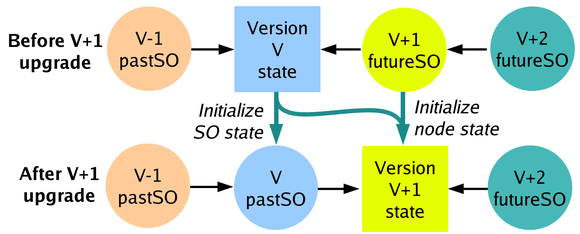
Research Questions

- How do upgrades affect a system's availability, fault-tolerance, performance, and security?
- How do different upgrade schedules affect these properties?
- How well can we simulate one version's behavior on another?
- How well can we transform one version's state to the next?
- How do we reason about upgrade correctness?



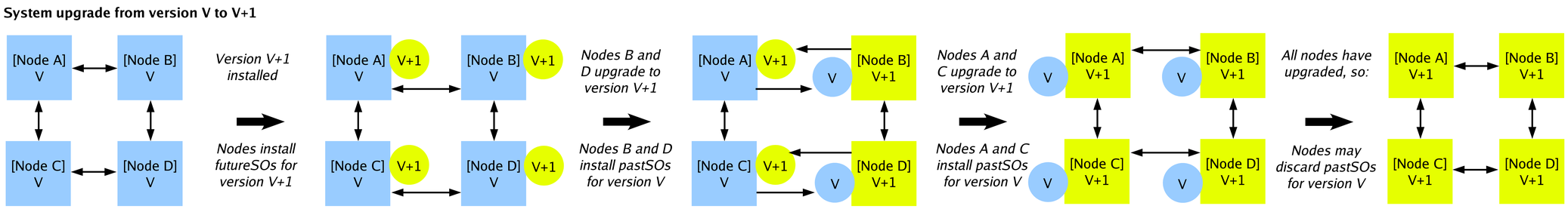
Upgrade Infrastructure

- Logically centralized *upgrade server*
  - Publishes upgrades for download
- Logically centralized *upgrade database*
  - Stores information on upgrade progress
- Per-node *upgrade manager*
  - Downloads, schedules, and installs upgrades
- Per-node *upgrade layer*
  - Handles cross-version calls via simulation objects (SOs). PastSOs handle older versions; futureSOs handle newer versions.



Node Upgrade

- Install futureSO to support the new version
- Wait until schedule allows node to upgrade
- Halt node software (abort operations in progress)
- Run state transform function (*pictured left*)
- Install pastSO to support the old version
- Start the new node software



System Upgrade

- New version is installed at the upgrade server
- Nodes discover the new version via polling or gossip
- Individual nodes upgrade according to the schedule
- Nodes eventually garbage-collect old pastSOs

Status

- Correctness criteria defined
- Upgrade server+database implemented as an SFS server
- Upgrade layer+manager implemented as a TESLA handler
- Upgrade scenarios planned for Thor, SFSACL, NFSv3/v4

Future Work

- Evaluate lazy vs. eager state transforms
- Recover from buggy upgrades
- Allow the upgrade infrastructure to upgrade
- Investigate upgrades for sensor networks