Modular Software Upgrades For Distributed Systems

Authors:
Sameer Ajmani
Barbara Liskov
Liuba Shrira

Presentation By:
Aldrin Montana

Requirements

1. Modularity
2. Generality
3. Transform persistent state (Schema Evolution?)
4. Automatic Deployment
5. Controlled Deployment
6. Continuous Service (Zero downtime)

Contributions

1. First complete solution for automatic and controlled upgrades in distributed systems
2. A way to understand and specify mixed mode

Data Model

- Distributed system is a collection of objects
- An object has:
    - Identity
    - Type (defines behavior)
    - State (a portion may be persistent)
        - Object reinitializes itself from persistent state
- Objects call each other’s methods

Data Model

An object is instance of a class that defines how it implements its type

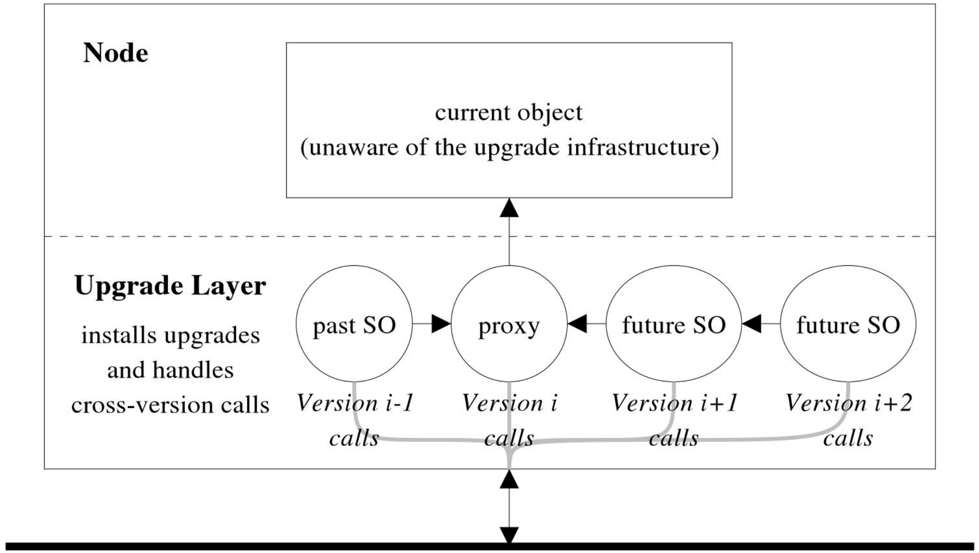
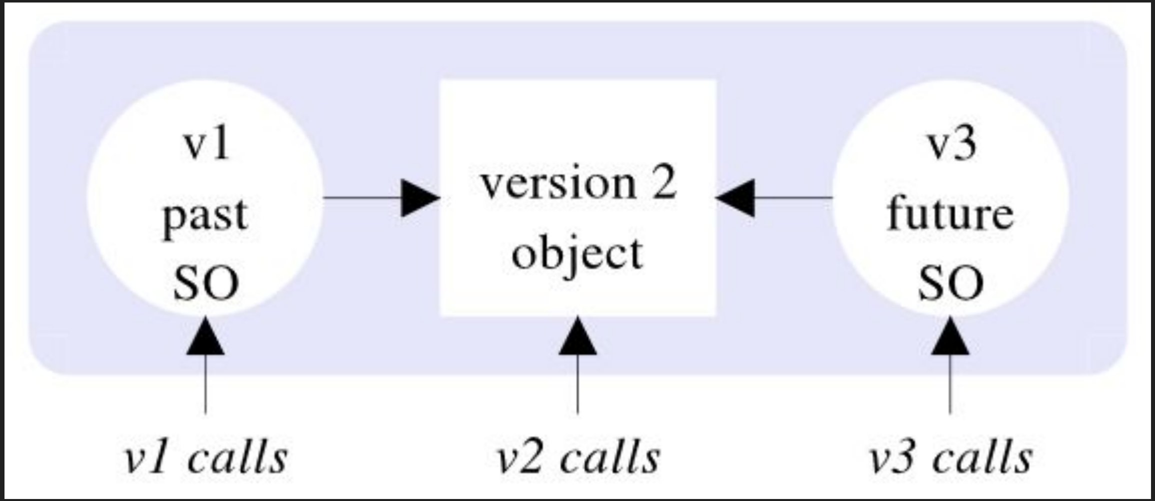


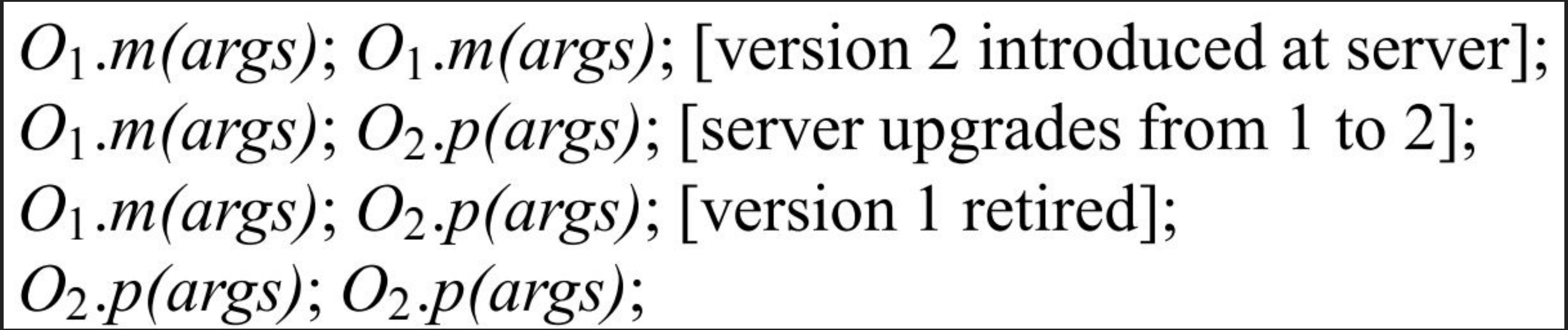
Fig. 1. *The structure of a node.*

Terms

- Simulation Object
- Delegate Object
- Invariant
- Type
- Event
- Call



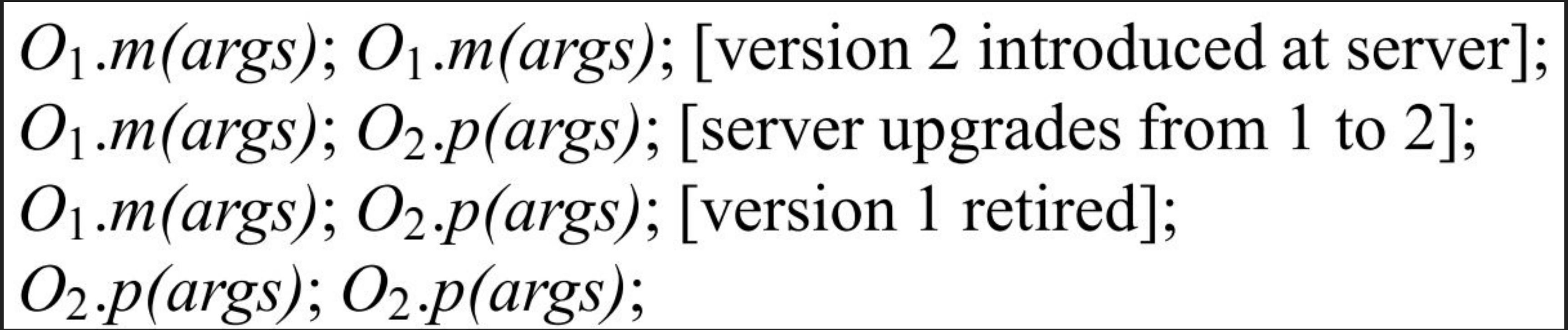
Type Requirement



$O_1.m(args)$; $O_1.m(args)$; [version 2 introduced at server];
$O_1.m(args)$; $O_2.p(args)$; [server upgrades from 1 to 2];
$O_1.m(args)$; $O_2.p(args)$; [version 1 retired];
$O_2.p(args)$; $O_2.p(args)$;

Type Requirement

- The class implementing a future SO must implement the new type
- The class implementing a past SO must implement the old type
- Effects of interleaving need to be defined



$O_1.m(args)$; $O_1.m(args)$; [version 2 introduced at server];
$O_1.m(args)$; $O_2.p(args)$; [server upgrades from 1 to 2];
$O_1.m(args)$; $O_2.p(args)$; [version 1 retired];
$O_2.p(args)$; $O_2.p(args)$;

Sequence and Disallow

- Sequence Requirement
  - An event is a call, upgrade, or introduction of a version
  - Calls to a current object or SO must reflect effects of calls to other objects
- Disallow Constraint
  - Calls to current object cannot be disallowed
  - Current object provides “real behavior”

Specifying Upgrades

- Three Parts:

Specifying Upgrades

- Three Parts:
    a. Invariant
    b. Mapping function
    c. Shadow methods

Specifying Upgrades - Invariant

- Corresponding set of old and new object state
- Must be **total**
  a. One-to-One?
  b. For each legal state O_{new} of T_{new}, there is some legal state, O_{old} of T_{old}
  c. And vice versa

Specifying Upgrades - Mapping Function

- Initial state for O_{new} given state of O_{old}
- Must be total and establish the invariant

State of new file system given old file system?

Specifying Upgrades - Shadow Methods

- Corresponding set of old and new methods
    a. Preconditions must hold
    b. Invariant must be preserved

Examples - ColorSet with direct simulation

- SO.insertColor(1, red)
- O.delete(1)
- O.insert(1)
- SO.getColor(1)

Examples - ColorSet with direct simulation

- SO.insertColor(1, red)
- O.delete(1)
- O.insert(1)
- SO.getColor(1)

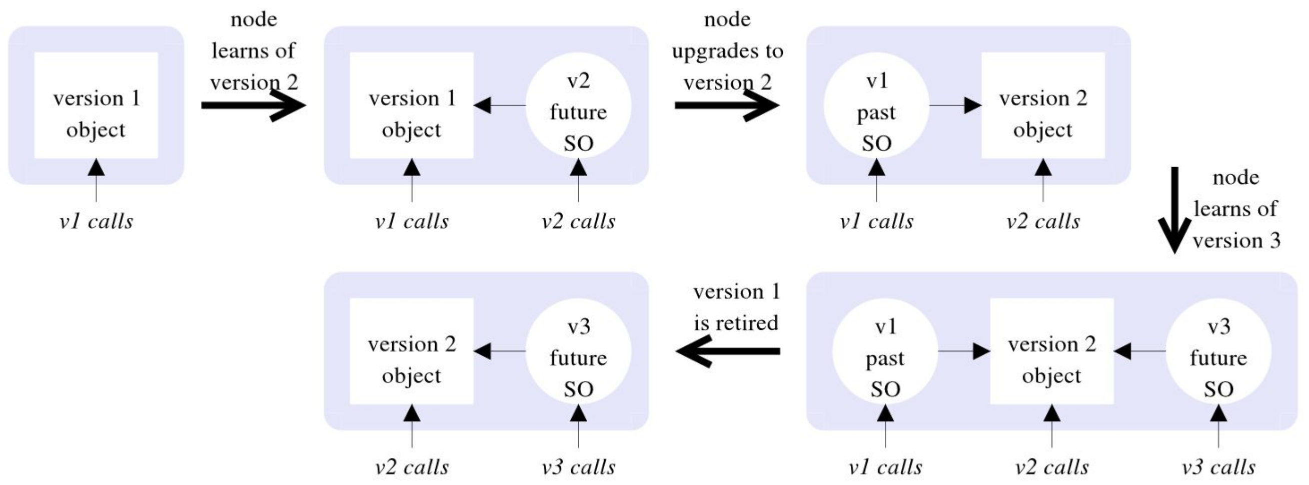


Fig. 2. *The direct approach, presented as a sequence of states of a node. Large arrows are state transitions. In each state, the box is the current object, and the circles are SOs. Objects may delegate calls as indicated by the small arrows. Each object handles calls only for its own version.*

Examples - interceptor simulation

- SO.insertColor(1, red)
- O.delete(1)
- O.insert(1)
- SO.getColor(1)

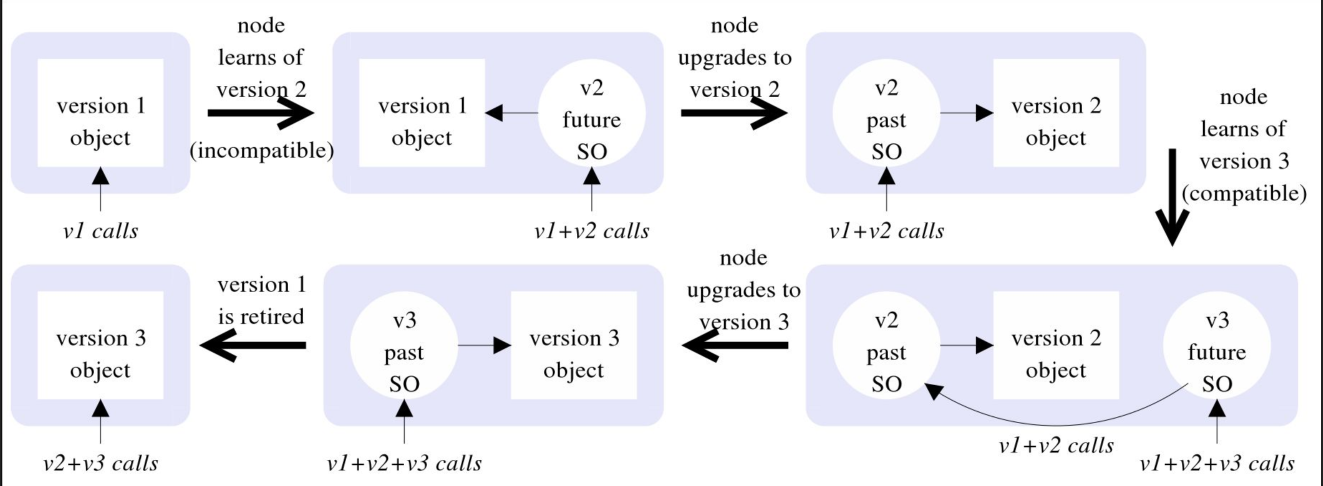


Fig. 3. *The interceptor approach, presented as a sequence of states of a node. Large arrows are state transitions. In each state, the box is the current object, and the circles are SOs. Objects may delegate calls as indicated by the small arrows. One object in each state intercepts all calls.*

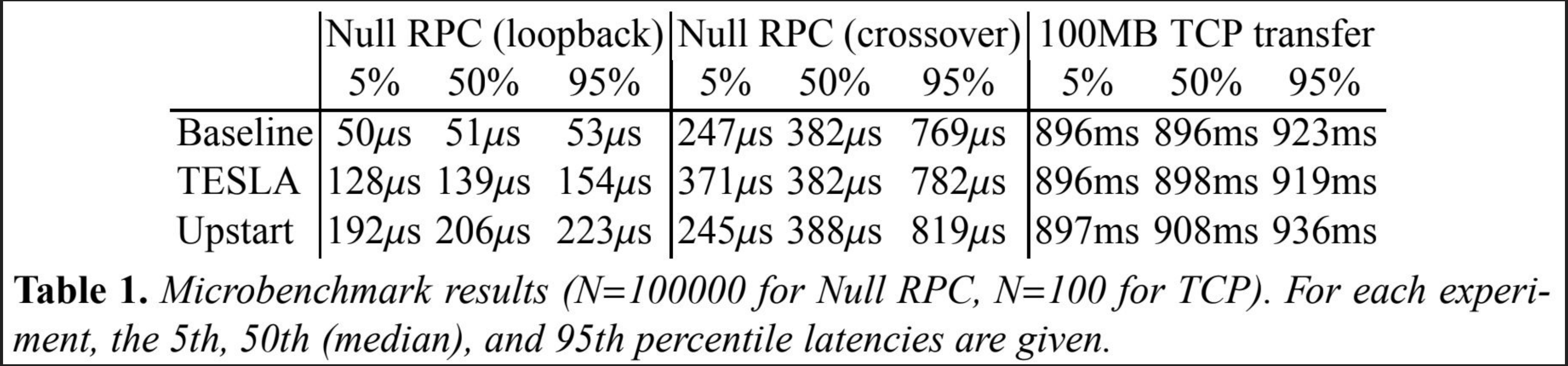
Transform and Scheduling Functions

- TF must implement identity mapping
    - post-TF of past SO ←→ pre-TF of old object
    - post-TF of new object ←→ pre-TF of future SO
    - Must be restartable (idempotent?)
- SF runs on node itself
    - Access to UDB (central knowledge) and LDB (local knowledge)
    - Strategies: Rolling upgrade, Big flip, Fast reboot

Evaluation - microbenchmarks

The overhead imposed by the upgrade layer

when no upgrades are happening



	Null RPC (loopback)			Null RPC (crossover)			100MB TCP transfer		
	5%	50%	95%	5%	50%	95%	5%	50%	95%
Baseline	50μs	51μs	53μs	247μs	382μs	769μs	896ms	896ms	923ms
TESLA	128μs	139μs	154μs	371μs	382μs	782μs	896ms	898ms	919ms
Upstart	192μs	206μs	223μs	245μs	388μs	819μs	897ms	908ms	936ms

Table 1. *Microbenchmark results (N=100000 for Null RPC, N=100 for TCP). For each experiment, the 5th, 50th (median), and 95th percentile latencies are given.*

Discuss