Fast Choreography of Cross-DevOps Reconfiguration with Ballet

Multi-Site OpenStack Case Study

Jolan Philippe, Antoine Omond, Hélène Coullon, Charles Prud'Homme, Issam Raïs

November 9, 2023

STACK, IMT Atlantique

Reconfiguration of Service oriented architecture

Reconf. plan:
$action_1$
$action_2$
...
$action_n$

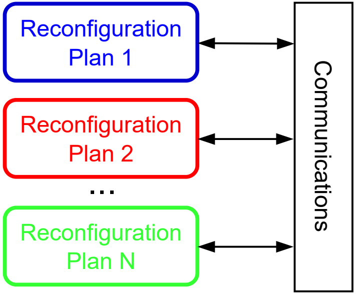
Postdoc objectives

⇒ Infer reconfiguration local actions

⇒ Coherent overall reconfiguration

Challenges

- Locally: Partial view of the system
- Need for communications decentralized operation



Case study: Deploy or update OpenStack with Galera cluster of MariaDB

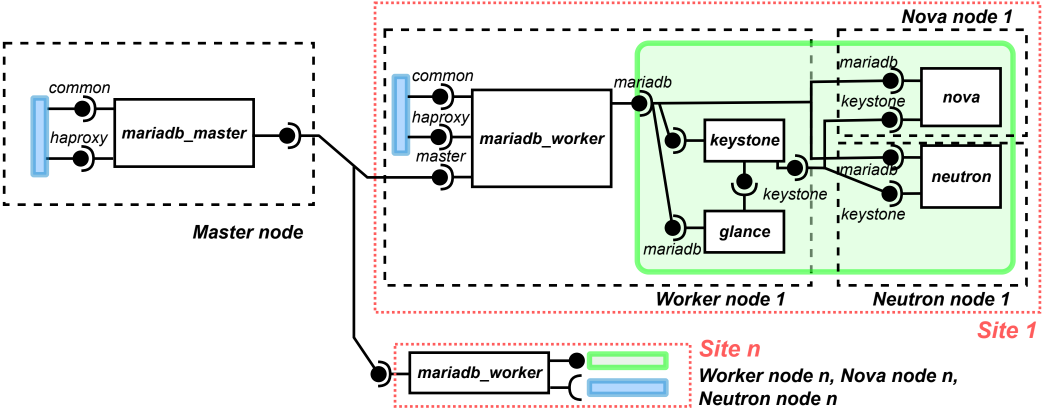


Figure 1: Assembly of a multi-site OpenStack with a Galera cluster of distributed MariaDB databases.

Motivation

- When facing complex projects: cross-functional and cross-geographical DevOps teams
- Each team tackles a set of services and associated DevOps operations on different parts of the project
- Each team usually use a centralized local DevOps tool with a local vision of the state of their part

Problem

DevOps operations applied by one DevOps team can necessitate operations on other elements tackled by other DevOps teams. This is in practice handle manually between teams as DevOps tools apply operations in a centralized manner.

Naive solution

Using a centralized tool on top of all DevOps teams is not suitable for scale and fault tolerance reasons.

Related work: Muse (Sokolowski et. al.)

Existing solution: Designed for components with fixed life cycle, and not efficient

Ballet overview

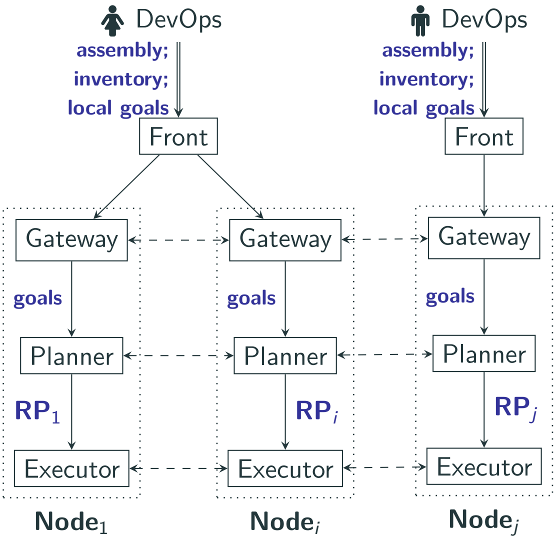


Figure 2: Ballet overview

- Declarative input
- Automatic planning
- Efficient reconfiguration

Gateway

Global knowledge building of reconfiguration goals

Planner

Decentralized inference of reconfiguration plans (RPs)

Executor

Coordinated execution of RP

Usage of Ballet

Ballet’s usage: Developer’s concern

- Specify components’ life-cycle (places, transitions, ports)
- Defining components’ dependencies

⇒ Scripts for deployment or update

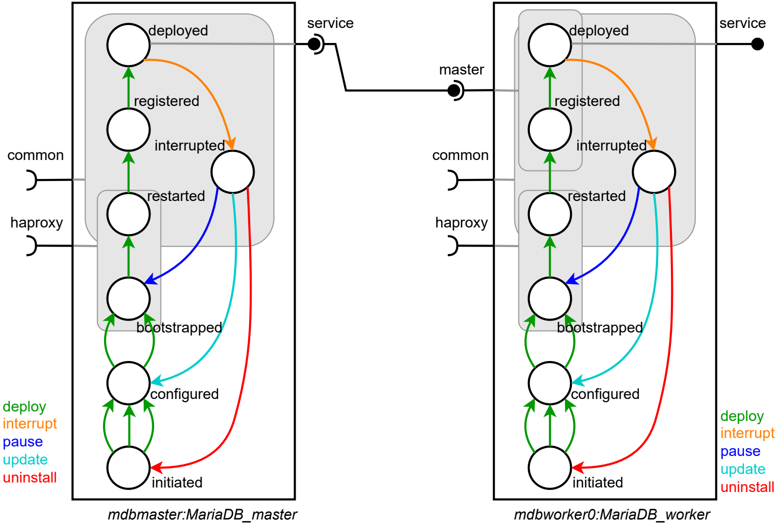


Figure 3: MariaDB_Master and MariaDB_Worker components

Ballet's usage: DevOps's concern

Listing 1: Language to define reconfiguration goals for DevOps usage

```
<goals> ::= behaviors: <bhvr_list>
            ports: <port_list>
            components: <comp_list>
<bhvr_list> ::= ...
<bhvr_item> ::= - forall: <bhvr_name>
              | - component: <comp_name>
                  behavior: <bhvr_name>
<port_list> ::= ...
<port_item> ::= - forall: <port_status>
              | - component: <comp_name>
                  port: <port_name>
                  status: <port_status>
<comp_list> ::= ...
<comp_item> ::= - forall: <comp_status>
              | - component: <comp_name>
                  status: <comp_status>
```

Language

Declarative language for defining reconfiguration goals

- **Behavior goal**: Specify a behavior that must be executed
- **Port goal**: Specify a port status (active, inactive)
- **State goal**: Specify a component state (specific, running, initial)

Case study reconfiguration

behaviors:
- ***component***: *mariadb_master*
  behavior: *update*

components:
- ***forall***: *running*

Ballet choreography engine

Execution language: Concerto-D (Antoine Omond's thesis)

Reconfiguration programs can

1. Create assemblies of components (software system)
2. Make this assembly evolve at runtime
3. Interact with the life cycle of components

The used language propose instructions for:

- Add/remove a component instance to the current assembly
- Connect/disconnect two component instances with compatible ports
- Push behavior to the behavior queue on a component instance
- Wait for a given component instance to execute a behavior

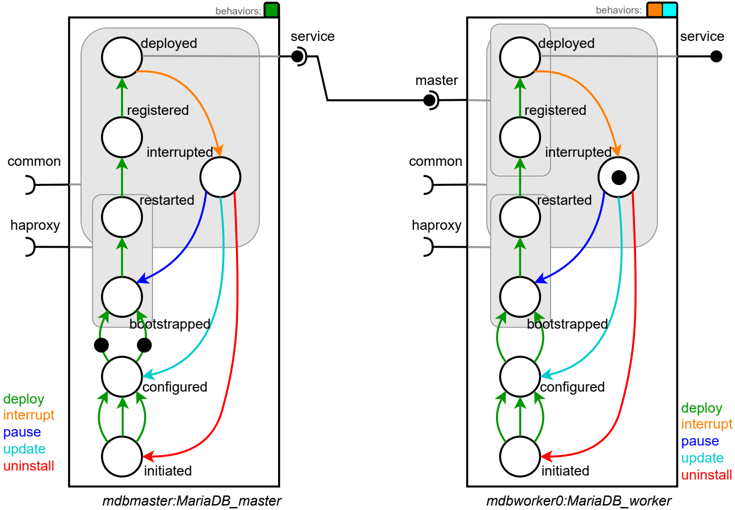
Decentralized execution: Concerto-D

mariadb_master's RP

pushB(master, interrupt)
pushB(master, update)
pushB(master, deploy)

mariadb_worker0's RP

pushB(worker, interrupt)
pushB(worker, update)
wait(master, interrupt)
pushB(worker, deploy)



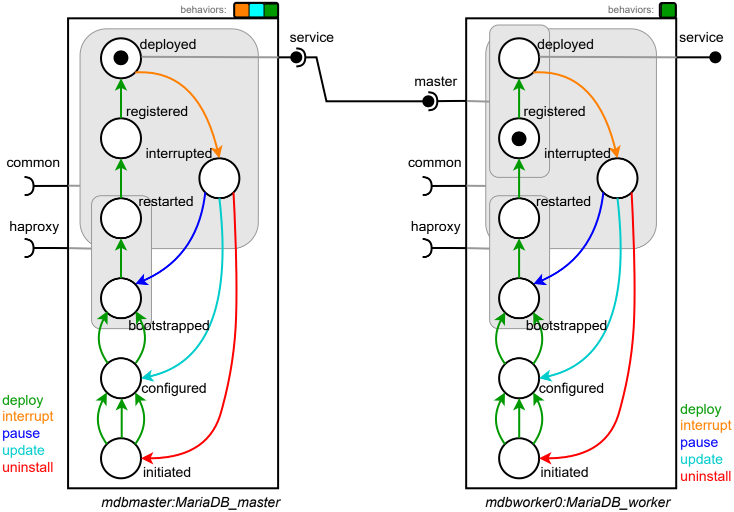
Failing example

mariadb_master's RP

pushB(master, interrupt)
pushB(master, update)
pushB(master, deploy)

mariadb_worker0's RP

pushB(worker, interrupt)
pushB(worker, update)
pushB(worker, deploy)



Approach for Ballet's planner

Local resolution

- **Purpose**: Find a sequence of behavior to execute
- **Hint**: Constraint programming approach

Constraint propagation

- **Purpose**: Inferring wait instructions (*i.e.*, synchro. bareer)
- **Hint**: Propagation based on Gossip algorithm
- **Hint**: Consensus using Paxos-like approach

CP for local planning

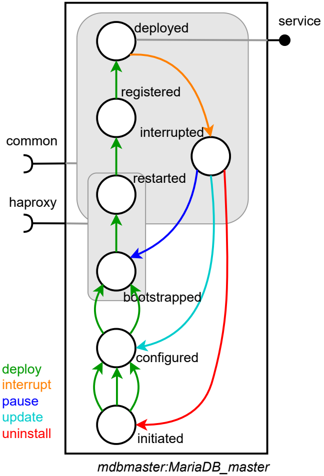
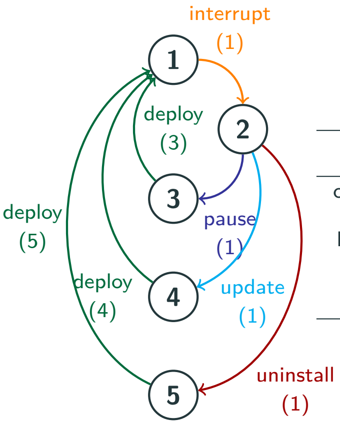


Figure 4: *MariaDB_master* control component



state	1	2	3	4	5
common	✓	✓	×	×	×
haproxy	×	×	✓	×	×
service	✓	×	×	×	×

Figure 5: Automaton representation of *Mariadb_master* component's life cycle with its matrix for ports statuses.

Message inference

Case study reconfiguration

behaviors:
- ***component***: *mariadb_master*
  behavior: *update*

components:
- ***forall***: *running*

Sequence := [interrupt, update, deploy]

States := [1, 2, 4, 1]

Port status :=

common:	[✓, ✓, ×, ✓]
haproxy:	[×, ×, ×, ×]
service:	[✓, ×, ×, ✓]

Must propagate constraints using messages:

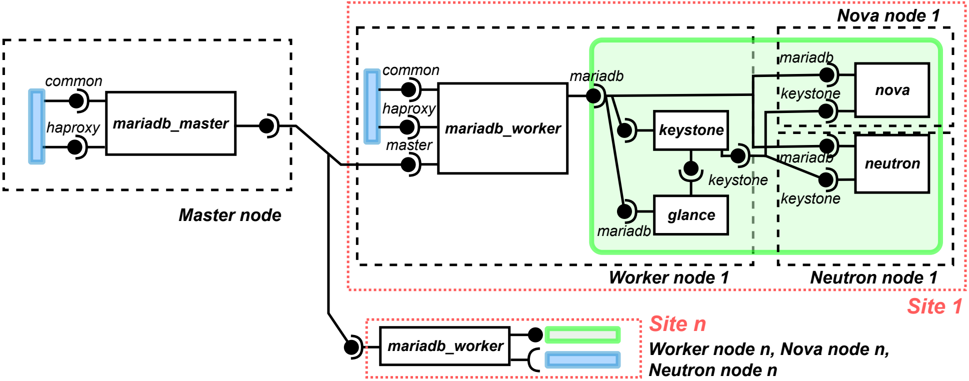
- "Components using **master**'s **common** must disconnect until **update** ends"

⇒ Message: (master, common, disconnect, update)

- "Components using **master**'s **service** must disconnect until **interrupt** ends"

⇒ Message: (master, service, disconnect, interrupt)

Constraint propagation



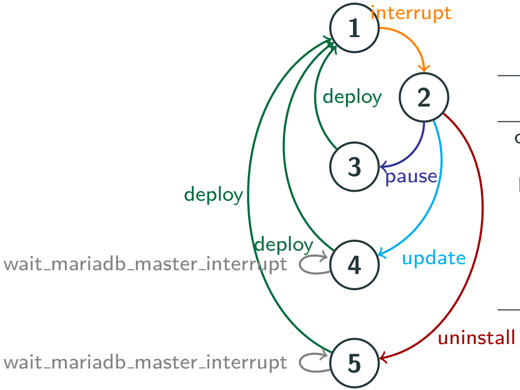
Propagated constraint (gossip + consensus) from mariadb_master for master's service

- mariadb_master ⇒ mariadb_worker
- mariadb_worker ⇒ keystone; glance; nova; neutron
- keystone ⇒ glance; nova; neutron

Enriched CP Model

Enriched CP problem

- Enriched automaton with synchronization instruction
- Additional constraint to have synchro. barrier in local plan



state	1	2	3	4	5
common	✓	✓	×	×	×
haproxy	×	×	✓	×	×
master	✓	✓	✓	×	×
service	✓	×	×	×	×

Figure 6: Enriched automaton representation of *Mariadb_worker*.

Experiments

Deployment and update of OpenStack with Galera cluster of MariaDB with $n \in [1, 2, 5, 10]$ sites, that is a total of $7 + 11 * n$ components.

Metric of interest

- For both the planner and the executor: Execution time
- For the planner: Inferred constraints, inferred actions, number of communications

Setup

- Results on $1 + 3 * n$ nodes Gros (Nancy) of Grid'5000
- Comparison to Muse (decentralized reconfiguration)
- Reproducible example on Grid'5000

Experimental results

Sc.	# Sites	Ballet			Muse	Gain
		Planning	Execution	Total		
Deploy	1	1.69s	306.02s	307.71s	536.57s	42.7%
	2	1.78s	306.09s	307.86s	536.69s	42.6%
	5	1.77s	306.19s	307.97s	537.09s	42.7%
	10	2.02s	306.14s	308.19s	538.13s	42.7%
Update	1	3.36s	416.84s	420.20s	555.56s	24.4%
	2	4.39s	416.92s	421.31s	555.70s	24.2%
	5	6.05s	417.17s	423.22s	556.08s	24.0%
	10	5.97s	417.46s	423.43s	556.77s	24.0%

Table 1: Comparison of time for planning and executing a deployment and an update of the `MariaDB_master` instance with Ballet and Muse.

Experimental results

Sc.	#Sites	#Constraints	#Instructions	#Messages
Deploy	n	$7 + 11 * n$	$7 + 11 * n$	0
	1	18	18	0
	2	29	29	0
	5	62	62	0
	10	117	117	0
Update	n	$3 + 20 * n$	$8 + 11 * n$	$9 * n$
	1	23	19	9
	2	43	30	18
	5	103	63	45
	10	203	118	90

Table 2: Results of the planning phase for the *deploy* and *update* scenario when varying the number of `Mariadb_workers` in a Galera cluster.

Concluding remarks

Postdoc contributions

- Ballet and SeMaFoR project
- Infer reconfiguration actions (CP model)
- Communication protocol
- Work under review for SANER2024

Target applications

- OpenStack, and CPS
- (SeMaFoR) Fog areas, smart cities, IoT devices, etc.

Perspectives

- Model-Driven Engineering approach for determining objectives
- Experiments on more topologies
- Formalization of Planner + Executor in Why3 for correctness

Backup

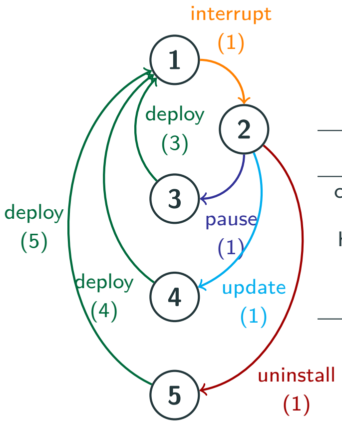
Ballet's usage: Developer's concern

Listing 2: Control component MariaDB master in Python

```
class MariaDB_Master(Component):
    def create(self):
        self.places = [ "initiated", "configured", "bootstrapped", "restarted",
                        "registered", "deployed", "interrupted"]
        self.transitions = {
            "configure0": ("initiated", "configured", "deploy", self.configure0),
            "configure1": ("initiated", "configured", "deploy", self.configure1),
            "configure2": ("initiated", "configured", "deploy", self.configure2),
            ...
        }
        self.dependencies = {
            "service": (DepType.PROVIDE, ["deployed"]),
            "haproxy": (DepType.USE, ["bootstrapped","restarted"]),
            ...
        }
        self.initial_place = 'initiated'
        self.running_place = 'deployed'

    def configure0(self):
      # concrete actions
```

CP Model



state	1	2	3	4	5
common	✓	✓	×	×	×
haproxy	×	×	✓	×	×
service	✓	×	×	×	×

Figure 8: Automaton representation of *Mariadb_master* component's life cycle with its matrix for ports statuses.

- $\textsc{Regular}(B, \Pi, s_{init}, S_{goal})$
- $s_{i+1} = inc_{\Pi}[s_i][b_i], \ \forall i \in 1..m$
- $\textsc{Count}(b, B, >, 0)$
- $status(p, s_{m+1}) = \Gamma_p$

where

$$\Gamma_p \in \{\texttt{active}, \texttt{inactive}\}$$
$$c_i = cost(s_i, b_i), \ \forall i \in 1..m$$
$$C = \textsc{Sum}([c_i \mid i \in 1..m])$$

Planner time

#Sites	Solving	Communications	Total
1	1.58 (0.06)	1.78 (0.44)	3.36 (0.43)
2	1.53 (0.13)	2.85 (1.62)	4.39 (1.72)
5	1.59 (0.06)	4.47 (0.92)	6.05 (0.91)
10	2.61 (0.17)	0.26 (0.01)	5.97 (0.63)

Table 3: Average duration in seconds (and standard deviation) to calculate the plans for the *update* scenario.

Gossip + Protocol

https://docs.google.com/presentation/d/18asPwHJ4HOZqAlmQqLEI5V-hX38_robjgia62bNtrig/edit?usp=sharing

Full execution with failure

https://docs.google.com/presentation/d/1pe4HXdohWJyxwJHHbdmIitxnkCEN_
UEdfBWqZRvZQbc/edit?usp=sharing

Cyber Physical System (CPS) performance

https://docs.google.com/presentation/d/1WwMoAma8trummqHhtNLrDV-AL7t4WSIZ7PMY5ZI-Jk0/edit?usp=sharing