Fast Choreography of Cross-DevOps Reconfiguration with Ballet

Multi-Site OpenStack Case Study

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

June 4th, 2024 - Journées GDR GPL

STACK team, IMT Atlantique - SeMaFoR project

Hello world

Jolan Philippe, Postdoc in the SeMaFoR project
IMT Atlantique, STACK Team (Inria)

Work with Hélène Coullon and Charles Prud'Homme

jolan.philippe@imt-atlantique.fr

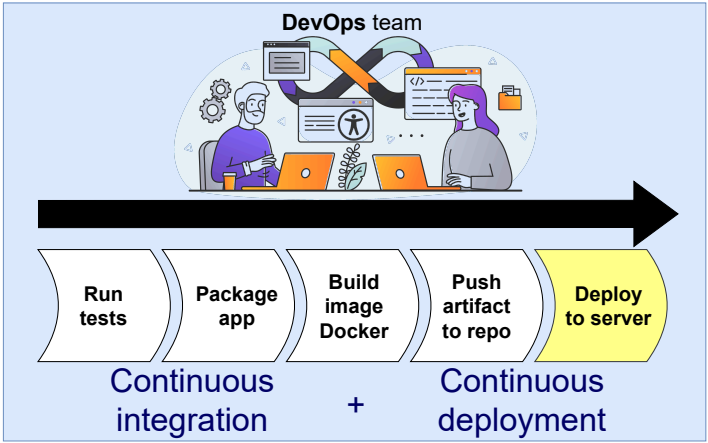
Previous work on

- Geometric approach for clustering, and cryptography
- Algorithmic Skeletons on Lists and Trees
- Study design-choice of distributed transformation engines

More details on: `https://jolanphilippe.github.io/`

DevOps deployment and reconfiguration

Responsible for **development**



Responsible for **operations**

⇒ Continuous deployment then **reconfiguration**

Cross-DevOps reconfiguration

Responsible for **development** of S1

DevOps team

Run tests on S1 → Package S1 → Build image for S1 → Push S1 to repo → Deploy S1

Responsible for **operations** on S1

Responsible for **development** of S2 which uses S1

DevOps team

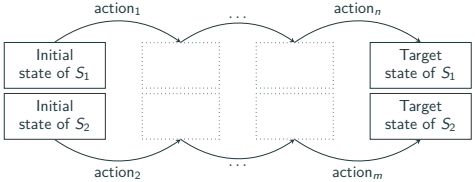
Run tests on S2 → Package S2 → Build image for S2 → Push S2 to repo → Deploy S2 (after S1?)

Responsible for **operations** on S2 (using S1)

Decentralized reconfiguration

Centralized approach

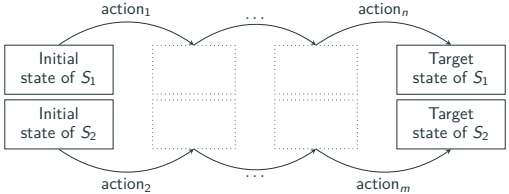
Central Plan
action_1 on S_1
action_2 on S_2
...
action_m on S_2
action_n on S_1



Decentralized approach

Plan for S_1
action_1
...
action_n

Plan for S_2
action_2
...
action_m



Example: Deploy distributed databases

Galera cluster:

Database Master (DM) plan

1. Configure the service
2. Bootstrap the database
3. Start the service
4. Expose API

Database Worker$_i$ (DW$_i$) plan

1. Configure the service
2. Bootstrap the database
3. Start the service
4. Register to master
5. Expose API

- **Component granularity:** DM $<<$ DW$_i$
- **Lifecycle granularity:** DM(4) $<<$ DW$_i$(4) (partial order)

Example: Update distributed databases

Galera cluster:

Database Master (DM) plan

1. Interrupt the service
2. Make the update
3. Start the service
4. Expose API

Database Worker$_i$ (DW$_i$) plan

1. Interrupt the service
2. Make the update
3. Register to master
4. Start the service
5. Expose API

- **Component granularity?** Destroy DW$_i$ << Update DM << Deploy DW$_i$
- **Lifecycle granularity:** DW$_i$(1) << DM(1) & DM(4) << DW$_i$(4)

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

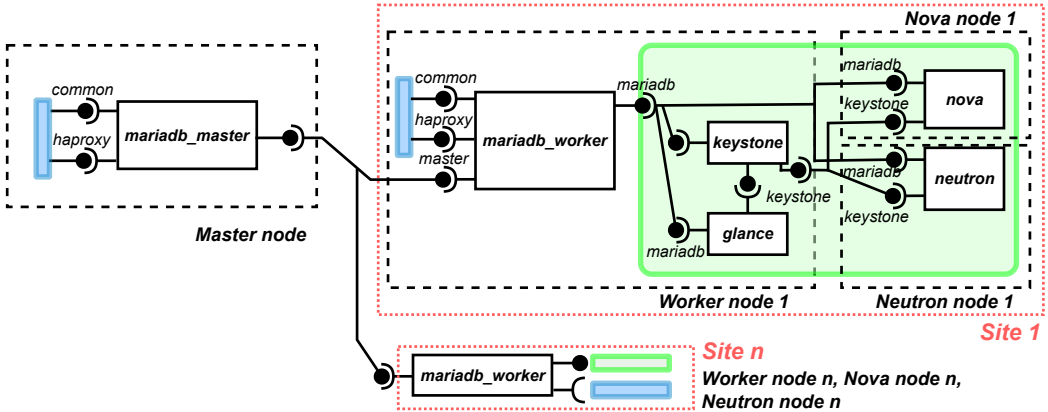


Figure 1: Assembly of a multi-site OpenStack with a Galera cluster of distributed MariaDB databases.

Decentralized reconfiguration

Challenges

- Human communication between DevOps is error-prone
- Infer reconfiguration actions with a local view
- Execute concurrently reconfiguration actions

Naive solution

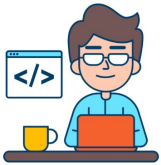
Using a centralized tool (e.g., Terraform, Kubernetes) on top of all DevOps teams is not suitable for scale and fault tolerance reasons.

Decentralized solution

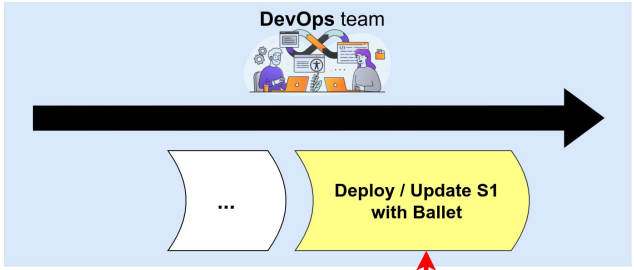
Make a plan for each DevOps team, and execute them concurrently.

Muse (Sokolowski et. al.) covers cross-DevOps decentralized reconfiguration with planning, but inefficient because of the fixed life cycles (i.e., on-off mode for resources).

Ballet for DevOps

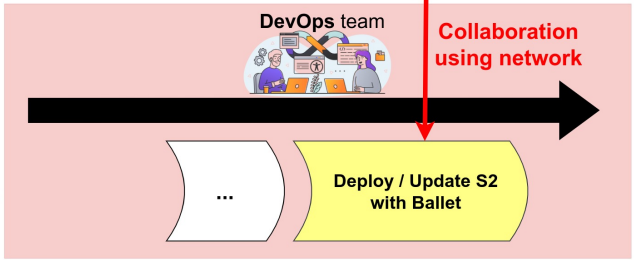


Responsible for **development** of S1



Responsible for **operations** on S1

Responsible for **development** of S2 which uses S1



Responsible for **operations** on S2 (using S1)

Ballet's modules overview

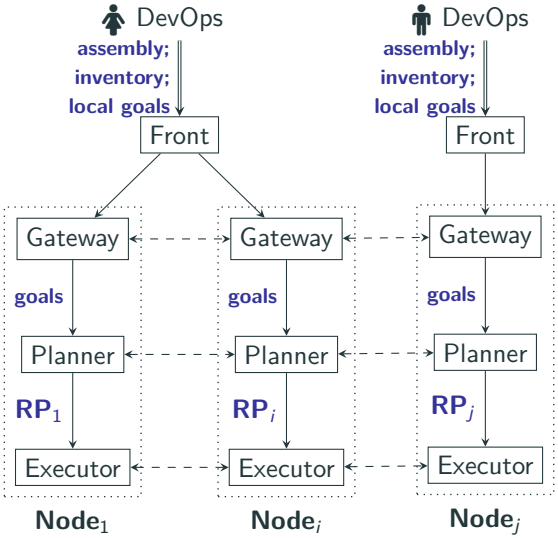


Figure 2: Ballet overview

- Decentralized tool (one instance of Ballet on each node)
- Declarative input
- Reconfiguration with automatic planning and efficient execution

Gateway

Global knowledge building of reconfiguration goals

Planner

Decentralized inference of reconfiguration plans (RPs)

Executor

Coordinated execution of RP

Developers' concern

Life-cycle and dependencies

Simple language to define component

- **Places**: milestones of the reconfiguration
- **Behaviors**: interface of actions for the DevOps
- **Transitions**: concrete actions between places, associated to behaviors
- **Ports**: Provide (resp. use) information to (resp. from) external components
  - Ports are bounded to places and transitions

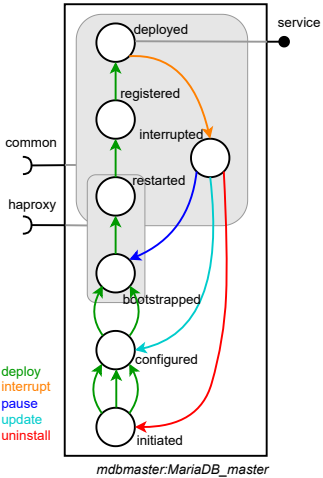


Figure 3: Visual representation of a component for MariaDB

Developer's concern

Listing 1: Control component MariaDB master in PYTHON

```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
```

DevOps' concern

Target assembly (YAML)

- A list of components to appear
- How components are connected

Reconfiguration goals

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)

Listing 2: 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>
```

Example - OpenStack assembly, update MariaDB master

Target assembly:

```
components:
  mdbmaster:
    type: 'MariaDB_master'
  facts:
    type: 'Facts'
  common:
    type: 'Common'
  haproxy:
    type: 'HAProxy'
  ...

connections:
  - facts, service, common, facts_service
  - facts, service, haproxy, facts_service
  - ...
  ${{ each site in range(0, sites) }}:
    - mdbmaster, service, mdbworker${{ site }}, master_service
```

Goals for deployment

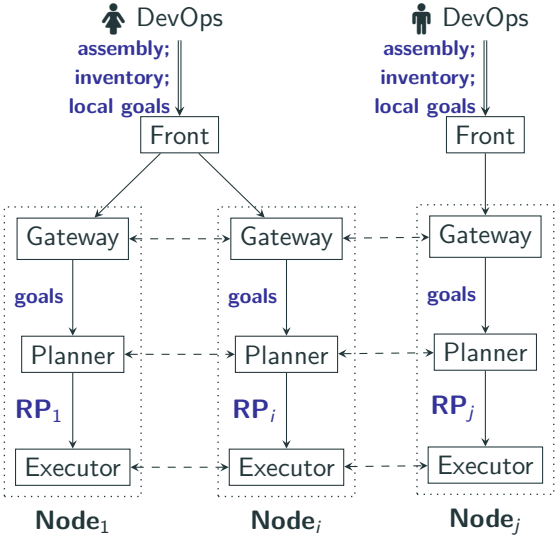
components:
- ***forall***: *running*

Goals for update database

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

components:
- ***forall***: *initial*

Reconfiguring process with Ballet



Outline, for clarity reasons

1. Skip gateway since there is no scientific challenge
2. Start with the execution, to understand planner's challenges
3. Followed by the planning

Decentralized execution

Concerto-D

- Based on Concerto model
  - As efficient as Concerto
  - Decentralized version
  - A model that includes communications inter-nodes
- One controller per node

Reconfiguration program

Within such a model, plans can

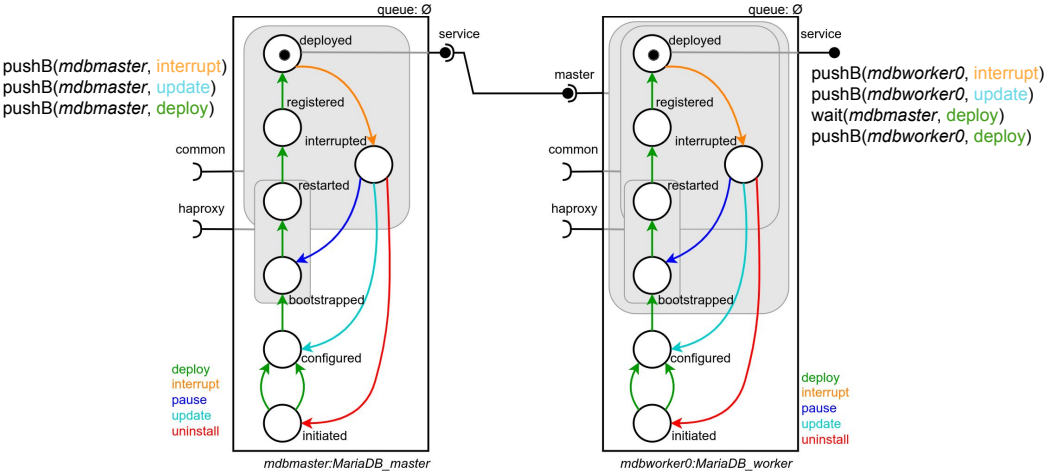
1. Create assemblies of components (software system)
2. Make this assembly evolve at runtime
3. Interact with the life cycle of components

Concerto-D language

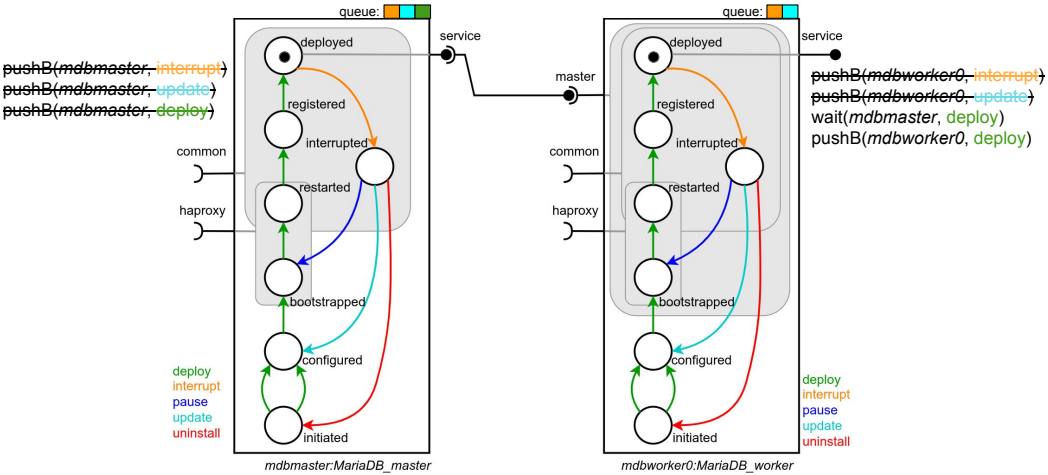
The used language propose instructions for:

- 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

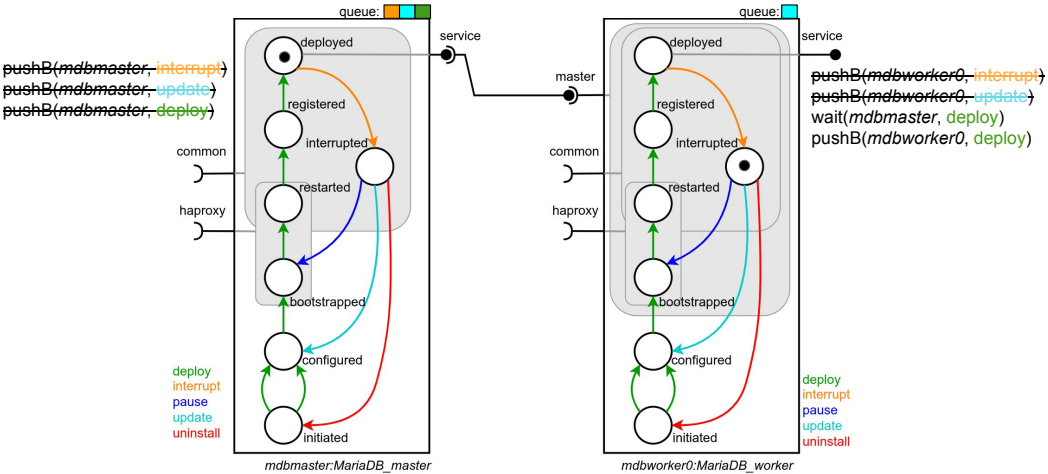
Program example



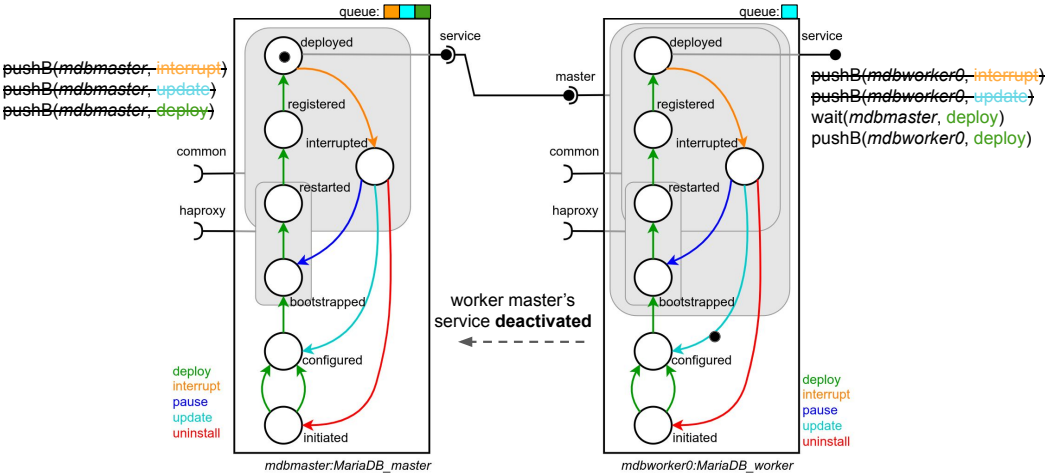
Program example



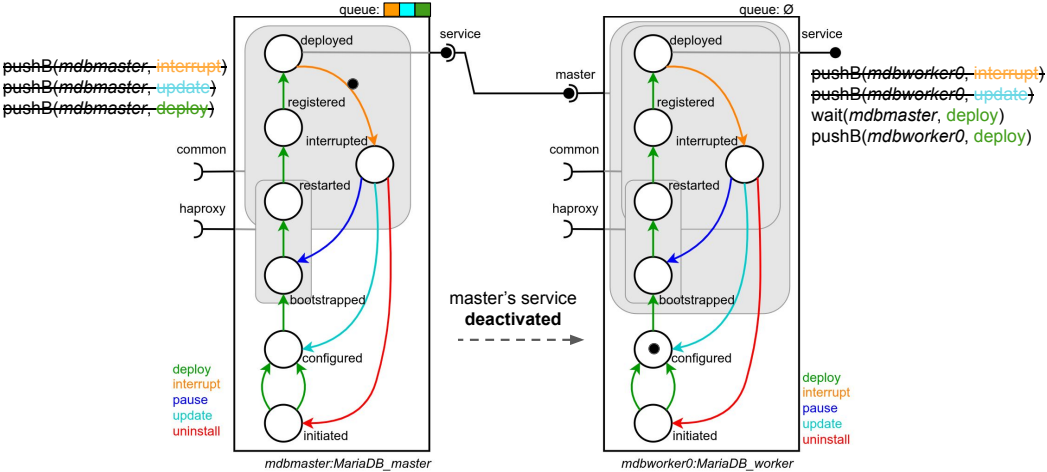
Program example



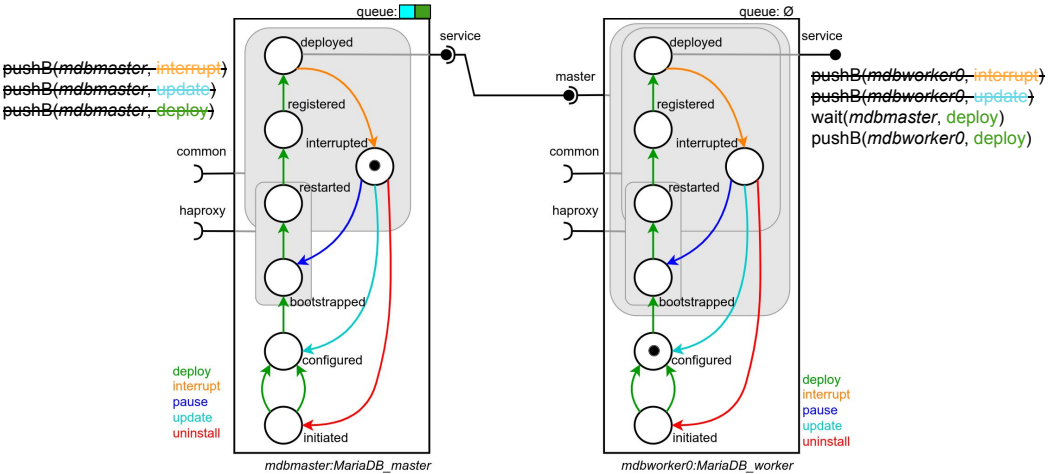
Program example



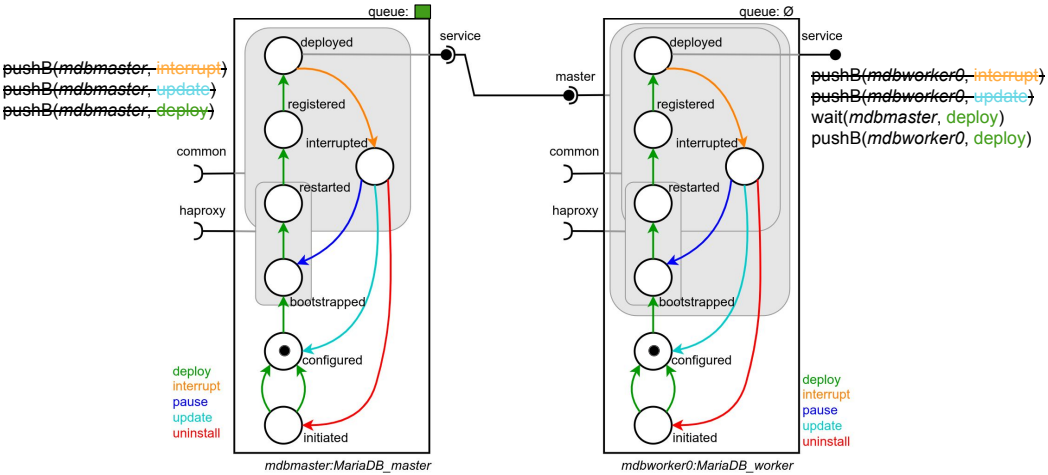
Program example



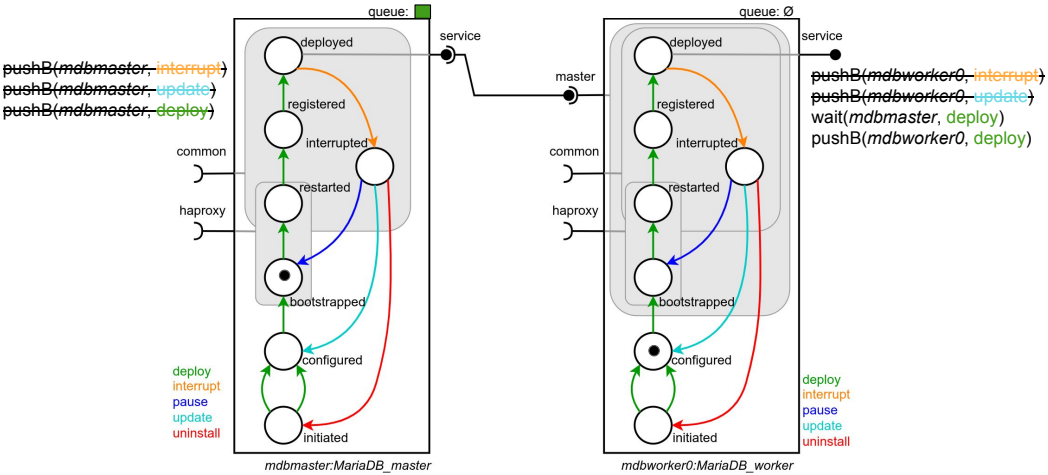
Program example



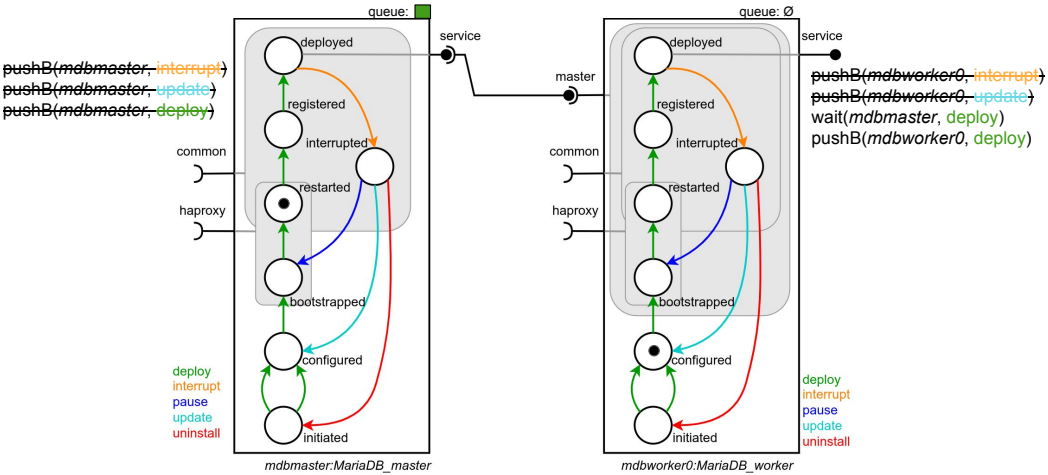
Program example



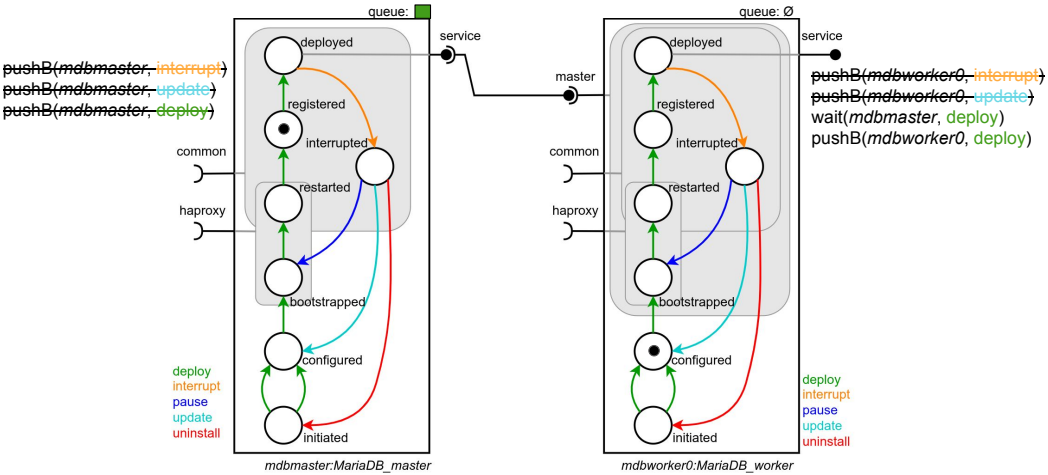
Program example



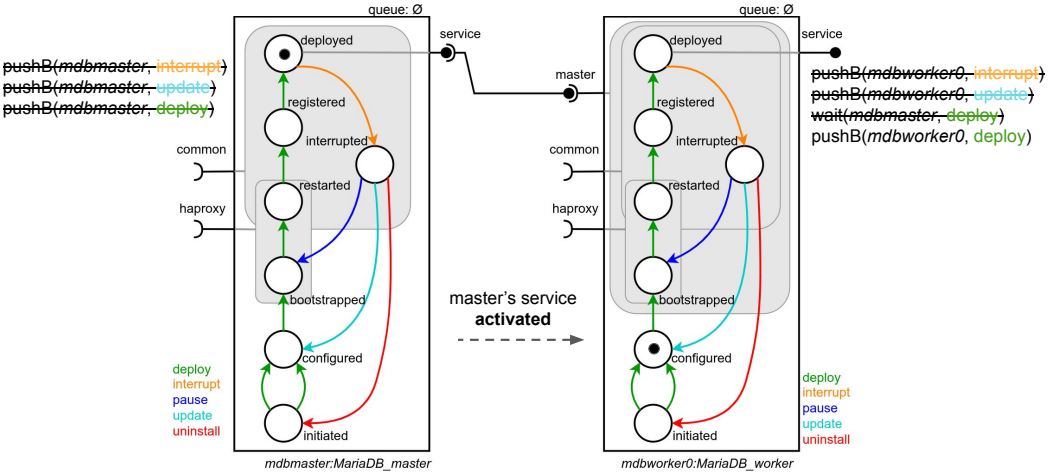
Program example



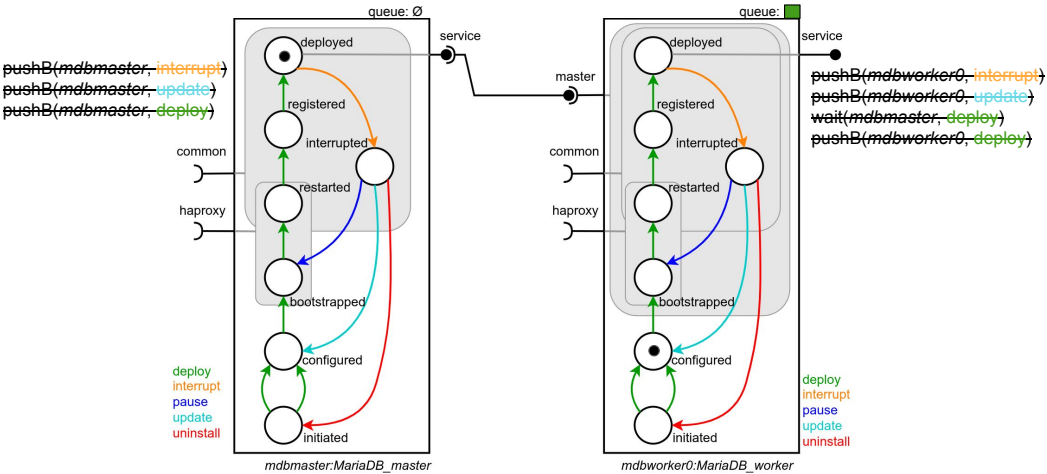
Program example



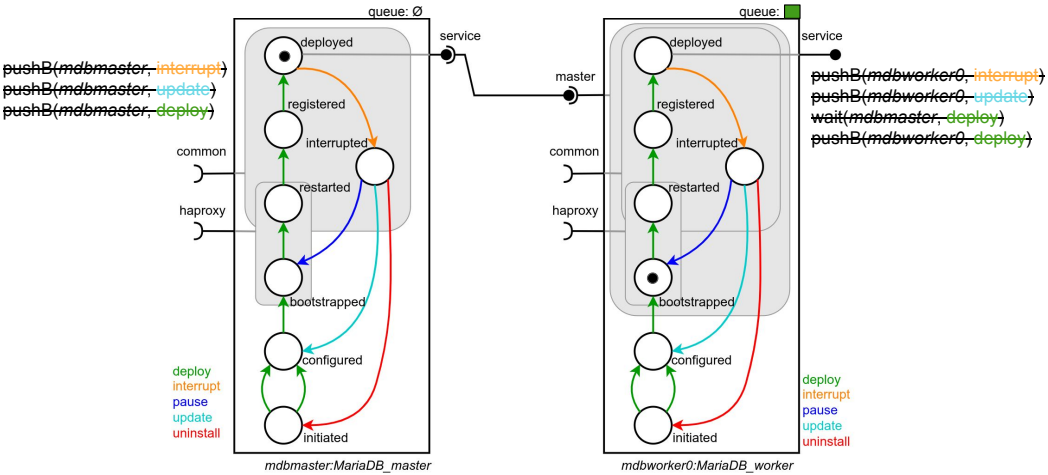
Program example



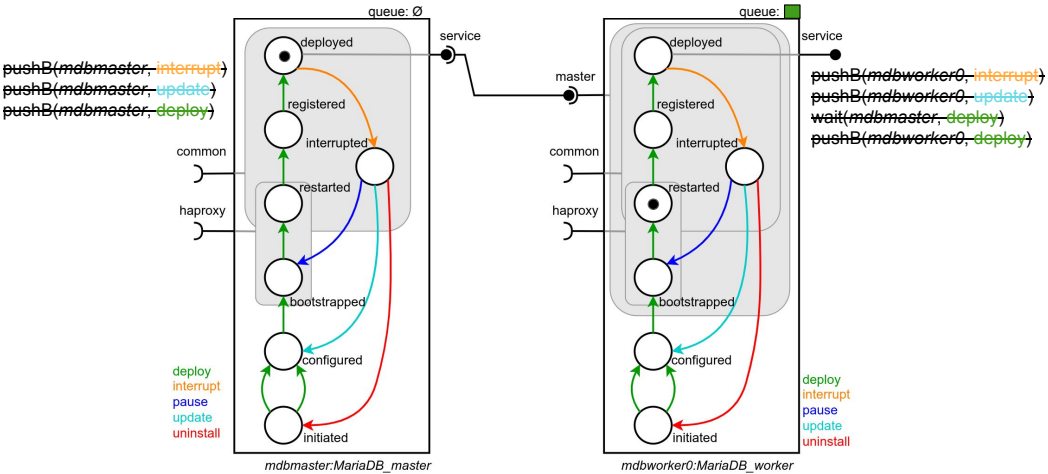
Program example



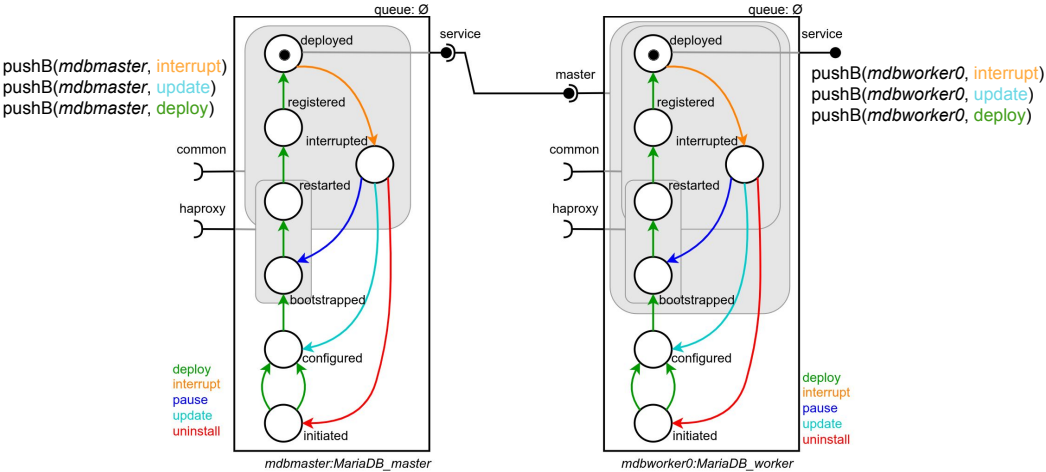
Program example



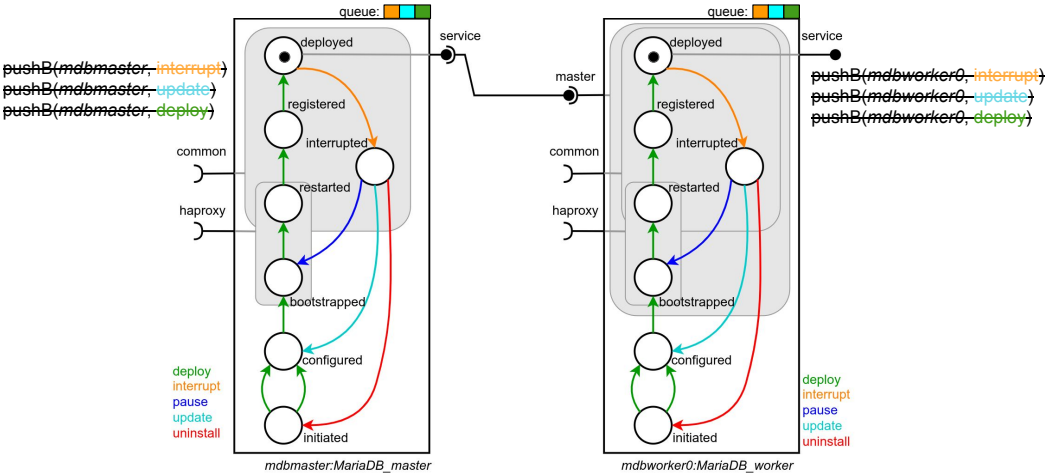
Program example



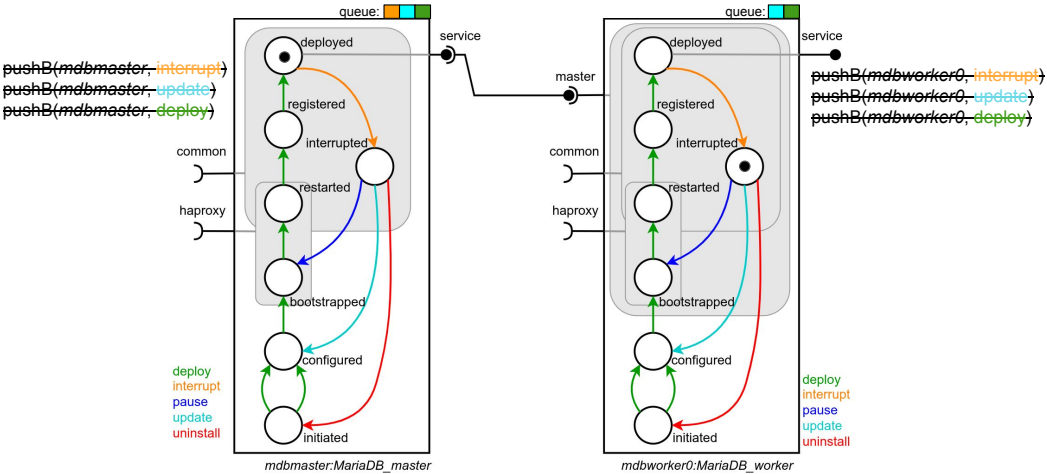
Failing example



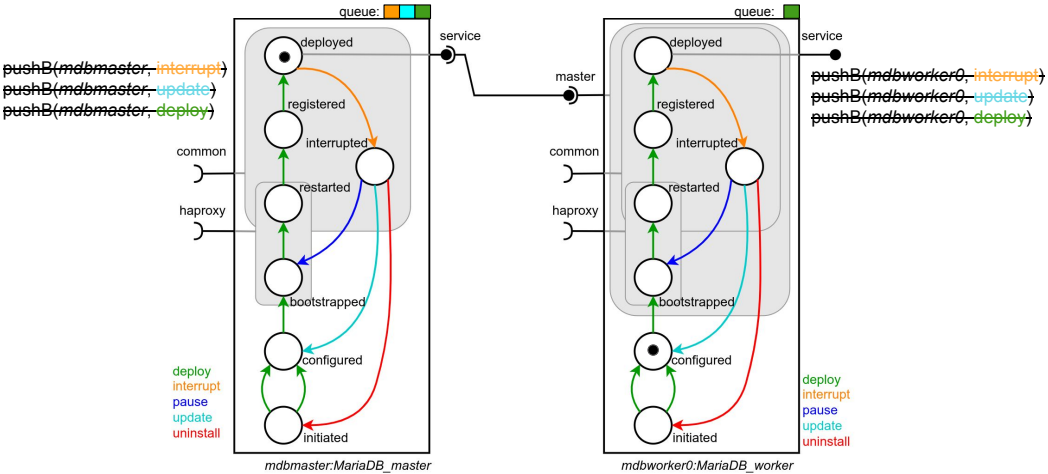
Failing example



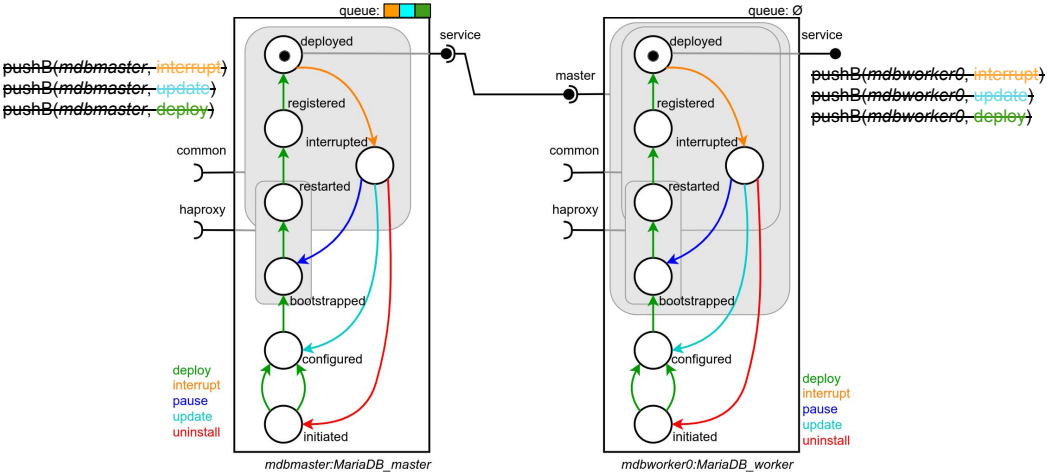
Failing example



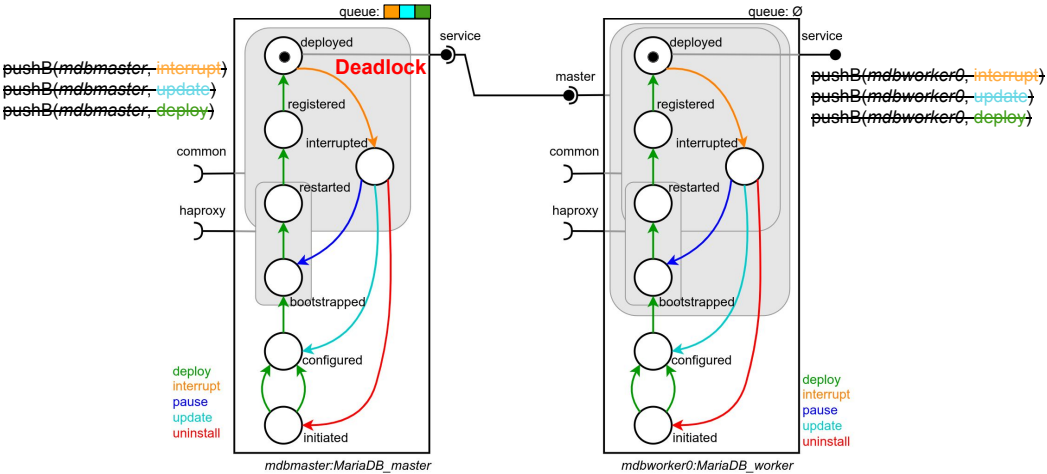
Failing example



Failing example



Failing example



Planner - Purpose and challenge

Goal: Infer local reconfiguration plan

Assembly

- **Goal**: Infer add/connect and delete/disconnect instructions
- **Approach**: Make a diff between assembly.yaml and current state of the assembly
- **Challenge**?

Local behaviors and sync. barriers

- **Goal**: Find a sequence of pushB and wait
- **Approach**: Constraint programming
- **Challenge**: Infer synchronization barriers (i.e., wait instructions)

Global approach

Local resolution

- Model components as automata
- Find a word in the automata
- Add constraint on the word to meet with reconfiguration goals

Constraint diffusion

- Analyze what would be the impact on port
- Infer messages to send new constraints to neighbors
- Gossip approach

Example: MariaDB master as an automaton

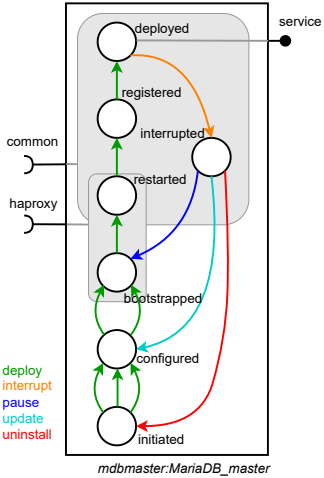
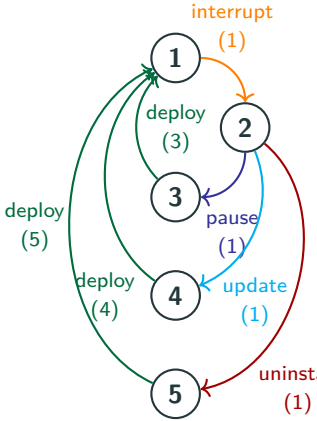


Figure 4: *MariaDB_master* control component



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.

Find a sequence for MariaDB master

Case study reconfiguration

behaviors:

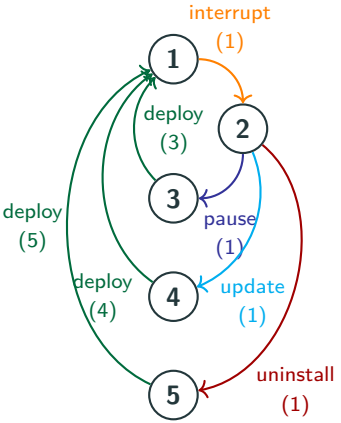
- ***component***: *mariadb_master*
  behavior: *update*

components:

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

CP Result

Sequence :=	[interrupt, update, deploy]	
States :=	[1, 2, 4, 1]	
Ports statuses :=	common:	[✓, ✓, ×, ✓]
	haproxy:	[×, ×, ×, ×]
	service:	[✓, ×, ×, ✓]



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

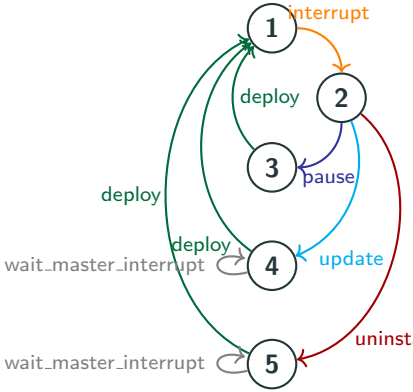
⇒ Message: (master, service, disconnect, interrupt)

Enrich MariaDB workers model

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

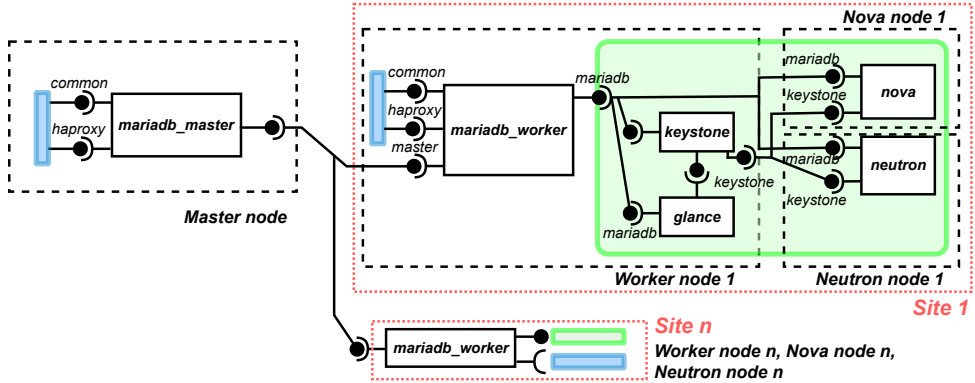
⇒ Message: (master, service, disconnect, interrupt)

⇒ Constraint: the use of master's service must be turned off during the reconfiguration, and a wait instruction must be placed



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

Constraint diffusion



Diffuse constraint from mariadb_master for master's service

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

Implementation of Ballet

- Open source: `https://github.com/Concerto-D/Ballet`
- about 10000 LoC (Python)
- Communications are operated with gRPC
- Planner uses MiniZinc modeling language with C++ Gecode solver as backed

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 - Performance

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 - Inference

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.

Few more words

Ongoing and future work

1. Study and management of unsat reconfiguration
   - Find the minimum Minimal Unsatisfiable Subsets of Constraints
   - Return conflicts to DevOps teams
2. Maude model of Ballet's executor (Concerto-D) for reasoning
3. Study safety properties of reconfiguration in modern IaC languages (Terraform, Pulumi)

Publications

[SANER24] Jolan Philippe, Antoine Omond, Hélène Coullon, Charles Prud'Homme, and Issam Raïs. Fast Choreography of Cross-DevOps Reconfiguration with Ballet: A Multi-Site OpenStack Case Study.

[ICE24] Farid Arfi, Hélène Coullon, Frédéric Loulergue, Jolan Philippe, and Simon Robillard. A Maude Formalization of the Distributed Reconfiguration Language Concerto-D.

Conclusion

Contributions

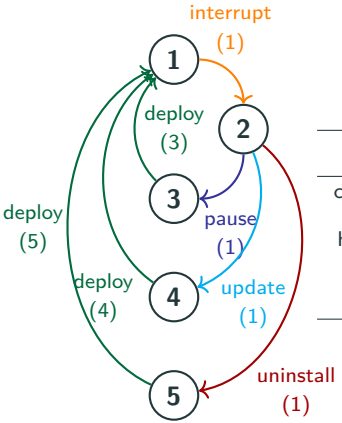
- Ballet as a DevOps reconfiguration tool
- Code and benchmark: `https://zenodo.org/records/10472116`
- Infer reconfiguration actions
- Efficient execution of actions

Target applications

- Multi-site OpenStack
- CPS with sensors
- Management of Fog area

Backup

CP Model



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

Figure 6: Automaton representation of *Mariadb_master* component's life cycle with its matrix for ports statuses.

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

where

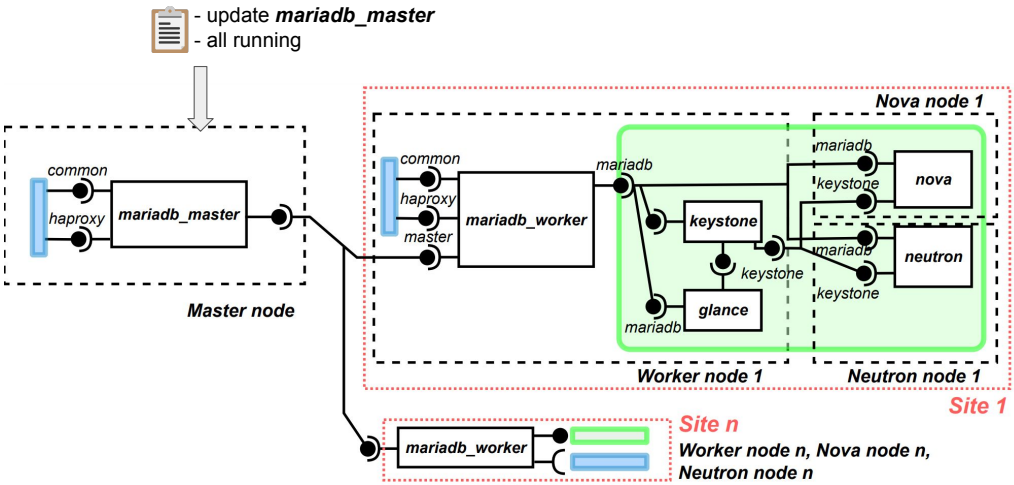
Π an automaton with $\mathcal{C}$ costs

B a sequence of m behaviors

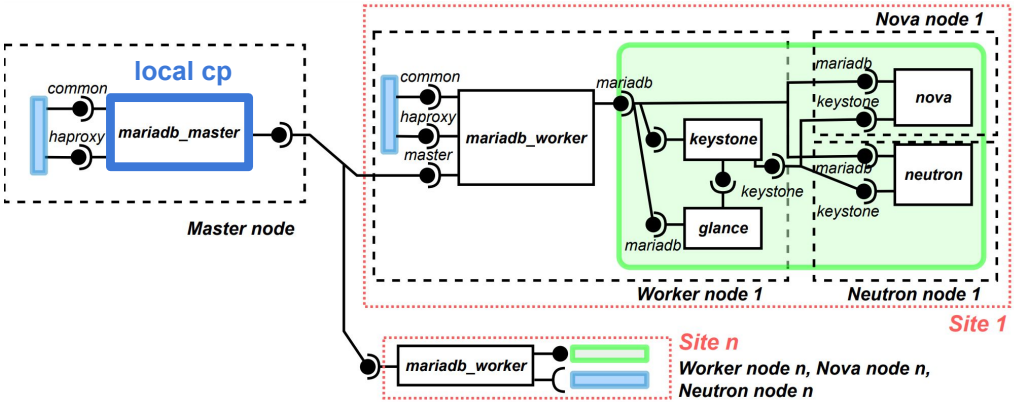
$\Gamma_p \in \{\texttt{active}, \texttt{inactive}\}$i.e. { ✓, × }

$b \in$ { interrupt, deploy, pause, update, uninstall}

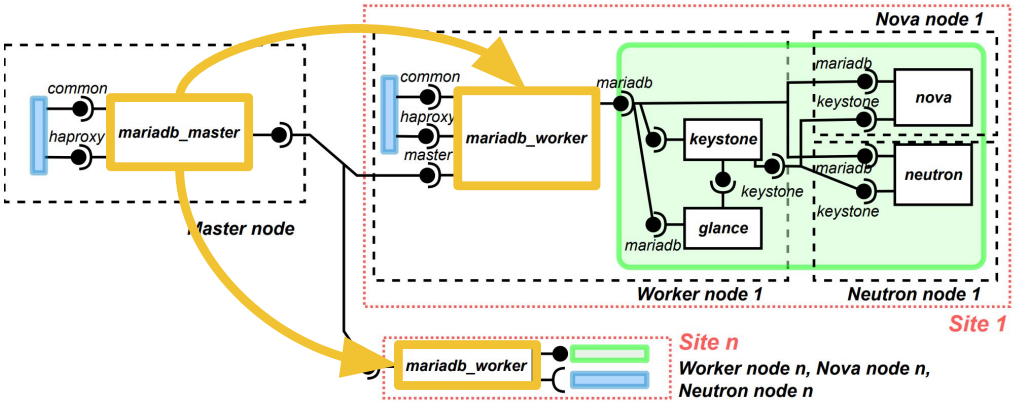
Communication protocol



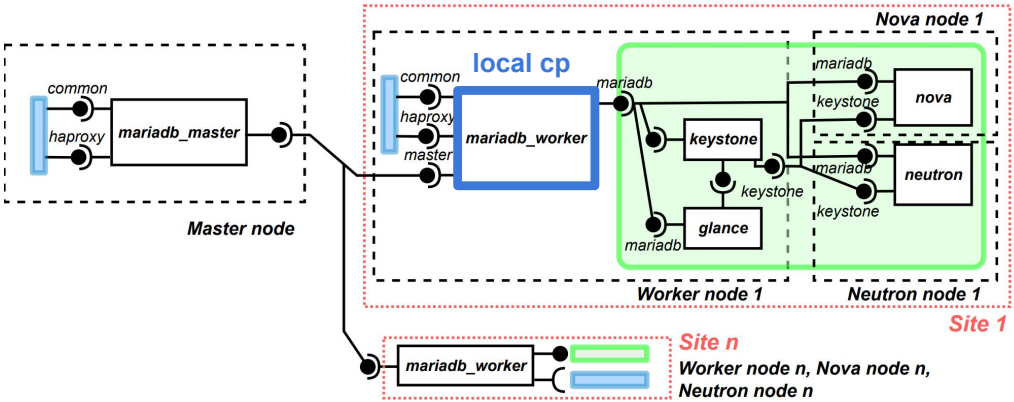
Communication protocol



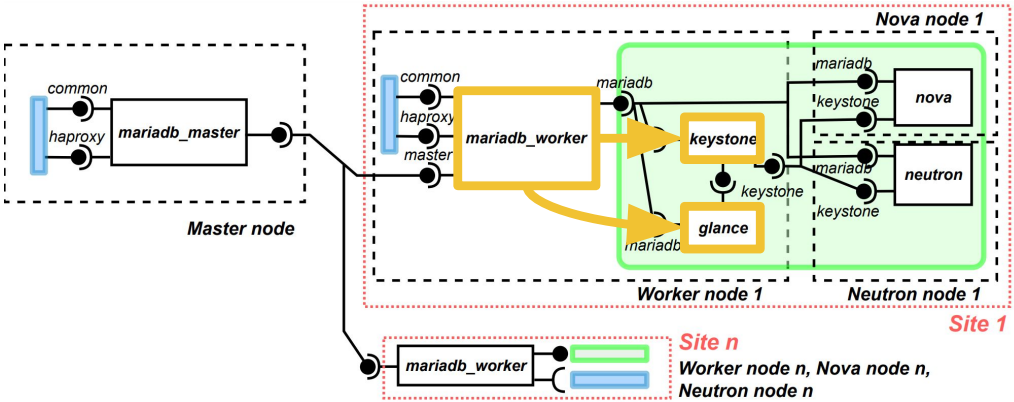
Communication protocol



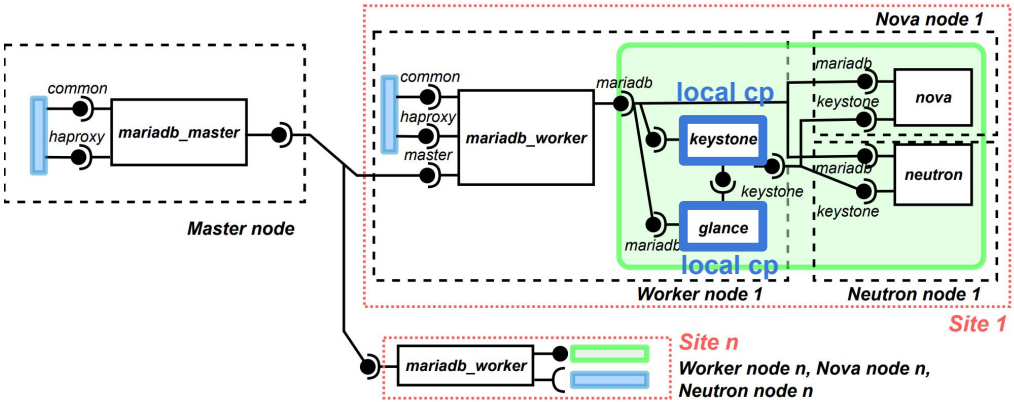
Communication protocol



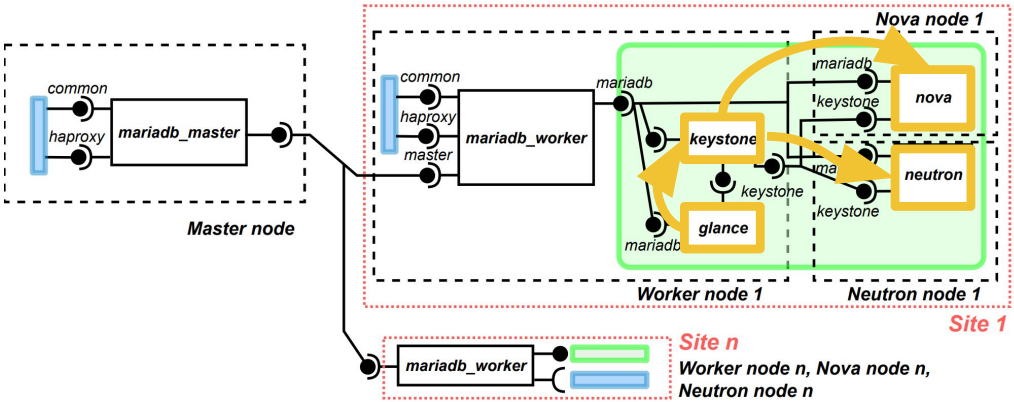
Communication protocol



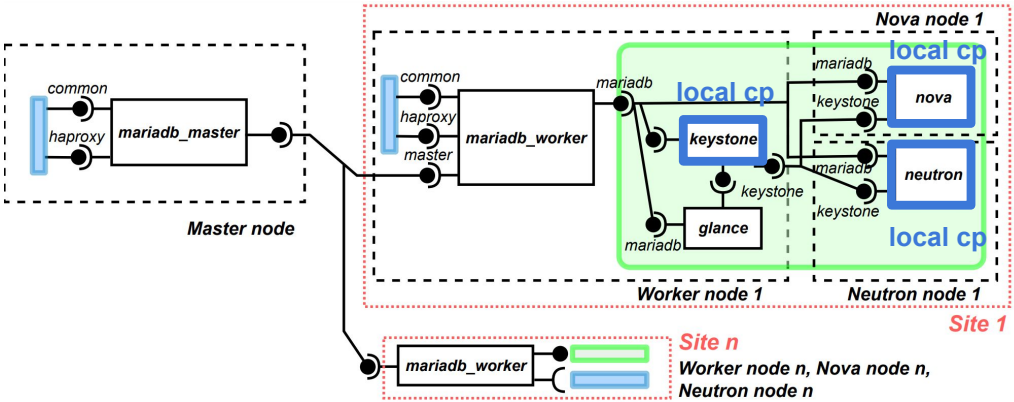
Communication protocol



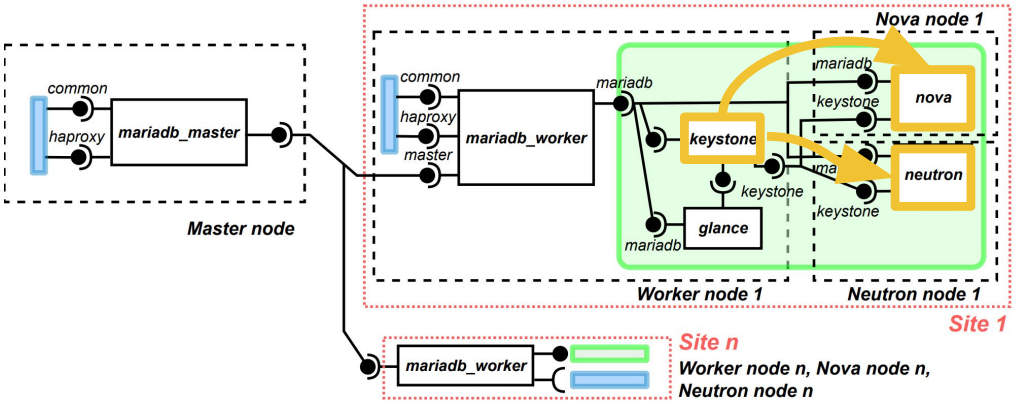
Communication protocol



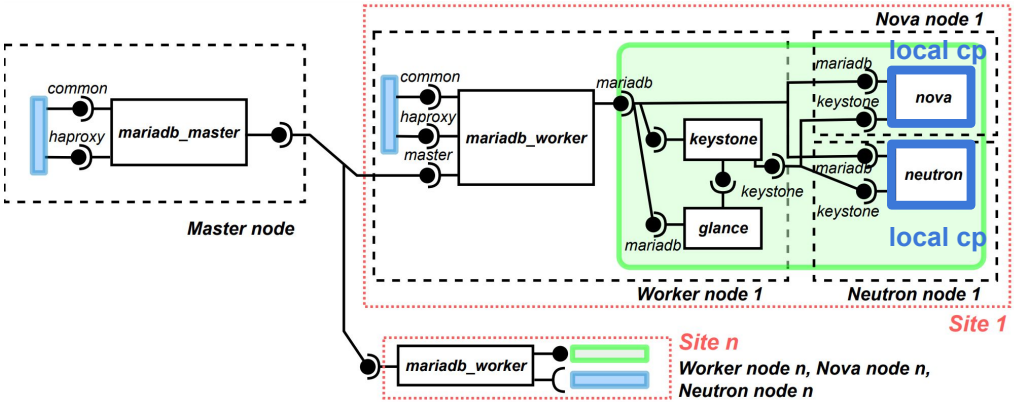
Communication protocol



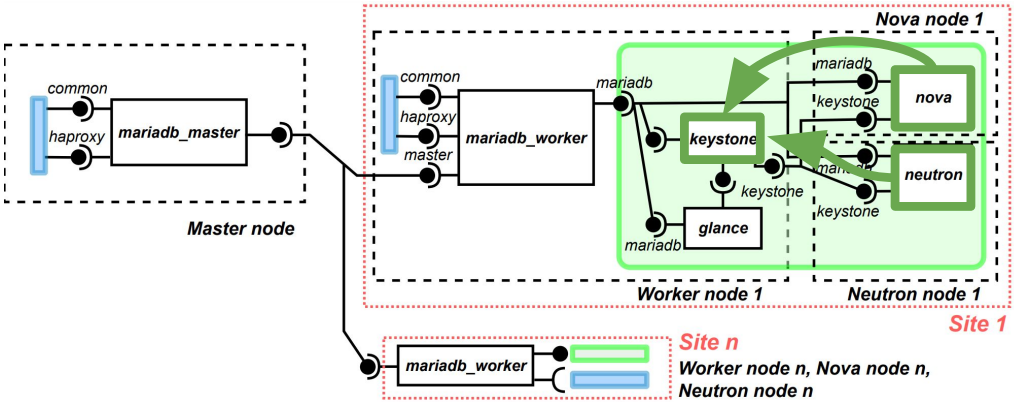
Communication protocol



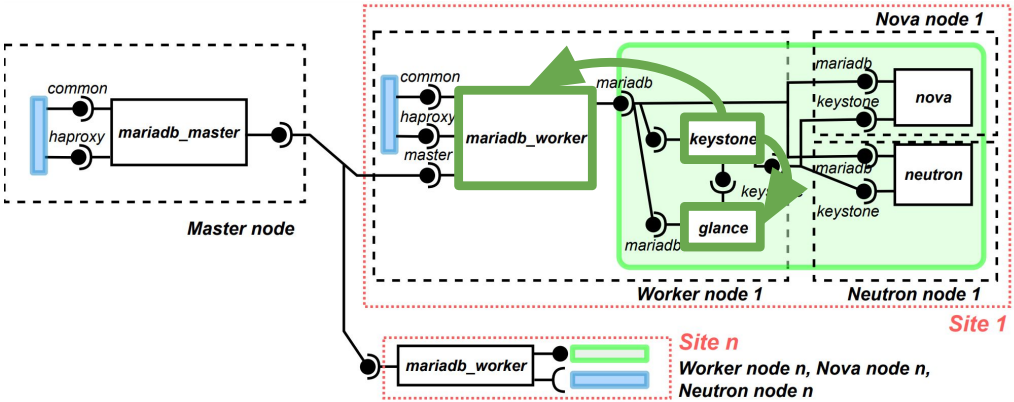
Communication protocol



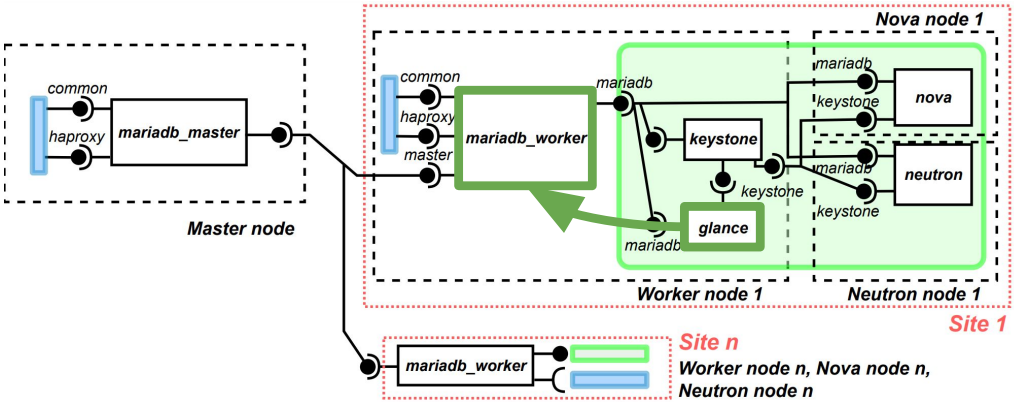
Communication protocol



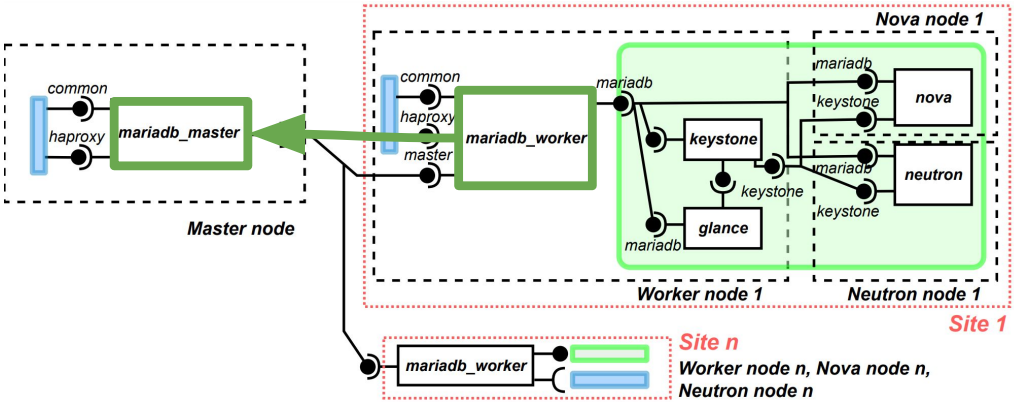
Communication protocol



Communication protocol



Communication protocol



Communication protocol