Managing Conflicts in Cross-DevOps Declarative Reconfigurations

Jolan Philippe, Charles Prud'Homme, Hélène Coullon

March 24, 2025

DiverSE team - IRISA

Hello world

Jolan Philippe, Postdoc in the Taranis project, WP2
Université de Rennes, DiverSE Team
(λx.λy.x@y) Jolan.Philippe inria.fr

Topics of interest

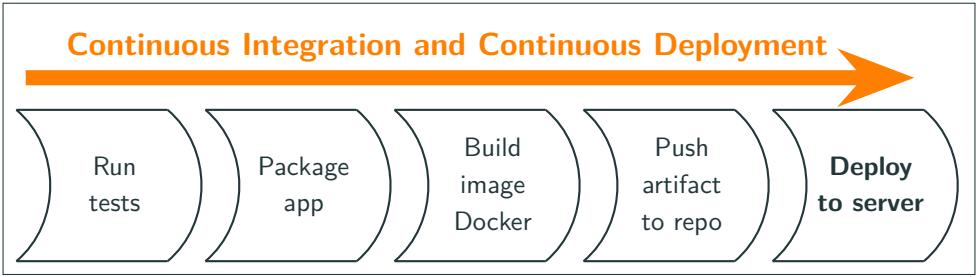
- Distributed computing
- Infrastructure-as-Code
- Decentralized reconfiguration

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

DevOps

An emerging philosophy

- **Development Side** writes code with frequent updates, introducing potential errors;
- **Operations side** uses the application and requires it to function properly;
- **DevOps** bridge the gap between Development and Operations, with tools and practices

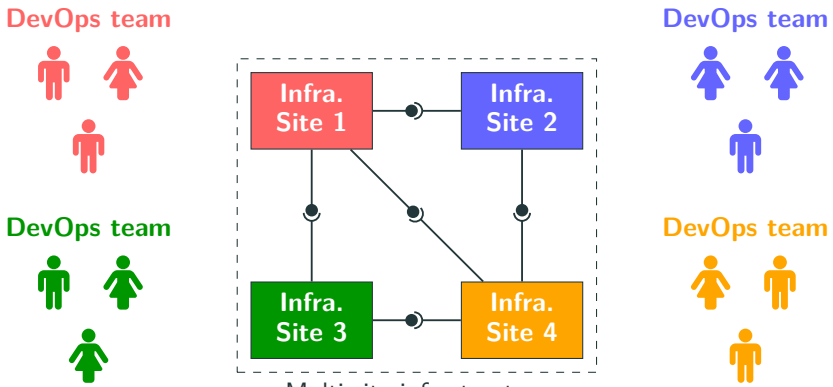


Continuous Deployment

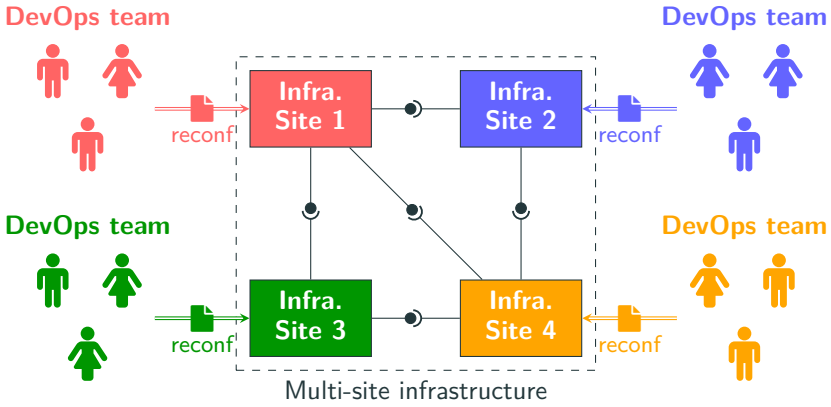
Declarative Reconfiguration

- **Specify the desired state**: Define the target configuration of resources
- **Automated reconfiguration**: A declarative engine plans and executes the reconfiguration
- **Adopted in IaC** (Infrastructure as Code) for provisioning or configuration management

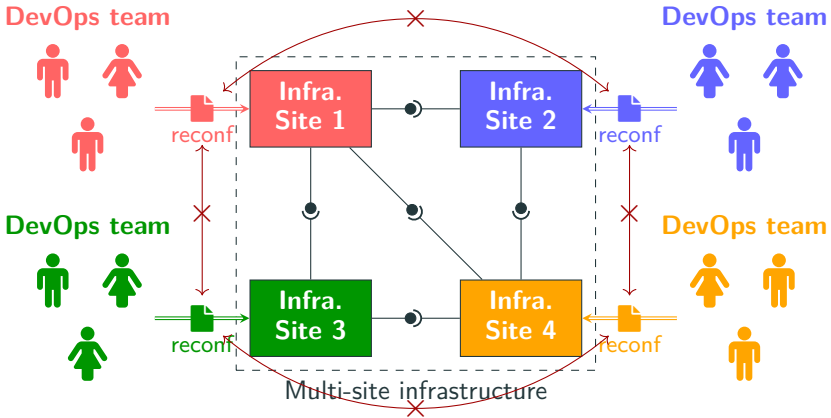
Cross-DevOps teams for multi-site infrastructure



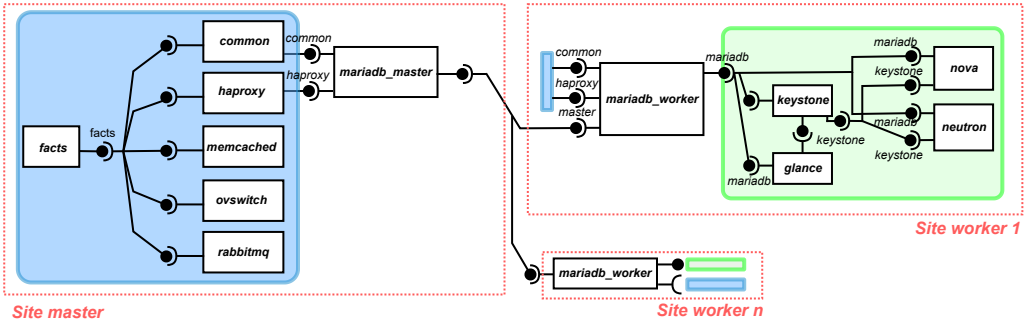
Cross-DevOps teams for multi-site infrastructure



Cross-DevOps teams for multi-site infrastructure



Running example: Reconfiguration with conflicts of multi-site OpenStack



Hypothesis

- Strong version dependency between components
- Components have versions v_1, v_2, v_3
- Components are all in v_1

Reconfiguration

- Update Site master's ***common*** from v_1 to v_2
- Update Site worker 1's ***nova*** from v_1 to v_3

Challenge and approach

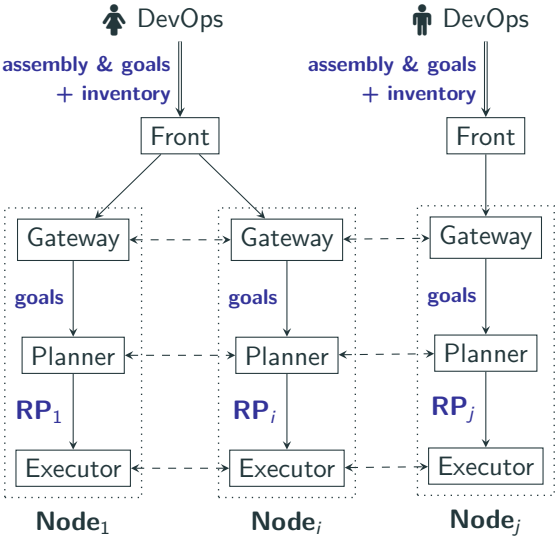
Challenges

- How to detect and explain conflicting reconfiguration objectives?
- How to manage multi-objective reconfiguration conflicts, in a decentralized environment, without any oracle?

Approach

- Extended decentralized reconfiguration engine BALLET into BALLET$^+$
- Local planning with conflict management
- Backtracking of conflict causes to DevOps

Ballet for decentralized reconfiguration



Planner

Decentralized inference of reconfiguration plans (RPs)

1. Iterative process for local actions
   - Local inference of behaviors
   - Constraint diffusion with message propagation (Gossip)
   - When propagation ends, inference of RPs

Executor [A. Omond's thesis]

Coordinated execution of RPs

Ballet's usage: For Developers

Life-cycle and dependencies

Simple language (DSL in Python) 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

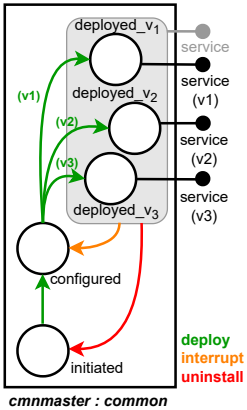


Figure 1: Visual representation of a versionned component for common

Ballet's usage: For DevOps

Target assembly (YAML)

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

Reconfiguration goals

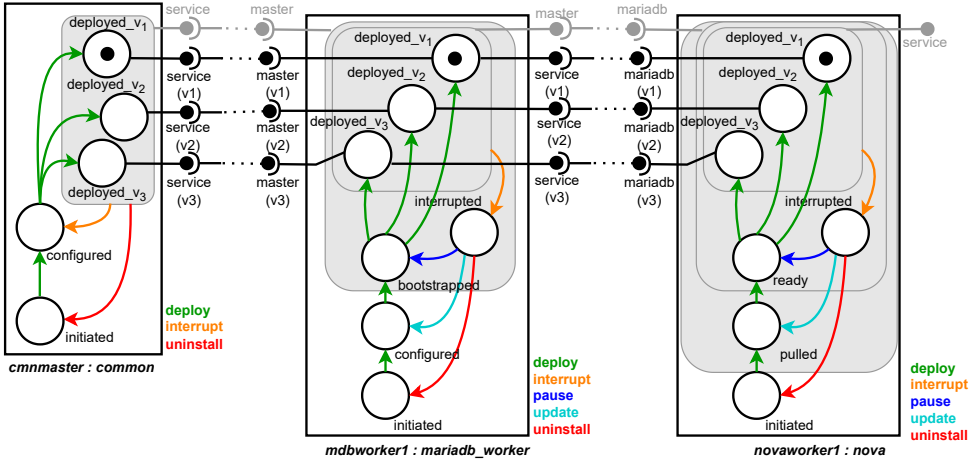
Declarative language for defining reconfiguration goals

- **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)

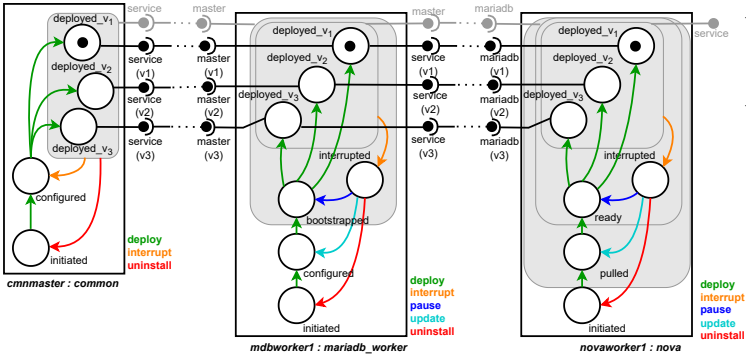
Listing 1: Language to define reconfiguration goals for DevOps usage

```
<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>
```

Running example: Assembly



Running example: Goals




```
goals:
- components:
  - component: cmnmaster
    status: deployed_v2
- ports:
  - forall
    port: service
    status: active
```

Planning Concerto-D programs

Decentralized planner

- **Input**: goals and lifecycle
- **Output**: a reconfiguration plan

Iterative process

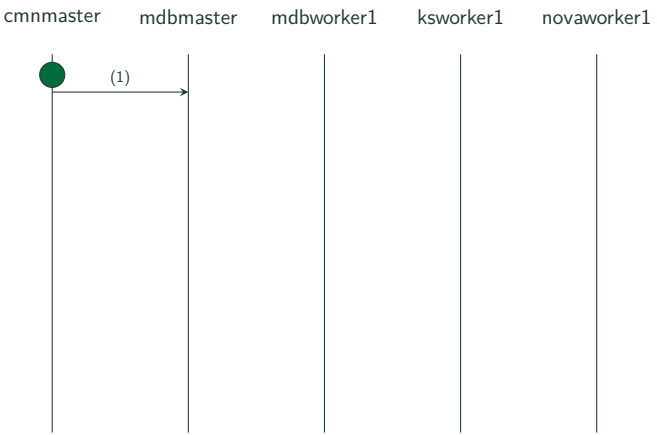
- **Local solving**: Model the components' lifecycles as automaton, and using constraint programming, find a word (i.e., sequence of behaviors) in this automaton meetings with reconfiguration goals
- **Message diffusion**:
  - **OUT** From the word, calculate status of ports and infer potential sync needs. Messages with ports statuses are diffused.
  - **IN** Received messages are translated into additional constraints to enrich local model

Constraint diffusion - without conflicts

local solve (SAT)

message := (source, port, status, final)

Constraint diffusion - without conflicts

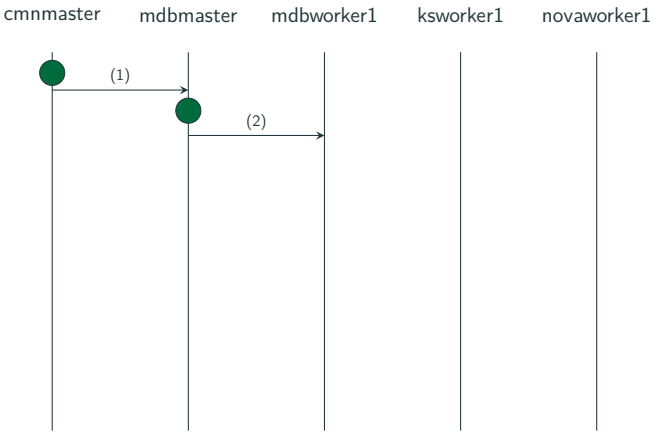


local solve (SAT)

message := (source, port, status, final)

(1) (cmnmaster, service(v2), enabled, True)

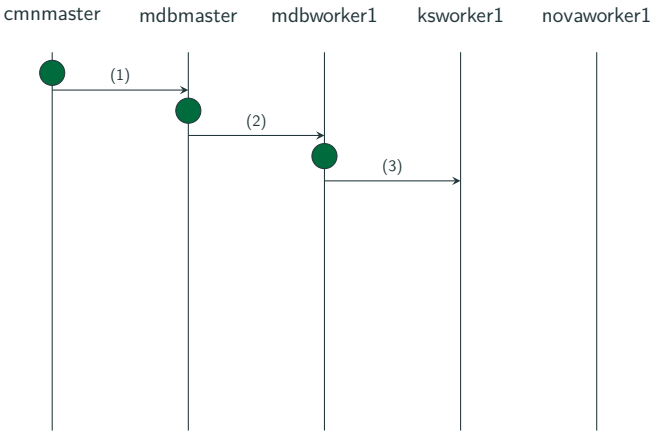
Constraint diffusion - without conflicts



local solve (SAT)

message := (source, port, status, final)
(1) (cmnmaster, service(v2), enabled, True)
(2) (mdbmaster, service(v2), enabled, True)

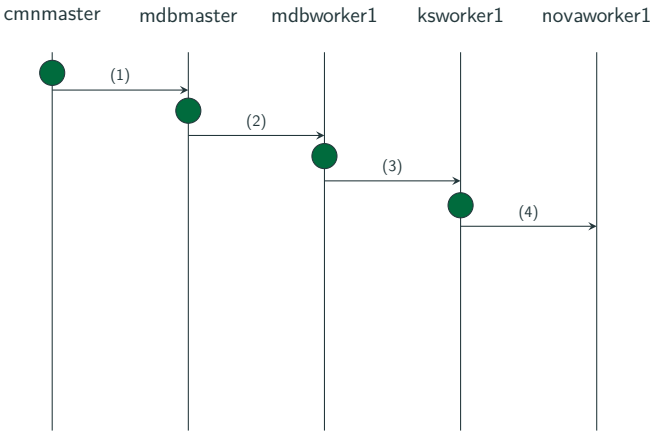
Constraint diffusion - without conflicts



local solve (SAT)

message := (source, port, status, final)
(1) (cmnmaster, service(v2), enabled, True)
(2) (mdbmaster, service(v2), enabled, True)
(3) (mdbworker1, service(v2), enabled, True)

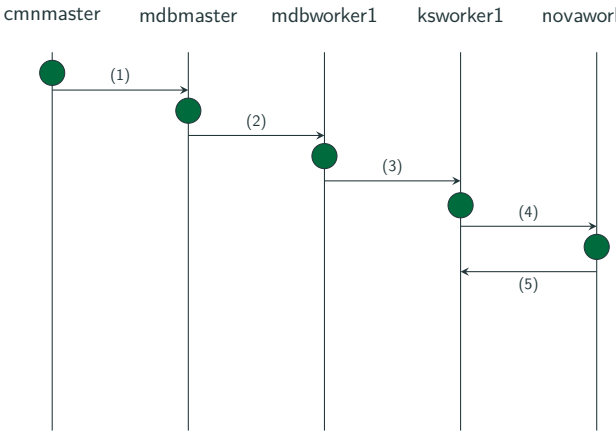
Constraint diffusion - without conflicts



local solve (SAT)

message := (source, port, status, final)
(1) (cmnmaster, service(v2), enabled, True)
(2) (mdbmaster, service(v2), enabled, True)
(3) (mdbworker1, service(v2), enabled, True)
(4) (ksworker, service(v2), enabled, True)

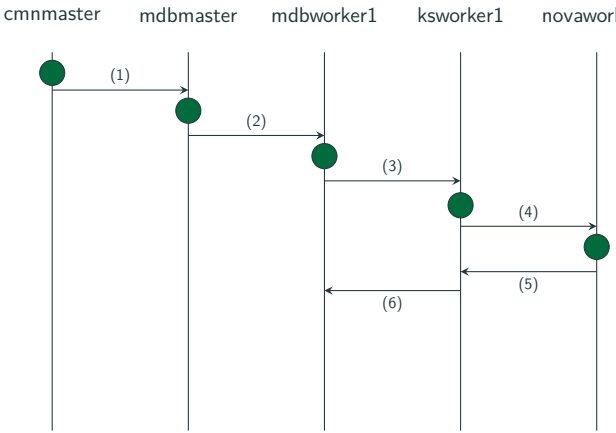
Constraint diffusion - without conflicts



local solve (SAT)

message := (source, port, status, final)
(1) (cmnmaster, service(v2), enabled, True)
(2) (mdbmaster, service(v2), enabled, True)
(3) (mdbworker1, service(v2), enabled, True)
(4) (ksworker, service(v2), enabled, True)
(5) ACK (4)

Constraint diffusion - without conflicts



local solve (SAT)

message := (source, port, status, final)

(1) (cnmmaster, service(v2), enabled, True)

(2) (mdbmaster, service(v2), enabled, True)

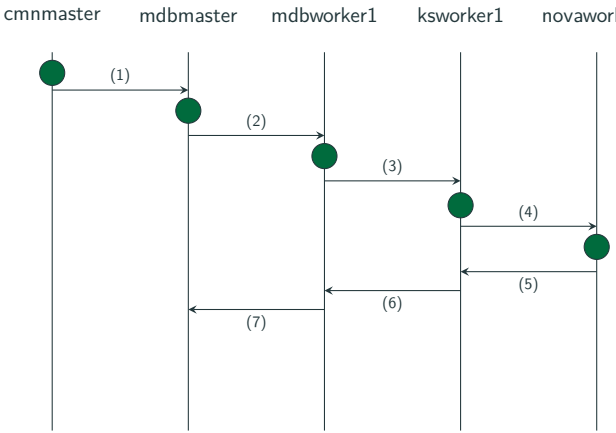
(3) (mdbworker1, service(v2), enabled, True)

(4) (ksworker, service(v2), enabled, True)

(5) ACK (4)

(6) ACK (3)

Constraint diffusion - without conflicts

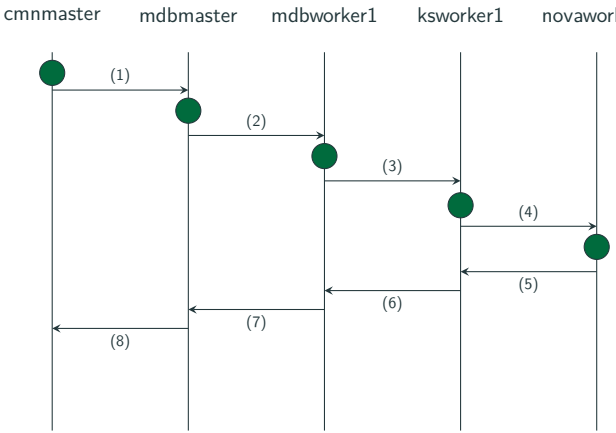


local solve (SAT)

message := (source, port, status, final)

(1) (cmnmaster, service(v2), enabled, True)
(2) (mdbmaster, service(v2), enabled, True)
(3) (mdbworker1, service(v2), enabled, True)
(4) (ksworker, service(v2), enabled, True)
(5) ACK (4)
(6) ACK (3)
(7) ACK (2)

Constraint diffusion - without conflicts

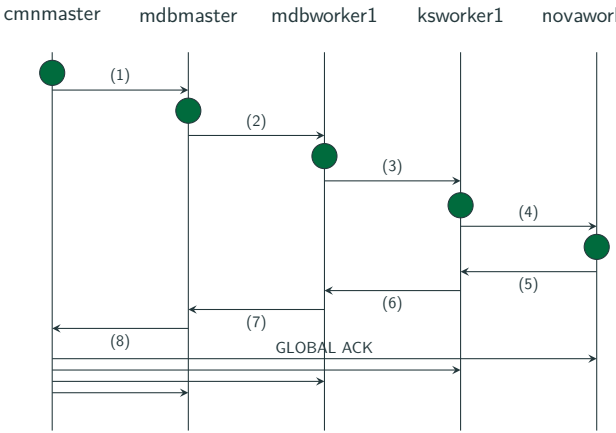


local solve (SAT)

message := (source, port, status, final)

(1) (cmnmaster, service(v2), enabled, True)
(2) (mdbmaster, service(v2), enabled, True)
(3) (mdbworker1, service(v2), enabled, True)
(4) (ksworker, service(v2), enabled, True)
(5) ACK (4)
(6) ACK (3)
(7) ACK (2)
(8) ACK (1)

Constraint diffusion - without conflicts



local solve (SAT)

message := (source, port, status, final)

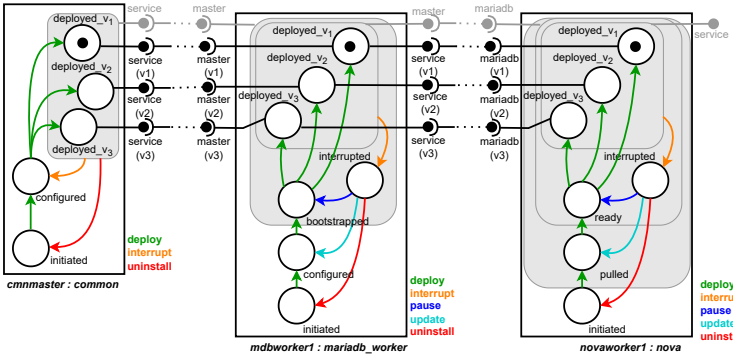
(1) (cmnmaster, service(v2), enabled, True)
(2) (mdbmaster, service(v2), enabled, True)
(3) (mdbworker1, service(v2), enabled, True)
(4) (ksworker, service(v2), enabled, True)
(5) ACK (4)
(6) ACK (3)
(7) ACK (2)
(8) ACK (1)

Unsatisfiable models management in Ballet^{+}

Approach

- We first attempt solving the model. Two possible cases
  - SAT: A word is found, the process is like BALLET
  - UNSAT: We find the minimal set of unsatisfiable constraints using **QuickXplain** algorithm
- In case of UNSAT, we backtrack the messages responsible of the local failure to the DevOps, building an incidence tree

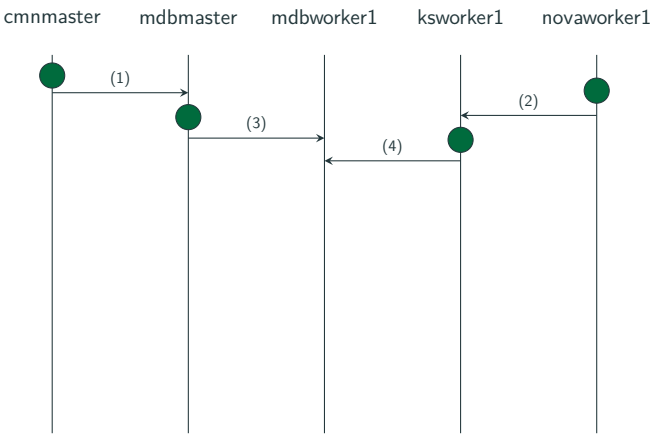
Unsatisfiable local models with Ballet




```
goals:
- components:
  - component: cmnmaster
    status: deployed_v2
- ports:
  - forall
    port: service
    status: active
```

```
goals:
- components:
  - component: novaworker1
    status: deployed_v3
- ports:
  - forall
    port: service
    status: active
```

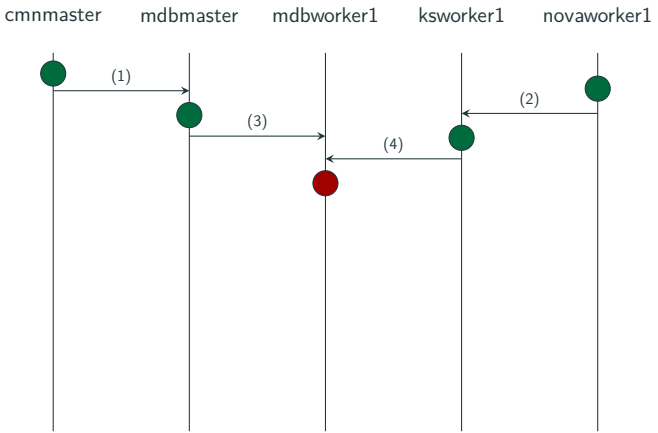
Constraint diffusion - with conflicts



local solve (SAT)

message := (source, port, status, final)
(1) (cmnmaster, service(v2), enabled, True)
(2) (novaworker1, service(v3), enabled, True)
(3) (mdbmaster, service(v2), enabled, True)
(4) (ksworker, service(v3), enabled, True)

Constraint diffusion - with conflicts



local solve (SAT)

local solve (UNSAT)

message := (source, port, status, final)

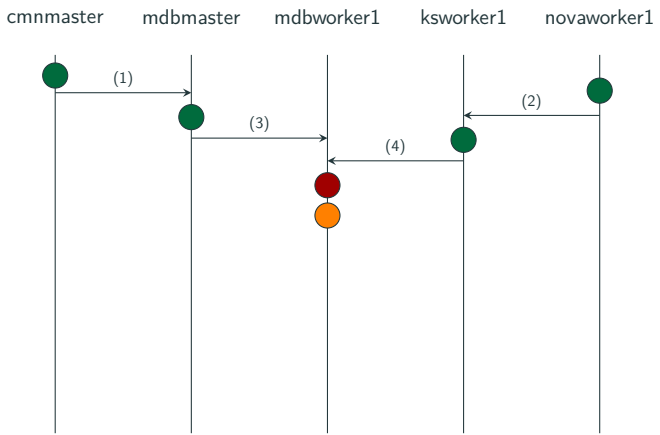
(1) (cmnmaster, service(v2), enabled, True)

(2) (novaworker1, service(v3), enabled, True)

(3) (mdbmaster, service(v2), enabled, True)

(4) (ksworker, service(v3), enabled, True)

Constraint diffusion - with conflicts



local solve (SAT) QuickXplain

local solve (UNSAT)

message := (source, port, status, final)

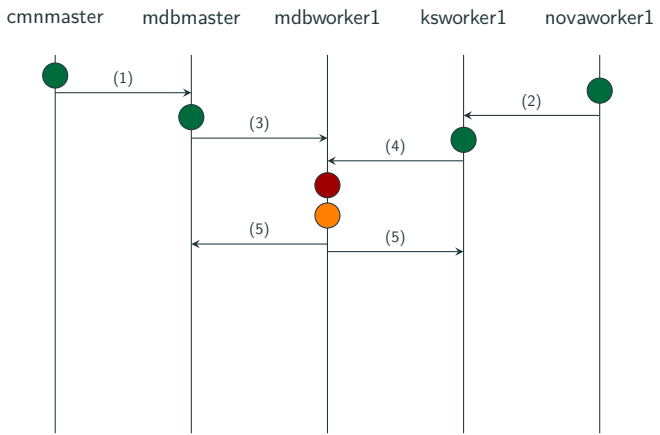
(1) (cmnmaster, service(v2), enabled, True)

(2) (novaworker1, service(v3), enabled, True)

(3) (mdbmaster, service(v2), enabled, True)

(4) (ksworker, service(v3), enabled, True)

Constraint diffusion - with conflicts



local solve (SAT) QuickXplain

local solve (UNSAT)

message := (source, port, status, final)

(1) (cmnmaster, service(v2), enabled, True)

(2) (novaworker1, service(v3), enabled, True)

(3) (mdbmaster, service(v2), enabled, True)

(4) (ksworker, service(v3), enabled, True)

(5) REFUSE caused by (3) and (4)

Constraint diffusion - with conflicts

local solve (SAT) QuickXplain

local solve (UNSAT)

message := (source, port, status, final)

(1) (cmnmaster, service(v2), enabled, True)

(2) (novaworker1, service(v3), enabled, True)

(3) (mdbmaster, service(v2), enabled, True)

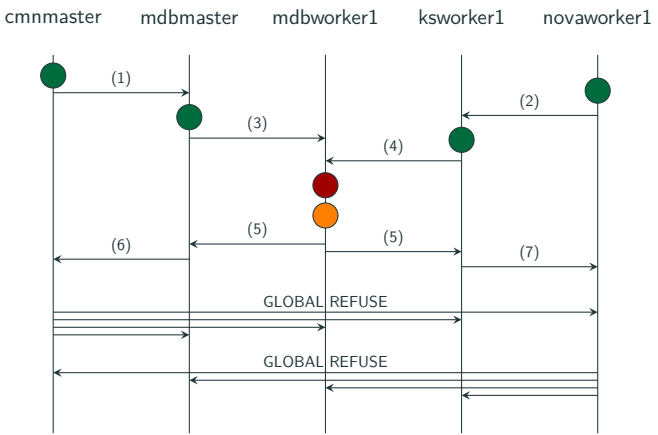
(4) (ksworker, service(v3), enabled, True)

(5) REFUSE caused by (3) and (4)

(6) REFUSE caused by (3), (4) and (1)

(7) REFUSE caused by (3), (4) and (2)

Constraint diffusion - with conflicts



local solve (SAT) QuickXplain

local solve (UNSAT)

message := (source, port, status, final)

(1) (cmnmaster, service(v2), enabled, True)

(2) (novaworker1, service(v3), enabled, True)

(3) (mdbmaster, service(v2), enabled, True)

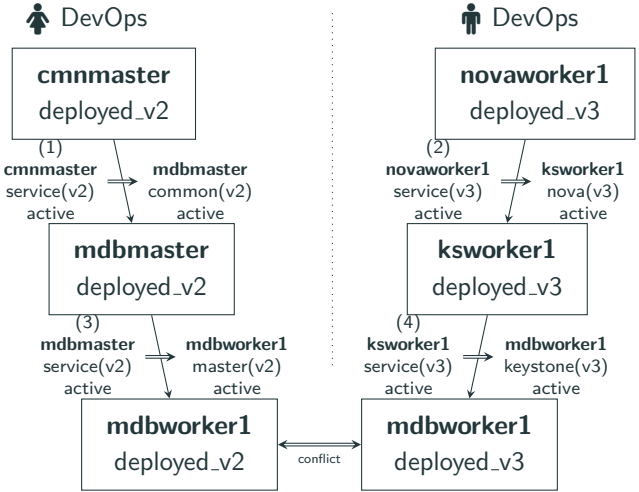
(4) (ksworker, service(v3), enabled, True)

(5) REFUSE caused by (3) and (4)

(6) REFUSE caused by (3), (4) and (1)

(7) REFUSE caused by (3), (4) and (2)

Output to DevOps



Returned tree

- The tree captures constraints and responsible messages
- Iteratively built during backtracking
- DevOps only get their branch, from the conflict to the submitted goals

⇒ Does the UNSAT management introduce an overhead ?

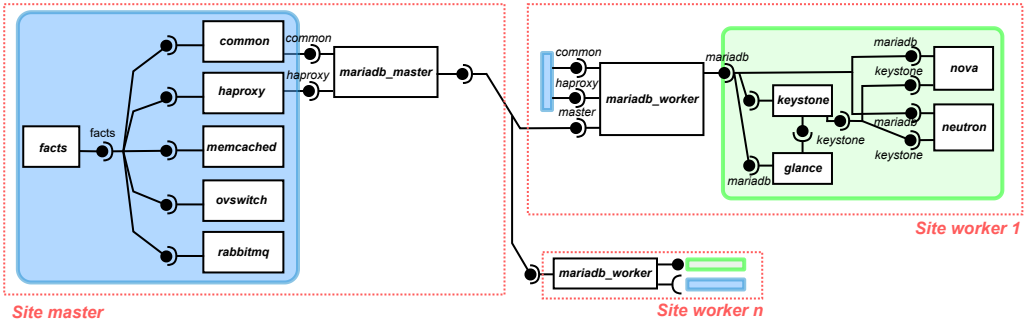
Metrics

- Compare total time of planning phase in a satisfiable scenario (no conflict between goals) and in an unsatisfiable scenario (conflicting goals)
- Compare the maximum solving time (which might include running Qx) in a satisfiable scenario and in an unsatisfiable scenario

Setup

- Experiments run on Grid5000 (`gros` on Nancy site)
- Run cases on multi-site Openstack case and on topological assemblies

Evaluating Ballet$^+$ on real use-case



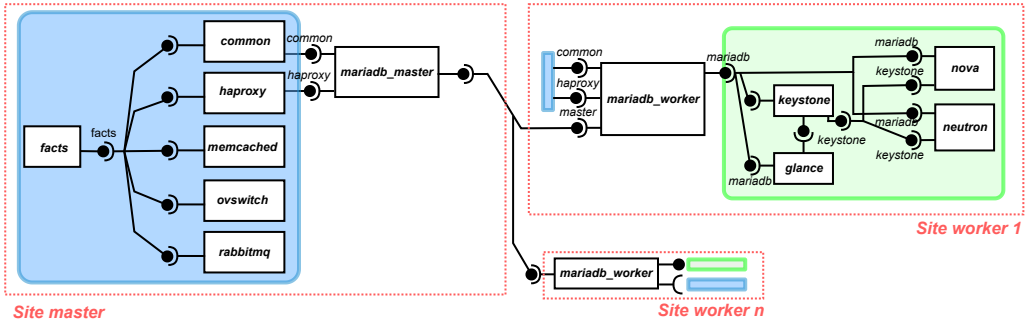
SAT-case goals

- Update Site master's ***common*** from v_1 to v_2
- all components end `running`

UNSAT-case goals

- Update Site master's ***common*** from v_1 to v_2
- Update Site worker 1's ***nova*** from v_1 to v_3
- all components end `running`

Evaluating Ballet$^+$ on real use-case



	SAT-case time (σ)	**UNSAT-case** time (σ)
Planning time	8.4922s (1.29)	10.4161s (1.45)
Solving time	3.85s (0.11)	3.2124s (0.86)

Table 1: Average time (for 10 runs, with standard deviation) of the full planning process, and average time of the maximum local solving times

Evaluating Ballet$^+$ on synthetic examples

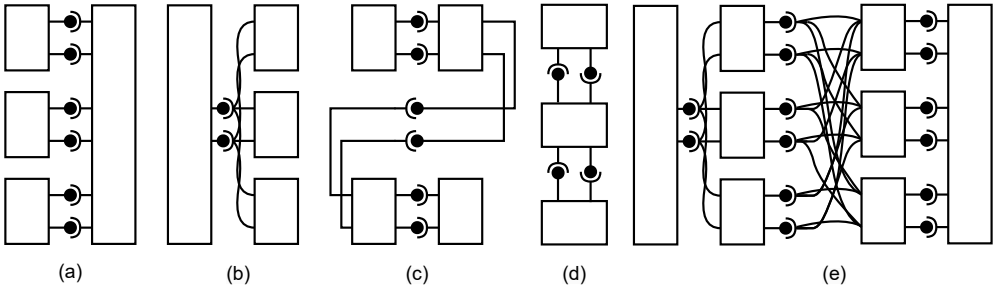


Figure 2: The five assembly topologies used in synthetic examples: (a) central-user (*c-user*), (b) central-provider (*c-provider*), (c) linear (*linear*), (d) circular (*circular*), and (e) stratified (*stratified*).

- One instance of BALLET$^+$ for each component
- Comparing two scenarios: a non-conflicting reconf. (**SAT-case**) & and confiicting one (**UNSAT-case**)

Evaluating Ballet$^+$ on synthetic examples

topology	#components	initial states	**SAT-case** goals	**UNSAT-case** goals
c-user	1 user; 15 providers	user: running provider$_i$: running	user: (state) running provider$_i$: (state) running + (behavior) update	user: (state) running provider$_i$: (state) uninstalled
c-provider	1 provider; 15 users	user$_i$: running provider: running	user$_i$: (state) running provider: (state) running + (behavior) update	user$_i$: (state) running provider: (state) uninstalled
linear	1 provider; 15 transformers	provider: running transformer$_i$: running	provider: (state) running + (behavior) update transformer$_i$: (state) running	provider: uninstalled transformer$_i$: (state) running
circular	1 provider; 15 transformers; 1 user	provider: running transformer$_i$: running user: running	provider: (state) running + (behavior) update transformer$_i$: (state) running user: (state) running	provider: uninstalled transformer$_i$: (state) running user: (state) running
stratified	1 provider 15 mid-users 1 user	provider: running mid-user$_i$: running user: running	provider: (state) running + (behavior) update mid-user$_i$: (state) running user: (state) running	provider: uninstalled mid-user$_i$: (state) running end-user: (state) running

Table 2: Testing scenarios for assembly topologies with goals for a satisfiable case (SAT-case), and an unsatisfiable case (UNSAT-case)

Evaluating Ballet$^+$ on synthetic examples

topology	**SAT**-case		**UNSAT**-case	
	total time (σ)	**max solving time** (σ)	**total time** (σ)	**max solving time** (σ)
c-user	23.362s (3.79)	4.1970s (1.40)	19.8341s (0.91)	1.3841s (0.23)
c-provider	2.0057s (0.30)	11.1907s (1.15)	9.7482s (0.91)	0.7192s (0.17)
linear	0.7192s (0.12)	34.9178s (2.40)	0.6753s (0.11)	29.0932s (1.64)
circular	1.0587s (0.63)	14.0585s (0.52)	0.4983s (0.10)	1.9996s (0.61)
stratified	1.8807s (0.25)	19.9338s (1.19)	0.7918s (0.14)	24.0926s (1.32)

Table 3: Average times (for 10 runs, with standard deviation) for the full planningtime alongside the maximum local solving time for one component, in the SAT and UNSAT cases for each scenario on topologies.

Concluding remaks

Contribution

- We propose a solution for managing conflicting goals in cross-DevOps declarative reconfigurations
- The proposed solution does not introduce a time overhead
- The presented work has been submitted to ICSME 2025 (Software Maintenance and Evolution)

Perspectives

- Propose reconfiguration options to satisfy a maximum of goals
- Improve BALLET$^+$'s usage of constraint solvers to manage larger models