IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Inria
INVENTEURS DU MONDE NUMÉRIQUE

LS2N
LABORATOIRE DES SCIENCES DU NUMÉRIQUE DE NANTES

NORGES ARKTISKE UNIVERSITET
UIT

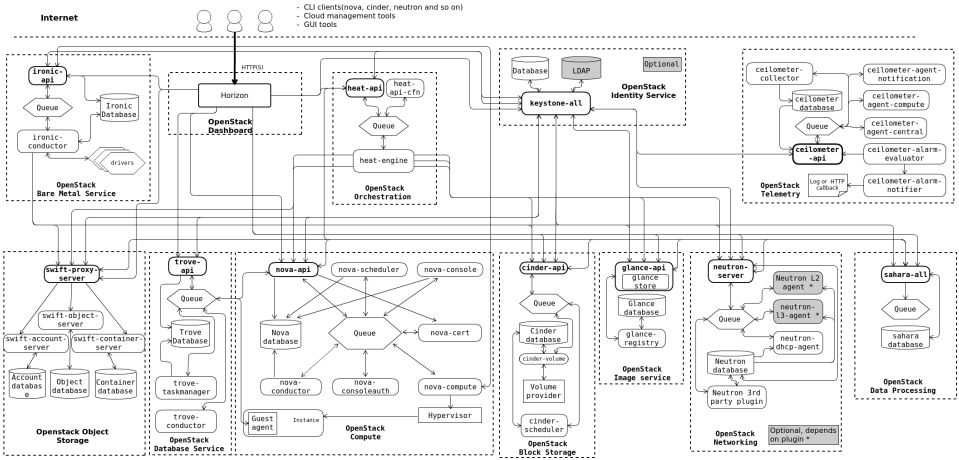
Towards efficient and safe autonomic (re)configuration

Hélène Coullon
Associate professor IMT Atlantique, Inria chair, UiT

Introduction

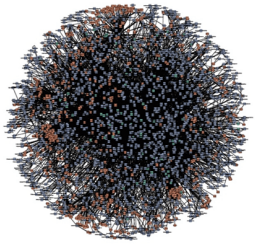
Complex distributed software systems

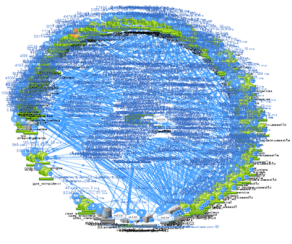
OpenStack (infrastructure management) - up to 250 modules



Complex distributed software systems

Micro-services architectures of Netflix and Amazon - thousands of modules





Context

Distributed software systems

- Ever-lived and long-lived module-based systems
- High number of modules
- Complexity of dependencies between modules

Examples of management operations at runtime (deployment/reconfiguration)

- Unavailable services (faults, errors)
- Need to add/remove modules and/or connections
- Change of internal configurations
- Update of some modules

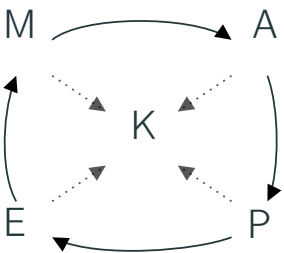
Autonomic reconfiguration

Objective

- Autonomic distributed systems
  - *"Computing systems that can manage themselves given high-level objectives from administrators" [1]*
  - Autonomic reconfiguration

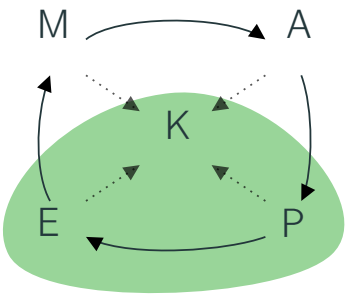
MAPE-K autonomic loop [1]

- (M)onitoring
- Decisions: (A)nalysis, (P)lanning
- (E)xecution
- (K)nowledge



[1] The vision of autonomic computing. J. O. Kephart and D. M. Chess. In Computer, 2003.

VeRDi project - Generic autonomic reconfiguration

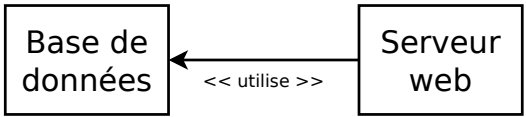


VeRDi project

- Led by *Hélène Coullon*
- Verified & efficient generic reconfiguration
- Co-supervision of *Maverick Chardet* (PhD)
  - funded by the IPL Discovery of Adrien Lebre
  - *Christian Perez*
  - *[defended the 2020-12-03]*
- Supervision of two postdocs and one engineer
  - *Dimitri Pertin*, postdoc
  - *Simon Robillard*, postdoc *[ongoing]*
  - *Charlène Servantie*, engineer

Motivation and state of the art

(Re)configuration execution = Coordination



Database (DB)

1. Install MySQL package + deps
2. Configure parameters
3. Start the service
4. Add a user
5. Create some tables

Web-server (WS)

1. Install Apache package + deps
2. Configure the firewall
3. Download the website content
4. Configure parameters
5. Start the service

Dependencies: WS(4) $\rightarrow$ DB(3), WS(5) $\rightarrow$ DB(5)

Concerto: goals and philosophy

A model for reconfigurations in component-based systems

- represent the lifecycle of components
  - non-functional aspect
  - in this talk, component = *control* component
- coordinate reconfiguration actions
  - e.g. starting/stopping VM, downloading images, installing/updating software...

Concerto: goals and philosophy

A model for reconfigurations in component-based systems

- represent the lifecycle of components
  - non-functional aspect
  - in this talk, component = *control* component
- coordinate reconfiguration actions
  - e.g. starting/stopping VM, downloading images, installing/updating software...

Performance

- structured parallelism
- reach quickly a configuration
- avoid disruption time

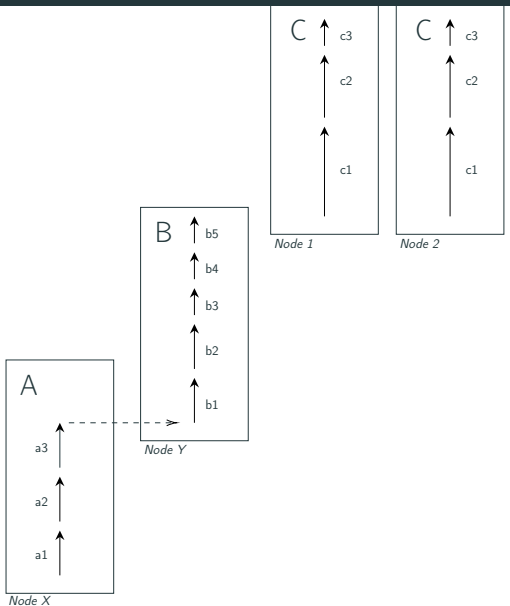
Safety

- formally-defined semantics
- tools to assist during design
- verification of properties

Performance through parallelism

level 1: multiple nodes, same action

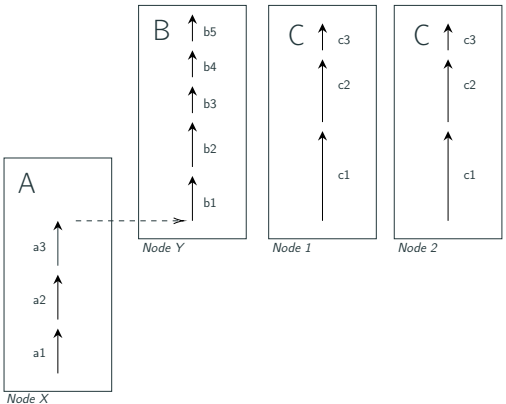
- no dependencies declared
- procedural execution order
- *[Ansible]*



Performance through parallelism

level 2: non-dependent components

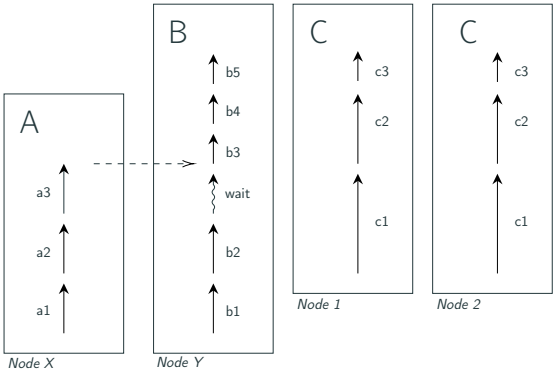
- dependencies at the component level
- *[Deployware, Tosca]*



Performance through parallelism

level 3: inter-component

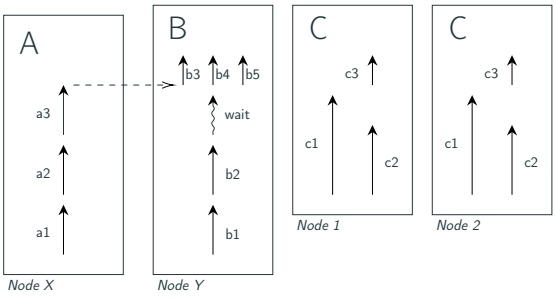
- dependencies at the task level
- *[Aeolus]*



Performance through parallelism

level 4: intra-component

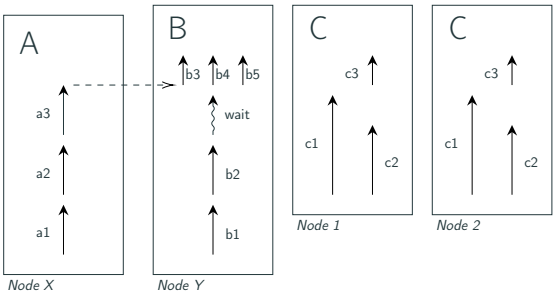
- internal task dependencies
- *[Concerto]*



Performance through parallelism

level 4: intra-component

- internal task dependencies
- *[Concerto]*



Parallel execution **requires** precise description of dependencies

State of the art

In the rest of the talk comparison with

Very popular production tool

- *[Ansible]* `https://www.ansible.com/`.

Closest contribution in the literature

- *[Aeolus] Aeolus: a component model for the Cloud. Di Cosmo, Roberto and Mauro, Jacopo and Zacchiroli, Stefano and Zavattaro, Gianluigi. In Information and Computation, 2014.*

Concerto

Contributions

[3] Predictable Efficiency for Reconfiguration of Service-Oriented Systems with Concerto. Maverick Chardet, Hélène Coullon, Christian Perez. In CCGrid 2020.

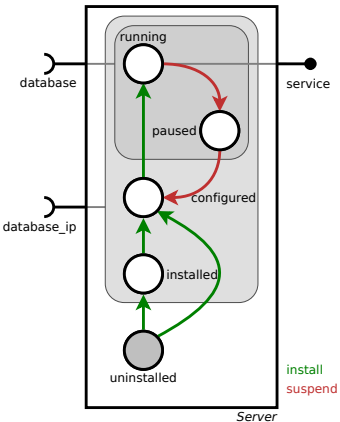
[4] Toward Safe and Efficient Reconfiguration with Concerto. Maverick Chardet, Hélène Coullon, Simon Robillard. In journal SCP, 2020.

[5] Enhancing Separation of Concerns, Parallelism, and Formalism in Distributed Software Deployment with Madeus. Maverick Chardet, Hélène Coullon, Christian Perez, Dimitri Pertin, Charlène Servantie, Simon Robillard. In journal JSS. [minor revision]

[6] Integrated Model-checking for the Design of Safe and Efficient Distributed Software Commissioning. Hélène Coullon, Didier Lime, Claude Jard. In iFM 2019, Bergen, Norway.

Concerto - (1/2) control components

Written by the component developers

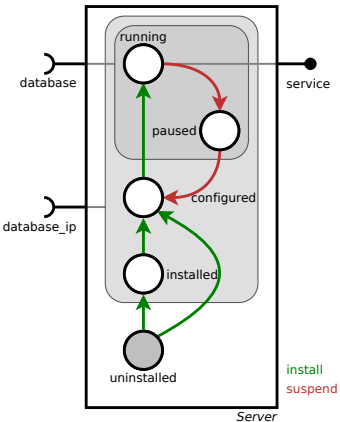


Internal net

- places = milestones
- transitions = actions to perform
  - concretely: scripts are attached to transitions
  - in the model: exact nature/effects of actions not represented, only coordination

Concerto - (1/2) control components

Written by the component developers



Interfaces

- data or service ports
  - use ports = requirements
  - provide ports = provisions
  - during execution: active/inactive
- behaviors
  - subset of transitions
  - during execution: active/inactive

Control components in practice

Written by the component developers

```
class Server(Component):
    def create(self):
        self.places = ['uninstalled','installed','configured','running','paused']

        self.initial_place = 'uninstalled'

        self.behaviors = ['b_install','b_suspend']

        self.transitions = {
            'install1': ('uninstalled','installed','b_install',self.install1),
            'install2': ('uninstalled','configured','b_install',self.install2),
            'configure': ('installed','configured','b_install',self.configure),
            'start': ('configured','running','b_install',self.start),
            'suspend1': ('running','paused','b_suspend',self.suspend1),
            'suspend2': ('paused','configured','b_suspend', self.suspend2)
        }
```

Control components in practice

Written by the component developers

```
class Server(Component):
    def create(self):
        ...

        self.dependencies = {
            'database_ip': (DepType.USE, ['installed','configured','running','paused']),
            'database': (DepType.USE, ['running','paused']),
            'service': (DepType.PROVIDE, ['running'])
        }

    # Definition of the actions
    def install1(self):
        remote = SSHClient()
        remote.connect(host, user, pwd)
        remote.exec_command(cmd)
        ...
```

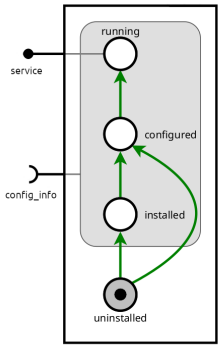
Execution semantics of a control component

"Petri net" style of semantics

- can be in multiple places at once
- transitions not atomic
- can execute multiple transitions at once

Coordination via ports

- use port need to be provided before reaching places
- places cannot be left while provide ports are used



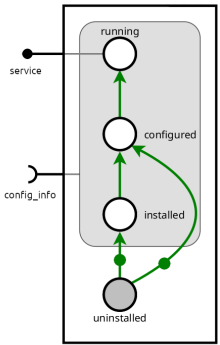
Execution semantics of a control component

"Petri net" style of semantics

- can be in multiple places at once
- transitions not atomic
- can execute multiple transitions at once

Coordination via ports

- use port need to be provided before reaching places
- places cannot be left while provide ports are used



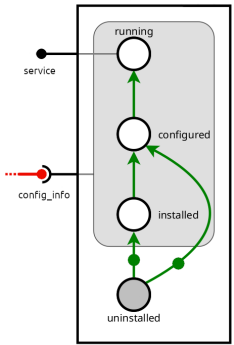
Execution semantics of a control component

"Petri net" style of semantics

- can be in multiple places at once
- transitions not atomic
- can execute multiple transitions at once

Coordination via ports

- use port need to be provided before reaching places
- places cannot be left while provide ports are used



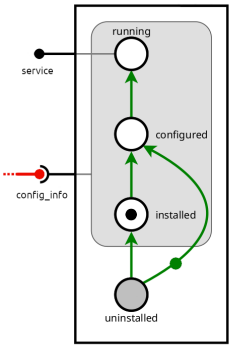
Execution semantics of a control component

"Petri net" style of semantics

- can be in multiple places at once
- transitions not atomic
- can execute multiple transitions at once

Coordination via ports

- use port need to be provided before reaching places
- places cannot be left while provide ports are used



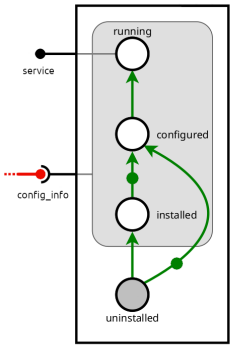
Execution semantics of a control component

"Petri net" style of semantics

- can be in multiple places at once
- transitions not atomic
- can execute multiple transitions at once

Coordination via ports

- use port need to be provided before reaching places
- places cannot be left while provide ports are used



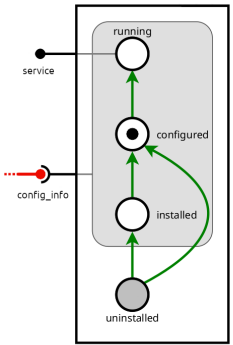
Execution semantics of a control component

"Petri net" style of semantics

- can be in multiple places at once
- transitions not atomic
- can execute multiple transitions at once

Coordination via ports

- use port need to be provided before reaching places
- places cannot be left while provide ports are used



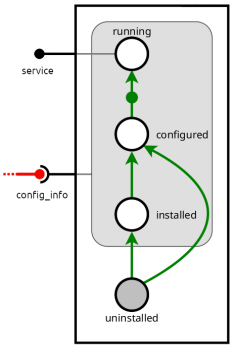
Execution semantics of a control component

"Petri net" style of semantics

- can be in multiple places at once
- transitions not atomic
- can execute multiple transitions at once

Coordination via ports

- use port need to be provided before reaching places
- places cannot be left while provide ports are used



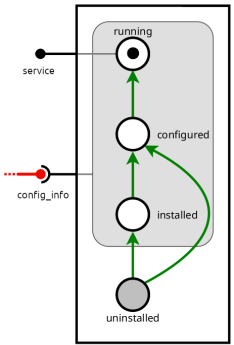
Execution semantics of a control component

"Petri net" style of semantics

- can be in multiple places at once
- transitions not atomic
- can execute multiple transitions at once

Coordination via ports

- use port need to be provided before reaching places
- places cannot be left while provide ports are used



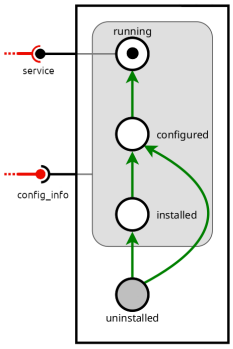
Execution semantics of a control component

"Petri net" style of semantics

- can be in multiple places at once
- transitions not atomic
- can execute multiple transitions at once

Coordination via ports

- use port need to be provided before reaching places
- places cannot be left while provide ports are used



Is this reconfiguration?

Control component and its semantics

- life-cycle evolution through time
- coordination with other life-cycles

Is this reconfiguration?

Control component and its semantics

- life-cycle evolution through time
- coordination with other life-cycles

reconfiguration

- not sufficient for reconfiguration
- need for a reconfiguration language to modify an assembly of component instances
- need to manipulate sequences of behaviors

Concerto - (2/2) reconfiguration language

Add
Add a component instance to the current assembly

Remove
Remove a component instance from the current assembly

Connect
Connect two component instances with compatible ports

Disconnect
Disconnect two component instances

Push behavior
Push a behavior to the behavior queue on a component instance

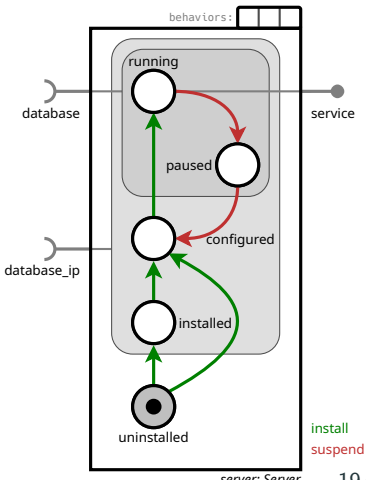
Wait
Wait for a given behavior of a component instance

Reconfiguration language semantics

Written by the reconfiguration developer

Deployment program

```
1 add(server: Server)
2 add(db: Database)
3 con(server.database_ip,db.ip)
4 con(server.database,db.service)
5 pushB(server, install)
6 pushB(db, deploy)
7 wait(server)
```

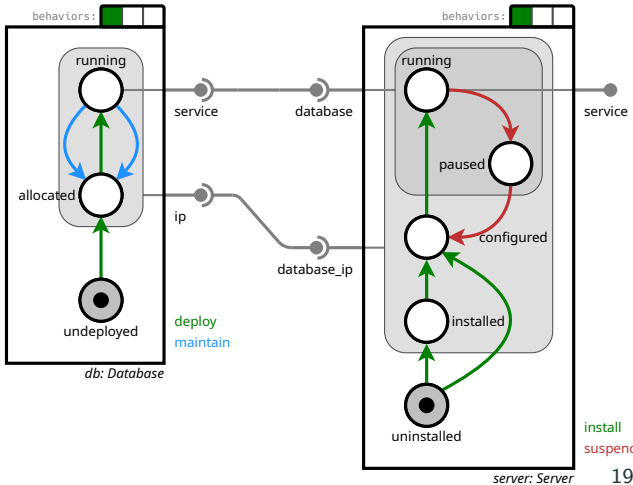


Reconfiguration language semantics

Written by the reconfiguration developer

Deployment program

```
1 add(server: Server)
2 add(db: Database)
3 con(server.database_ip,db.ip)
4 con(server.database,db.service)
5 pushB(server, install)
6 pushB(db, deploy)
7 wait(server)
```

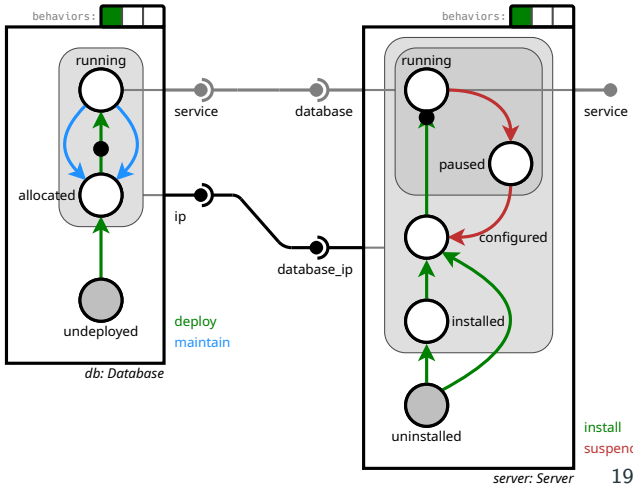


Reconfiguration language semantics

Written by the reconfiguration developer

Deployment program

```
1 add(server: Server)
2 add(db: Database)
3 con(server.database_ip,db.ip)
4 con(server.database,db.service)
5 pushB(server, install)
6 pushB(db, deploy)
7 wait(server)
```

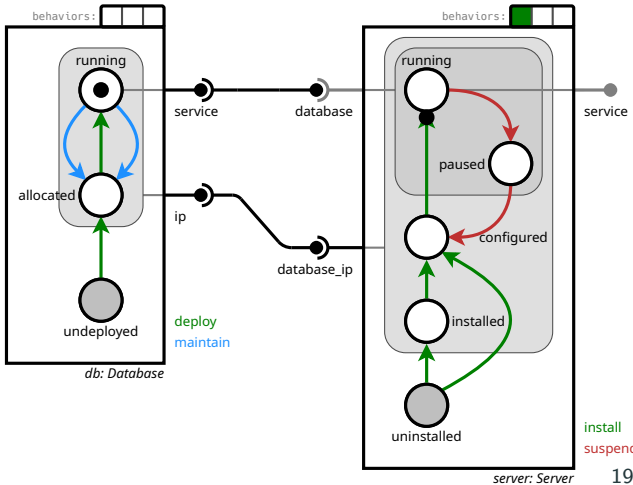


Reconfiguration language semantics

Written by the reconfiguration developer

Deployment program

```
1 add(server: Server)
2 add(db: Database)
3 con(server.database_ip,db.ip)
4 con(server.database,db.service)
5 pushB(server, install)
6 pushB(db, deploy)
7 wait(server)
```

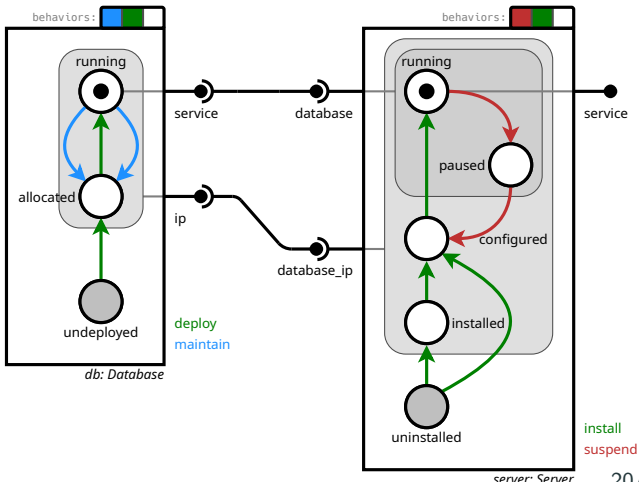


Reconfiguration language semantics

Written by the reconfiguration developer

Maintenance program

```
1 pushB(db, maintain)
2 pushB(db, deploy)
3 pushB(server, suspend)
4 pushB(server, install)
5 wait(server)
```

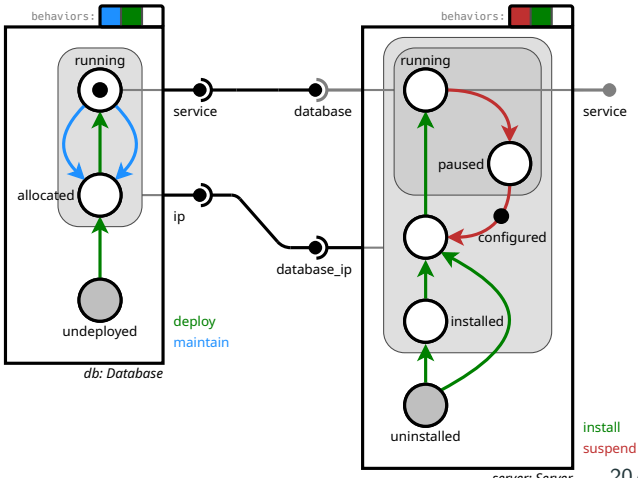


Reconfiguration language semantics

Written by the reconfiguration developer

Maintenance program

```
1 pushB(db, maintain)
2 pushB(db, deploy)
3 pushB(server, suspend)
4 pushB(server, install)
5 wait(server)
```

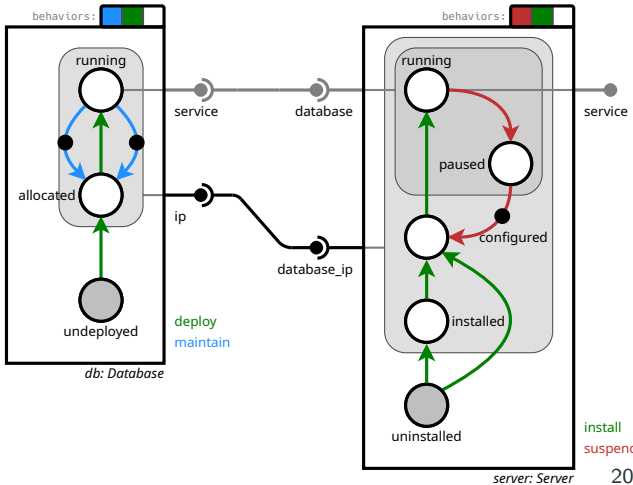


Reconfiguration language semantics

Written by the reconfiguration developer

Maintenance program

```
1 pushB(db, maintain)
2 pushB(db, deploy)
3 pushB(server, suspend)
4 pushB(server, install)
5 wait(server)
```



Reconfiguration language in practice

Written by the reconfiguration developer

```
1  class ServerClient(Assembly):
2      def __init__(self):
3          self.server = Server()
4          self.database = Database()
5          Assembly.__init__(self)
6
7      def deploy(self):
8          self.add_component('database', self.database)
9          self.add_component('server', self.server)
10         self.connect('server', 'database_ip', 'database', 'ip')
11         self.connect('server', 'database', 'database', 'service')
12         self.push_b('server', 'install')
13         self.push_b('database', 'deploy')
14         self.wait_all()
```

Reconfiguration language in practice

Written by the reconfiguration developer

```
class ServerClient(Assembly):
    ...

    def maintain(self):
        self.push_b('database', 'maintain')
        self.push_b('database', 'deploy')
        self.push_b('server', 'suspend')
        self.push_b('server', 'install')
        self.wait_all()
```

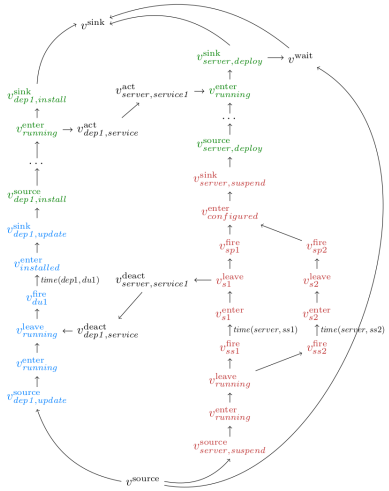
Performance prediction

Inputs

- reconfiguration program
- time estimations for transitions

Dependency graph generation

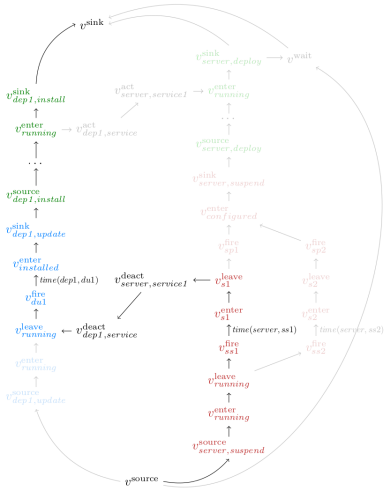
- nodes for events such as reaching/leaving place, firing transition
- transition arcs are weighted to reflect execution time
- other arcs are 0-weighted



Performance prediction

Critical path

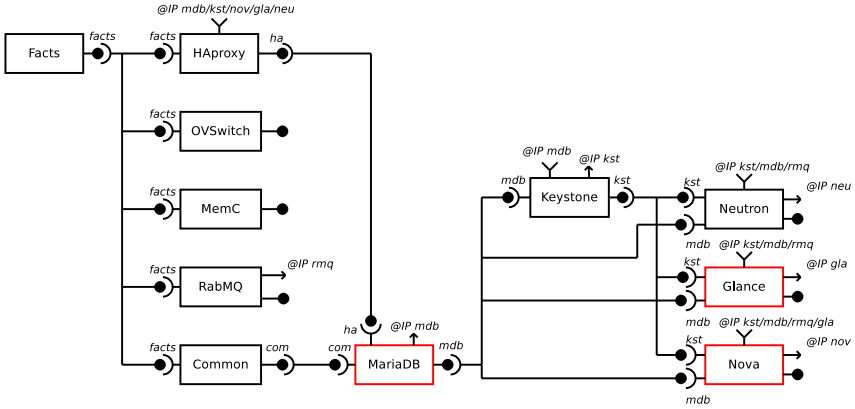
- length = reconfiguration time (assuming hardware can execute as many concurrent threads as needed)
- highlights the transitions that should be optimized



A few results

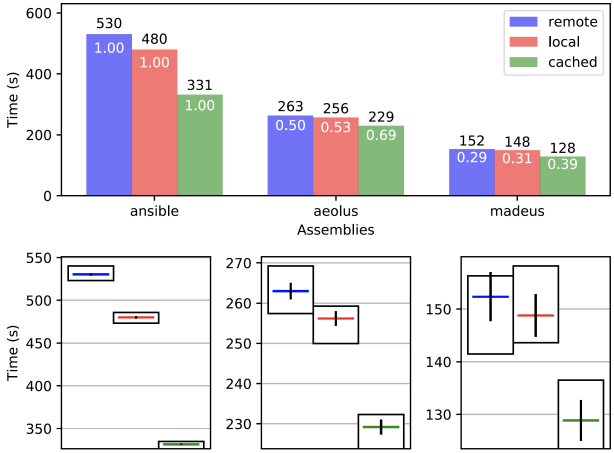
Evaluation on the deployment of OpenStack

- subpart of OpenStack: 11 components, 36 services in total
- Comparison to Kolla-Ansible (production tool), and Aeolus (literature)
- Reproducible experiments on Grid'5000



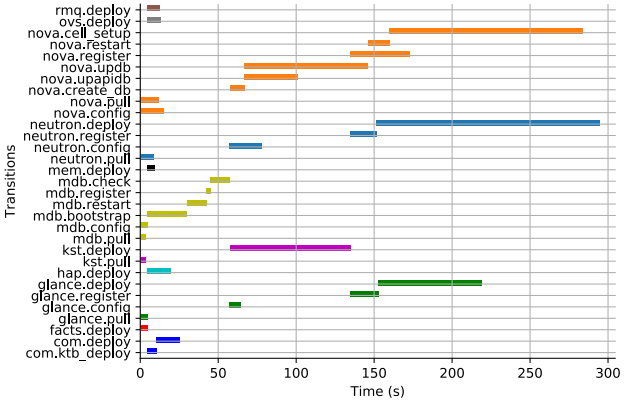
Evaluation on the deployment of OpenStack

Results on three nodes Ecotype (Nantes) of Grid'5000

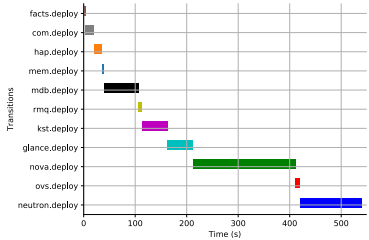


Evaluation on the deployment of OpenStack

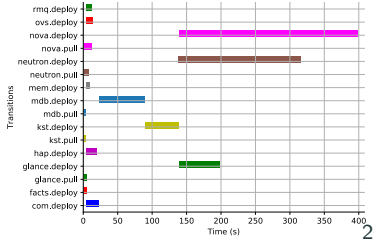
[concerto]



[ansible]



[aeolus]



Evaluation on the deployment of OpenStack

- Traces of the OpenStack continuous Integration platform
- February 19 to February 27 2020
- Exactly 2963 deployments of OpenStack have been recorded (329 runs per day)
- Projection of the gain with deployment times of our experiments in remote mode

	Kolla	Madeus	gain
reference time(s)	529	150	71%
projection on 9 days(h)	435	123	71%
projection on av./day(h)	48	14	71%

Evaluation on the reconfiguration of MariaDB

Real use-case extracted from the OpenStack Summit 2018 on a multi-region deployment of OpenSatck

Initial state

- centralized MariaDB running
- additional nodes running some generic components (docker, pip. . .)

decentralization

- replaces centralized DB with a distributed version with instances on n nodes
- requires a backup of the data, and a restart of the master node

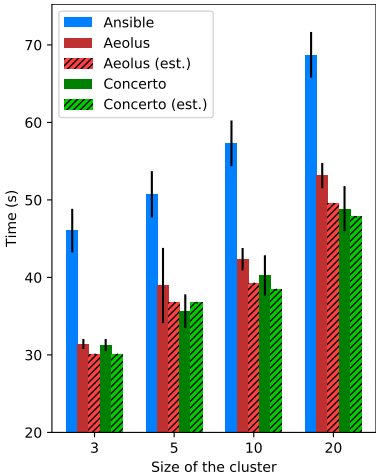
scaling

- deploys m new nodes with an instance of the distributed DB

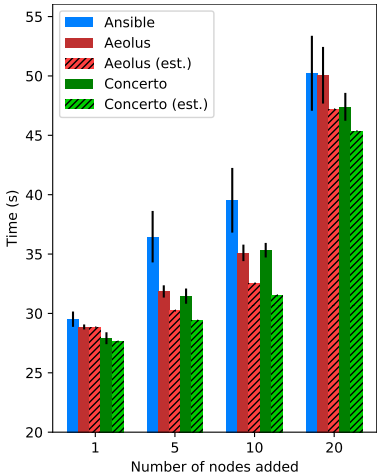
Evaluation on the reconfiguration of MariaDB

Results on nodes of UvB (Sophia) of Grid'5000

Decentralization



Scaling



Verification of properties

Verification of deployments

Hypothesis: *the deployment of a distributed software system already exists and the developer wants to use Madeus to enhance its efficiency*

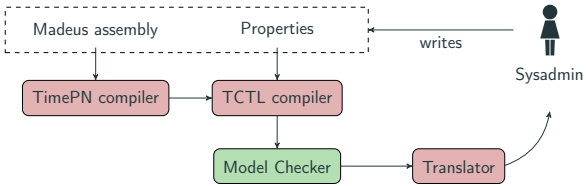
1. how to enhance the efficiency without running the deployment?
2. how to avoid safety issues such as deadlocks without running the deployment?

Goal

Study the use of model checking to help in the two above challenges

Verification of deployments

- Qualitative properties
- Quantitative properties



Properties (1/2)

- Time Petri nets are used
  - intervals of time given for each transition representing a Madeus transition

```
def set_interval(self, component, transition, min, max)
def add_deployment(self, name, dict_componentsplaces)
```

- High Abstraction Level Properties (HALP)
  - qualitative properties
  - quantitative properties

```
def deployability(self, deployment_name, with_intervals)
def sequentiality(self, ordered_list_components_transition)
def forbidden(self, list_marked, list_unmarked)
def parallelism(self, full_assembly, list_components)
def gantt_boundaries(self, deployment_name, mini, maxi, critical)
```

Properties (2/2)

HALP automatically transformed to TCTL (Time Computational Tree Logic) formulae

Qualitative properties

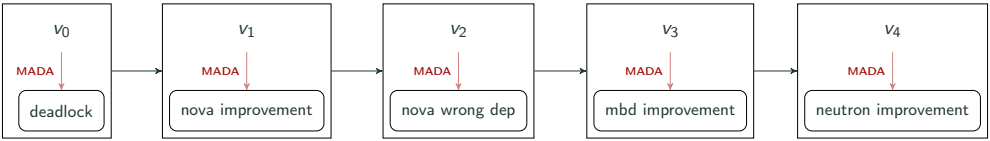
- deployability $\longrightarrow$ inevitability
- sequentiality $\longrightarrow$ observer subnet + invariant
- forbidden $\longrightarrow$ observer subnet + invariant

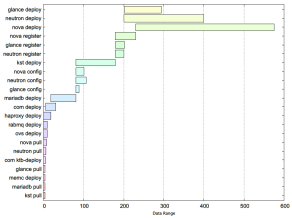
Quantitative properties

- parallelism $\longrightarrow$ max($\sum$(reachable markings))
- gantt boundaries: min/max costs + causality in the trace to get the critical path

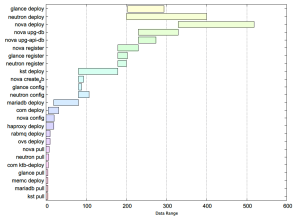
Evaluation (1/2)

5 versions of the OpenStack deployment successively enhanced with MADA





(a) 1-naive with critical path: *nova deploy, nova register, kst deploy, mariadb deploy, haproxy deploy*



(b) 2-nova with critical path: *nova deploy, nova upg-db, nova register, kst deploy, mariadb deploy, haproxy deploy*

Evaluation (2/2)

Experiments conducted with the model checker Romeo

	0-deadlock	1-naive	2-nova	3-nova	4-nova-mdb
Madeus places	27	27	28	28	29
Madeus transitions	22	22	25	25	28
Madeus connections	30	30	30	30	30
Petri net places	113	113	124	124	134
Petri net transitions	75	75	84	84	92
Transformation time (ms)	1.6	1.6	1.8	1.7	1.5
Deployability	False	True	True	True	True
Resolution time (s)	0	41.6	78.7	88.7	152.6
Parallelism nova	-	1	2	2	2
Resolution time (s)	-	42.1	82.7	93.6	154.3
Parallelism full	-	10	11	11	12
Resolution time (s)	-	43.2	86.1	98.4	162.9
Gantt & critical path	-	Fig	Fig	Fig	Fig
Resolution time (s)	-	130.1	266.9	275.4	588.1
Boundaries	-	[575,615]	[518,554]	[400,423]	[377,398]
Resolution time (s)	-	130.1, 128.8	266.9, 269.7	275.4, 267.6	588.1, 580.8

Formal methods for reconfigurations?

Simon Robillard, postdoc

Verification of Concerto programs

- bigger search space for Concerto with behaviors
- find the minimal interface needed for verification without entering a component
- verification by composition

Verification in the (P) phase

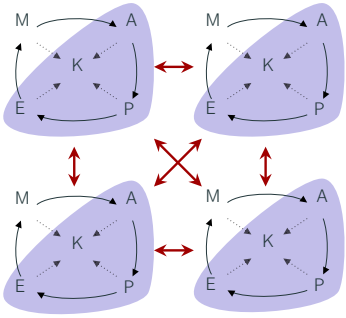
- inputs: partial specification of the reconfiguration
  - current configuration
  - target configuration
  - subset of behaviors to apply
- output: complete and correct reconfiguration plan

Perspectives

Perspectives

- Inference of correct-by-design Concerto programs (P)
- Inference of correct-by-design target configuration (A)
  - inputs: current configuration and monitored events
  - output: new target configuration
- Abstraction level for developers to ease the use of Concerto
  - control component patterns
  - reconfiguration patterns
- Integration in well known devops tools
  - work in progress with Madeus and Ansible
  - ongoing project on Kubernetes

SeMaFoR project - Distributed autonomic reconfiguration



SeMaFoR project

- Led by *Thomas Ledoux*
- Self Management of Fog Resources
- Work-package leader on decentralized reconfiguration
- Hiring one postdoc starting in March 2022