

Coordination-aware Elasticity

Stefano Mariani¹, Hong-Linh Truong², Georgiana Copil², Andrea Omicini¹,
Schahram Dustdar²

¹DISI, Alma Mater Studiorum, Università di Bologna

²Distributed Systems Group, Vienna University of Technology

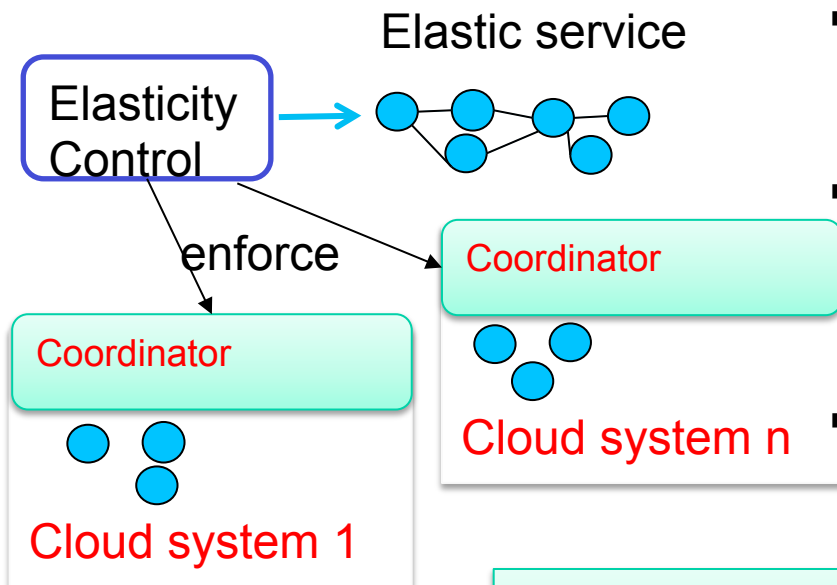
truong@dsg.tuwien.ac.at

<http://dsg.tuwien.ac.at/research/viecom>

- Motivation
- Approach to coordination-aware elasticity
- Integrating coordination and elasticity
- Prototype and examples
- Conclusions and future work

Motivation

- Elasticity for complex services in multiple cloud environments needs to move from “control” to “coordination”



- Current development/operation issues

- *Often Hard-code, design-time only parameters* (waiting times, number of tries, targets of operations, etc.)
- *Poor synchronisation* between enforcement mechanisms (hard-coded or undefined waiting times, missing knowledge about operations success/failure, etc.)
- *Lack of portability and reusability due to* low-level cloud APIs

Goal: Accelerating the development of elasticity through the notion of **coordination-aware elasticity**



Coordination-aware Elasticity Approach

- We need better ways to support **flexibility**, **safety** and **encapsulation**, and **separation and delegation** of concerns
 - the new way to develop elasticity and coordination in clouds
- **Elasticity programming directives**
 - **delegate** elasticity control to infrastructures and **separate** elasticity control from application logic
- **Coordination laws**
 - **delegation** of interaction management to middleware and **separation** of interactive behaviour from computational one

Coordination-aware Elasticity Approach

- Separate concerns between elasticity and coordination
 - Each cloud system has its own coordination model
 - Elasticity exploits coordination features
 - Both deal with **managing run-time dependencies in their own scopes**
- Benefits
 - Support **run-time** delegation and separation of concerns
 - Guarantee **safety** of interactions between elasticity controllers and cloud components/services/plugins
 - Improve **availability** of elasticity controllers, by delegating coordination-related computations
 - Ease **development** process, by enabling and supporting separation of duties and responsibilities between “elasticity developers” and underlying cloud environments

This paper – the first step

- **mapping** between elasticity and coordination abstractions
- **“usage patterns”** involving integrated usage of elasticity control and coordination
- An initial prototype of coordination-aware elasticity based on **rSYBL** elasticity controller and **ReSpecT** coordination framework

Integrating Concepts

Elasticity

Elasticity Service/
Resource

Elasticity
Infrastructure

Programming
Directive

Coordination

Agent
(Coordinable)

Coordination
Media

Coordination Law

Concepts from between two worlds must be mapped

- Semantics of concepts are easy to understand and map
- Implementation/syntax/performance are quite different



Integrating Concepts

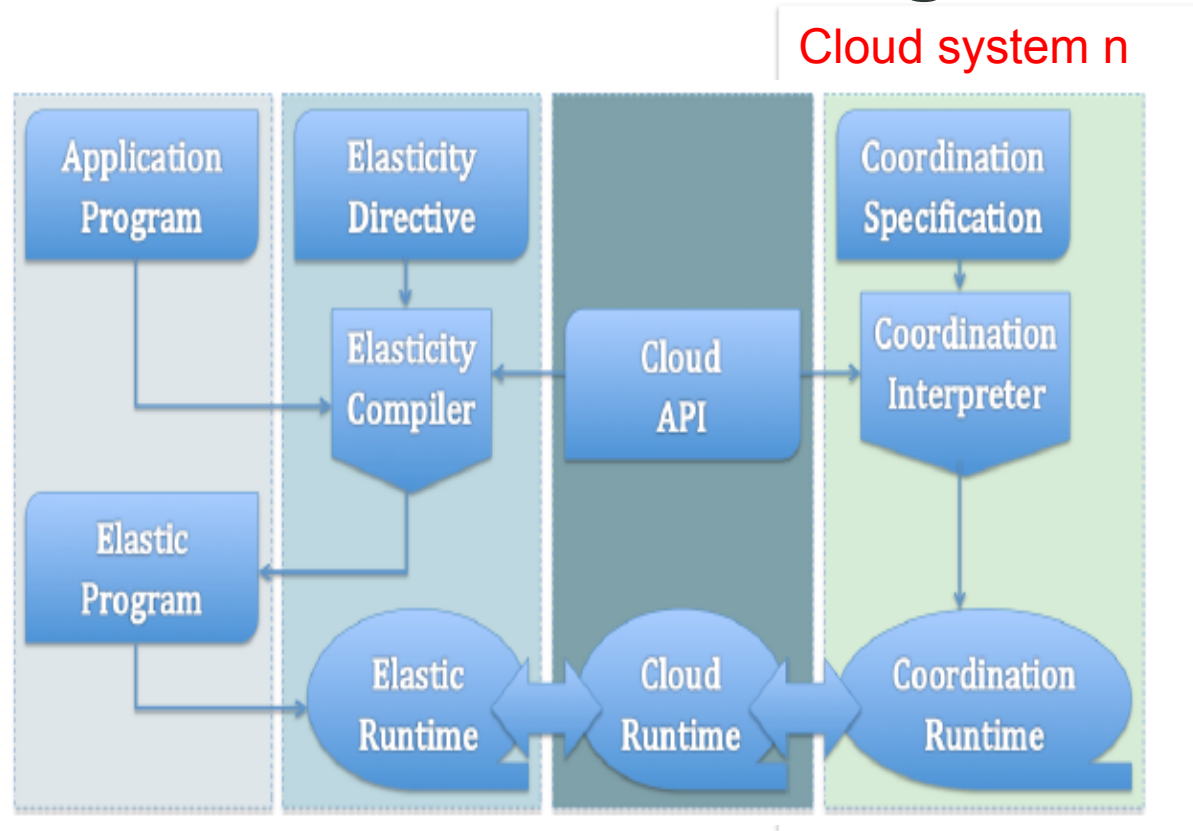
- Elastic service units and computational resources are the entities *subject of the coordination process* (the **coordinables**)
- Such a process is supported by a suitable **elastic coordination infrastructure** composed by a *network of distributed coordination media and elastic components*
- Such components are responsible for the *run-time enforcement and programmability* of the desired **elastic coordination directives**

Integrating Languages

Elasticity Directives	Coordination Laws
Directive	Law
Runtime Functions	Primitives
Monitoring	Event
Constraint	Observation
Strategy	Communication

- **monitoring** primitives and coordination **events**: enabling the system to react to changes in its state
- **constraints** and **observation** (guards): controlling execution of (elastic/coordination) computations in response to (elastic/coordination) events
- **Strategies** and **coordination computations**: allow the elastic/coordination infrastructure to perform computations in response to elastic/coordination events

Runtime and APIs integration



- elasticity and coordination controllers work in tight synergy
- the coordination-aware elasticity enforcement process dynamically delegates responsibilities to the right component

Prototype and Examples

- Prototype
 - SYBL language and runtime (rSYBL) - for elasticity
 - the ReSpecT language and runtime (TuCSoN) — for coordination
 - TuCSoN is used for coordinating resources in a single cloud system
 - rSYBL is used to control elasticity across systems
- GITs
 - <https://github.com/tuwiendsg/rSYBL>
 - <https://github.com/tuwiendsg/rSYBL/tree/coordination>

Example of delegation

```

1 if (actionName.contains("scaleout")) {
2   // coordinated scale out requested
3   ... // parse arguments
4   this.doCoordinatedScaleOut(args, e);
5 } else if (actionName.startsWith("scalein")) {
6   // coordinated scale in requested
7   ... // parse arguments
8   this.doCoordinatedScaleIn(args, e);
9 } else if (actionName.startsWith("monitorMetrics")) {
10  // coordinated monitoring requested
11  ... // parse arguments
12  this.doCoordinatedMonitorMetrics(args, e);
13 } else {
14  this.respect.delegate( // basic ReSpecT operation
15    actionName,
16    RespectEnforcementAPI.OP_TIMEOUT,
17    e);
18 }
19 // [scaleout case]
20 // read outcome from shared ReSpecT tuple centre
21 op = this.respect.delegate(
22   "in(scaleOut(" + node.getId() + "),
23   done(B))",
24   RespectEnforcementAPI.OP_TIMEOUT,
25   node);
26 this.executingControlAction = false;
27 if (op.isResultSuccess()) {
28   // scale out successful
29 } else {
30   // scale out failed
31 }

```

Run-time Benefits

Pseudocode 1 Coordination-unaware Elasticity Control

```

1: procedure SCALEOUT(node1)           ▷ elasticity control
2:   node2 ← cloudAPI.doScaleOut(node1) ▷ API lock-in
3:   if [was synch call] then             ▷ loss of control flow
4:     else if [was asynch call] then    ▷ undefined/busy wait
5:     end if
6:     if checkHealth(node2) then      ▷ scale out successful?
7:       startMonitoring(node2)
8:     else
9:       SCALEOUT(node1)                ▷ retry
10:    end if
11: end procedure
  
```

Elasticity controllers are “locked-in” to such API

synchronous cloud API calls → lose elasticity control flow, being obliged to wait until the call returns

asynchronous cloud API calls → busy-waits to know if scale out was successful?

something bad happens → elasticity controllers are responsible of failure-handling → tedious task

Run-time Benefits

Pseudocode 2 Coordination-aware Elasticity Control

```

1: procedure SCALEOUT( $n1$ )           ▷ elasticity control
2:    $coordServ \leftarrow getCoordinationService()$ 
3:    $coordServ.doScaleout(n1, tOut)$       ▷ asynch
4: end procedure
5: procedure ONSCALEOUTSUCCESS( $n2$ )
6:    $startMonitoring(n2)$ 
7: end procedure
8: procedure ONSCALEOUTFAILURE( $n1$ )
9:    $SCALEOUT(n1)$ 
10: end procedure
11: procedure ONSCALEOUTTIMEOUT( $n1$ )
12:    $SCALEOUT(n1)$ 
13: end procedure
14: procedure DOSCALEOUT( $n1, tOut$ )      ▷ coordination
15:   ...                               ▷ Read params, call cloud API, coordinate
                                   monitoring, wait completion, share result
16: end procedure

```

Scaling out is delegated to coordination services → **control flow is retained** by elasticity controllers, now free from cloud API issues (e.g. invocation semantics) and **no longer tied to the specific cloud provider**

API calls are now **asynchronous by default** → coordination guarantees replies in finite time and **failures are confined to** coordination services

Conclusions and Future work

- A new approach: **coordination-aware elasticity**
 - Programming and engineering elasticity across multiple clouds need better software engineering techniques
 - Elasticity languages and runtimes should *leverage coordination capabilities of underlying cloud systems* to deal with complex elasticity scenarios → **delegation** and **separation** of concerns
- However, it is just an early stage with simple coordination models
 - Cloud systems would have complex coordination models/languages
- Future work
 - Real-world coordination models for single cloud systems
 - Software languages and tools for engineering

Thanks for your attention!

Hong-Linh Truong

Distributed Systems Group
TU Wien

truong@dsg.tuwien.ac.at
dsg.tuwien.ac.at/research/viecom