Auto-Scaling Mechanism in the ICT converged Cross Stratum Orchestration Architecture for Zero-Touch Service and Network Management

Young Lee, Ricard Vilalta*, Ramon Casellas*, Ricardo Martínez*, Raul Muñoz*
Huawei USA Research Center
** Centre Tecnològic de Telecomunicacions de Catalunya (CTTC/CERCA)*
Tel: +1 (214) 244 7652, e-mail: leeyoung@huawei.com

ABSTRACT

This paper is aimed to provide a novel approach for an auto-scaling mechanism that are applied in the ICT converged cross stratum orchestration (CSO) architecture. CSO refers to an end-to-end orchestration across Application orchestration, Data Center SDN orchestration, and WAN SDN orchestration so that applications can be created seamlessly and optimally for operators and their customers. The auto-scaling mechanism presented in this paper is built on active monitoring of both DC compute/storage/network resources and DCI Transport network resources and then provides dynamically configurable auto-scaling and resource re-allocation. This mechanism will help tremendously the operators to control and manage their networks in an automated fashion.

Keywords: CSO, ICT convergence, ACTN, Orchestration, SDN, automation, auto-scaling, zero-touch service management.

1. INTRODUCTION

Distributed computing environments allow end-users, from individual to enterprises, to access to large pools of storage resources, computational resources and various application services (e.g., Video Caching, Virtual Machine mobility, media content delivery, IoT, etc.). Data centers provide the physical and virtual infrastructure in which applications and services are provided. Since the data centers used to provide application services are distributed geographically around a network (or a set of interconnected networks), application service instantiation can have a significant impact on the state of the network resources. Conversely the capabilities and current state of the network can have a major impact on the application performance.

A Zero-touch network is a network that minimizes service downtime and operating costs thanks to the removal of human intervention. A zero-touch network needs end-to-end network programmability for service creation and full automation of the service life-cycle. According to [1], a zero-touch network is built using cloud and SDN technologies with cloud-optimized network applications and network slices to address the needs of different consumer, and vertical industry.

This paper is aimed to provide a novel approach for an auto-scaling mechanism that are applied in the ICT converged cross stratum orchestration (CSO) architecture [2]. CSO refers to an end-to-end orchestration across Application orchestration, Data Center SDN orchestration, and WAN SDN orchestration so that applications can be created seamlessly and optimally for operators and their customers. The auto-scaling mechanism presented in this paper is built on active monitoring of both DC compute/storage/network resources and DCI Transport network resources and then provides dynamically configurable auto-scaling and resource re-allocation. This mechanism will help tremendously the operators to control and manage their networks in an automated fashion.

2. CSO ARCHITECTURES AND INTERFACES

Figure 1 shows the cross-stratum orchestrator between the application stratum and the network stratum interfacing different types of orchestrator/controllers [2].

The Data Center orchestrator provides its Data Center network resource abstraction pertaining to the applications to the CSO. Since application services may use resources in multiple Data Centers via Data Canter Interconnect (DCI), each Data Center orchestrator involving in the application service should provide its resource abstraction to the CSO so that the CSO would be able to compute the optimal resource sequence and path meeting the service objective [3][4].

The WAN SDN controller(s) is (are) also involved for an end-to-end service instantiation and its life cycle operation that may traverse multiple Data Centers dispersed in multiple domains. WAN SDN controllers may also comprise multiple hierarchical SDN controllers, each of which is responsible for domain control of IP or optical networks in multiple domain networks. ACTN addresses hierarchical SDN controller interactions and their interfaces [5].

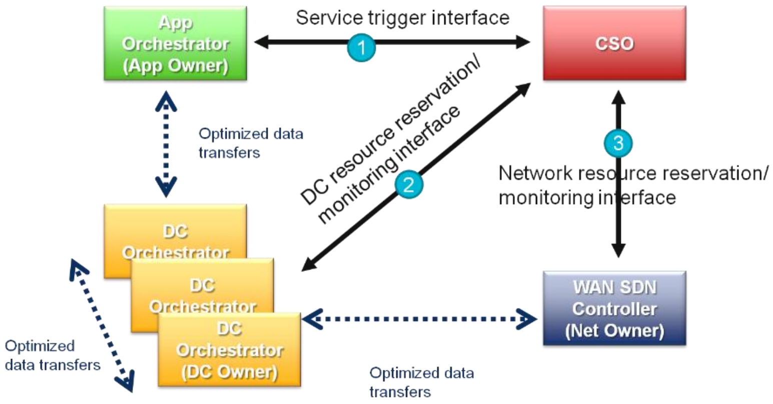

App Orchestrator (App Owner)
Service trigger interface
1
CSO
Optimized data transfers
DC resource reservation/ monitoring interface
2
3
Network resource reservation/ monitoring interface
DC Orchestrator
DC Orchestrator
DC Orchestrator (DC Owner)
WAN SDN Controller (Net Owner)
Optimized data transfers
Optimized data transfers

Figure 1. CSO Architecture

The definition of CSO interfaces [2] is recapitulated as follows:

- CSO interface type 1 is referred to as the Service Trigger Interface. This interface may have common components and functionality with typical North Bound Interface (NBI). The service intent is expressed on this interface. Service intent generally encompasses application context, SLA/QoS requirements as well as policy associated with services/applications.

- CSO interface type 2 is referred to as the DC Resource Reservation and Monitoring Interface. This interface on a high level abstracts a DC level resource abstraction as well as a host/server level resource abstraction needed per application. This interface is also referred to as DC API in Figure 2.

- CSO interface type 3 is referred to as the Network Resource Reservation and Monitoring Interface. This interface should provide several functionalities such as: abstraction of the network resource information of the operator's transport networks providing DC interconnection, service connection reservation request, and monitoring data and measurement pertaining to the service connection among the key requirements. This interface is referred to as CNC-MSDC Interface (CMI) in ACTN architecture. CSO Service Orchestrator in Figure 1 is a type of Customer Network Controller (CNC).

There are several steps that are involved to enable auto-scaling mechanisms for both DCs and WAN.

2.1 Performance Monitoring Notification Subscription Flow

CSO SO creates performance parameters for the application and sends the subscription message to MDSC and DC Controllers. The subscription notification is recursively sent down PNCs and NE's for network parameters and the DC for compute parameters. Figure 2.a shows notification subscription flow.

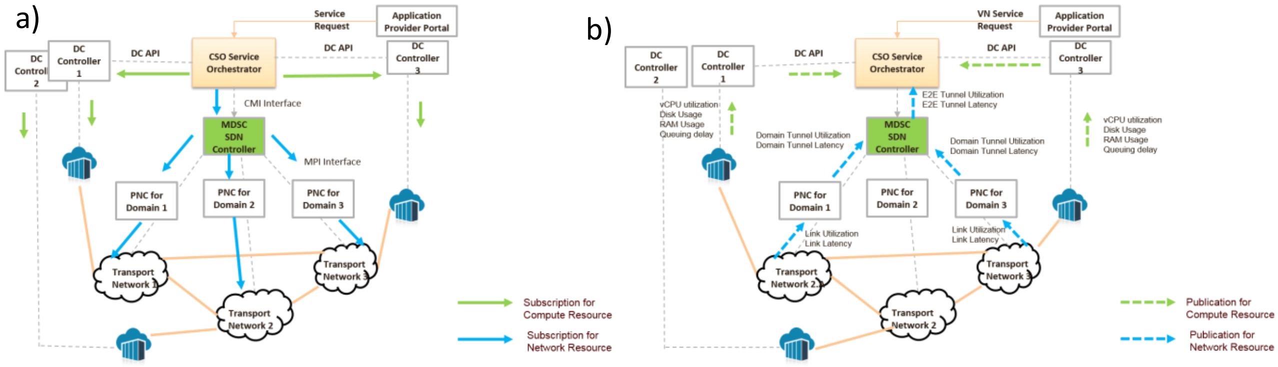


Figure 2. a) PM Notification Subscription Mechanism; b) PM Notification Publication Mechanism

The CSO Service Orchestrator (CSO SO) is the initiator of subscription of DC performance data and WAN performance data. CSO SO creates performance parameters for the application and sends the subscription message to MDSC and DC Controllers. As for the WAN network performance monitoring, the subscription notification is recursively sent down from MDSC to PNCs and NE's for network parameters. As for the DC network performance monitoring, the subscription notification is sent from CSO OS to DC Controllers for compute parameters.

2.2 Performance Monitoring Notification Publication Flow

Each network elements, DC elements push/publish the subscribed parameters to its NBI when the conditions are met. The network related publication notification is recursively sent up to PNCs and MDSC and CSO for network parameters. Likewise, the Data Center compute related publication notification is recursively sent up to DC controllers and the CSO for compute resources. Figure 2.b shows notification publication flow.

2.3 Auto Scaling Mechanism (Re-optimization and Re-Configuration)

Based on the performance data streamed to the CSO SO, the CSO SO re-optimizes the existing E2E provisioning path and trigger the re-configuration request message to its downstream. Re-optimizing and re-configuring of the tunnel are based on the performance monitoring data collected via the notification subscription and publication mechanism described in Sections 2.1 and 2.2. Figure 3.a shows a re-configuration flow.

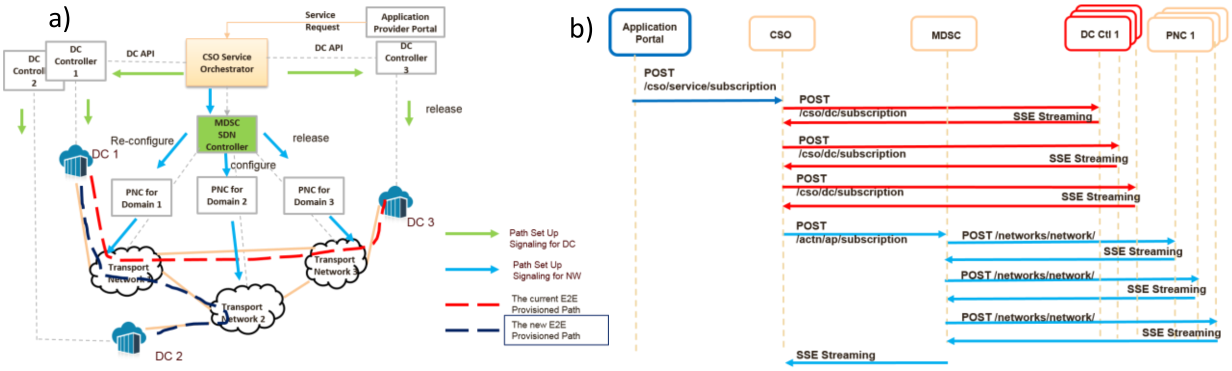


Figure 3. a) Re-configuration flow; b) CSO PM Notification/Subscription Automated Workflow.

Figure 3.a shows the case where the destination DC 3 is changed to DC 2 as the destination for the tunnel because the customer requirement for the service can no longer meet due to either the deficiencies of the DC 2 resources and/or its under-performing performance (e.g., queuing delay in the DC) or the network tunnel resources deficiencies caused by, for example, high utilization, delay, or failure.

For example, the following auto-scaling scheme can be implemented via YANG configuration: *If {Tunnel Utilization >= 90%} AND {Tunnel Latency >= 100ms} AND {Destination DC Queuing Delay >= 50ms}, Then Re-optimize and Re-configure the Tunnel.*

2.4 Automated Notification Subscription and Publication Workflow

Figure 3.b shows the control workflow for PM notification sub/pub mechanism described in Sections 2.1-2.3. Figure 3.b shows the automated intent-based auto-scaling mechanism that enables downstream notification/publication workflow from the Application Portal’s perspective in order to set the scaling thresholds based on the PM data triggered by the Application Portal.

Four steps are included in this process:

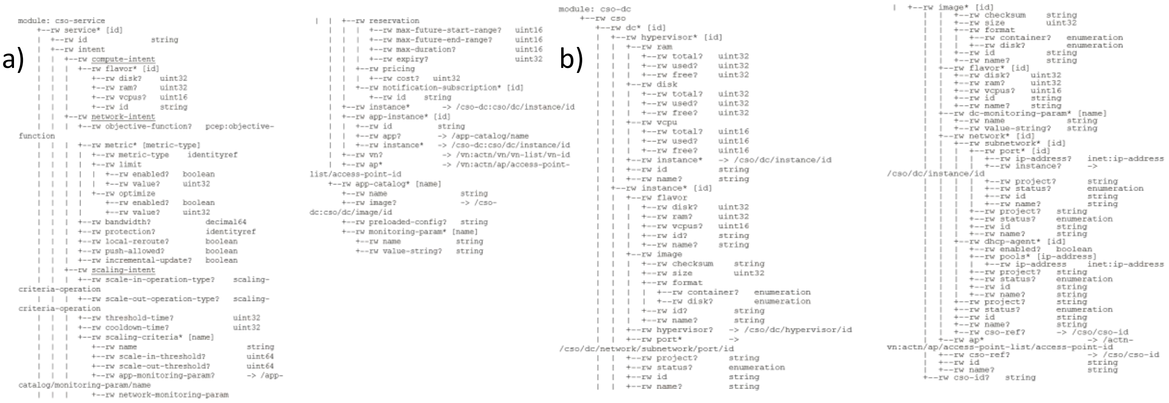
1) Service Intent and scaling in and out configuration by the Application Portal to the CSO.
2) Subscription/Notification of underlying DC PM data using OpenStack API [6] towards/from DC controller.
3) Subscription/Notification of underlying PM data using TE Topology towards/from MDSC and PNCs.
4) Re-Optimization and Re-configuration by the CSO when thresholds are hit as described in [2].

3. CSO YANG DATA MODELS

3.1 CSO Service Intent Model

CSO Service Intent can be described in the following YANG module (cso-service), as shown in Figure 4.a. Service intent is described for both compute-intent and network intent. Based on the measured data and scale-in and out operations set by the scaling-intent grouping, automatic scaling mechanism can be configured dynamically by the service owner.

The CSO-service YANG data model describes for each service/application, its intent that describes the requirements for the service and application. The intent may have a compute-intent (e.g., disk space, RAM, VCPUs, etc.), a network intent (e.g., objective function for the tunnel, QoS metric (e.g., latency, packet drop rate, etc.), tunnel protection and so on, and scaling-intent that are programmable per service/application. Any metric can be combined together to determine whether pre-set thresholds are hit, using the operation types such as OR, AND, MAX, MIN, AVG, etc. applied to the metric types.



```
module: cso-service
   +--rw service* [id]
   |  +--rw id              string
   |  +--rw intent
   |  |  +--rw compute-intent
   |  |  |  +--rw flavor* [id]
   |  |  |     +--rw disk?    uint32
   |  |  |     +--rw ram?     uint32
   |  |  |     +--rw vcpus?   uint16
   |  |  |     +--rw id       string
   |  |  +--rw network-intent
   |  |  |  +--rw objective-function?   pcep:objective-
function
   |  |  |  +--rw metric* [metric-type]
   |  |  |  |  +--rw metric-type    identityref
   |  |  |  |  +--rw limit
   |  |  |  |  |  +--rw enabled?   boolean
   |  |  |  |  |  +--rw value?     uint32
   |  |  |  |  +--rw optimize
   |  |  |  |     +--rw enabled?   boolean
   |  |  |  |     +--rw value?     uint32
   |  |  |  +--rw bandwidth?           decimal64
   |  |  |  +--rw protection?          identityref
   |  |  |  +--rw local-reroute?       boolean
   |  |  |  +--rw push-allowed?        boolean
   |  |  |  +--rw incremental-update?  boolean
   |  |  +--rw scaling-intent
   |  |  |  +--rw scale-in-operation-type?    scaling-
criteria-operation
   |  |  |  +--rw scale-out-operation-type?   scaling-
criteria-operation
   |  |  |  +--rw threshold-time?             uint32
   |  |  |  +--rw cooldown-time?              uint32
   |  |  |  +--rw scaling-criteria* [name]
   |  |  |     +--rw name                        string
   |  |  |     +--rw scale-in-threshold?         uint64
   |  |  |     +--rw scale-out-threshold?        uint64
   |  |  |     +--rw app-monitoring-param?       -> /app-
catalog/monitoring-param/name
   |  |  |     +--rw network-monitoring-param
   |  |  +--rw reservation
   |  |  |  |  +--rw max-future-start-range?   uint16
   |  |  |  |  +--rw max-future-end-range?     uint16
   |  |  |  |  +--rw max-duration?             uint16
   |  |  |  |  +--rw expiry?                   uint32
   |  |  |  +--rw pricing
   |  |  |  |  +--rw cost?   uint32
   |  |  |  +--rw notification-subscription* [id]
   |  |  |     +--rw id    string
   |  +--rw instance*        -> /cso-dc:cso/dc/instance/id
   |  +--rw app-instance* [id]
   |  |  +--rw id           string
   |  |  +--rw app?         -> /app-catalog/name
   |  |  +--rw instance*    -> /cso-dc:cso/dc/instance/id
   |  +--rw vn?              -> /vn:actn/vn/vn-list/vn-id
   |  +--rw ap*              -> /vn:actn/ap/access-point-
list/access-point-id
   +--rw app-catalog* [name]
      +--rw name                 string
      +--rw image?               -> /cso-
dc:cso/dc/image/id
      +--rw preloaded-config?   string
      +--rw monitoring-param* [name]
         +--rw name              string
         +--rw value-string?    string
```

```
module: cso-dc
   +--rw cso
      +--rw dc* [id]
      |  +--rw hypervisor* [id]
      |  |  +--rw ram
      |  |  |  +--rw total?   uint32
      |  |  |  +--rw used?    uint32
      |  |  |  +--rw free?    uint32
      |  |  +--rw disk
      |  |  |  +--rw total?   uint32
      |  |  |  +--rw used?    uint32
      |  |  |  +--rw free?    uint32
      |  |  +--rw vcpu
      |  |  |  +--rw total?   uint16
      |  |  |  +--rw used?    uint16
      |  |  |  +--rw free?    uint16
      |  |  +--rw instance*   -> /cso/dc/instance/id
      |  |  +--rw id          string
      |  |  +--rw name?       string
      |  +--rw instance* [id]
      |  |  +--rw flavor
      |  |  |  +--rw disk?    uint32
      |  |  |  +--rw ram?     uint32
      |  |  |  +--rw vcpus?   uint16
      |  |  |  +--rw id?      string
      |  |  |  +--rw name?    string
      |  |  +--rw image
      |  |  |  +--rw checksum    string
      |  |  |  +--rw size        uint32
      |  |  |  +--rw format
      |  |  |  |  +--rw container?   enumeration
      |  |  |  |  +--rw disk?        enumeration
      |  |  |  +--rw id?         string
      |  |  |  +--rw name?       string
      |  |  +--rw hypervisor?   -> /cso/dc/hypervisor/id
      |  |  +--rw port*         ->
/cso/dc/network/subnetwork/port/id
      |  |  +--rw project?      string
      |  |  +--rw status?       enumeration
      |  |  +--rw id            string
      |  |  +--rw name?         string
      |  +--rw image* [id]
      |  |  +--rw checksum    string
      |  |  +--rw size        uint32
      |  |  +--rw format
      |  |  |  +--rw container?   enumeration
      |  |  |  +--rw disk?        enumeration
      |  |  +--rw id          string
      |  |  +--rw name?       string
      |  +--rw flavor* [id]
      |  |  +--rw disk?    uint32
      |  |  +--rw ram?     uint32
      |  |  +--rw vcpus?   uint16
      |  |  +--rw id       string
      |  |  +--rw name?    string
      |  +--rw dc-monitoring-param* [name]
      |  |  +--rw name             string
      |  |  +--rw value-string?   string
      |  +--rw network* [id]
      |  |  +--rw subnetwork* [id]
      |  |  |  +--rw port* [id]
      |  |  |  |  +--rw ip-address?   inet:ip-address
      |  |  |  |  +--rw instance?     ->
/cso/dc/instance/id
      |  |  |  |  +--rw project?      string
      |  |  |  |  +--rw status?       enumeration
      |  |  |  |  +--rw id            string
      |  |  |  |  +--rw name?         string
      |  |  |  +--rw project?   string
      |  |  |  +--rw status?    enumeration
      |  |  |  +--rw id         string
      |  |  |  +--rw name?      string
      |  |  +--rw dhcp-agent* [id]
      |  |  |  +--rw enabled?   boolean
      |  |  |  +--rw pools* [ip-address]
      |  |  |  |  +--rw ip-address    inet:ip-address
      |  |  |  +--rw project?   string
      |  |  |  +--rw status?    enumeration
      |  |  |  +--rw id         string
      |  |  |  +--rw name?      string
      |  |  +--rw project?      string
      |  |  +--rw status?       enumeration
      |  |  +--rw id            string
      |  |  +--rw name?         string
      |  |  +--rw cso-ref?      -> /cso/cso-id
      |  +--rw ap*                    -> /actn-
vn:actn/ap/access-point-list/access-point-id
      |  +--rw cso-ref?               -> /cso/cso-id
      |  +--rw id                     string
      |  +--rw name?                  string
      +--rw cso-id?   string
```

Figure 4. a) CSO Service YANG; b) CSO DC YANG

3.2 CSO DC Subscription Model

CSO-DC YANG model is shown in Figure 4.b. The following YANG module describes cso-dc subscription model based on [6].

3.3 CSO Network Subscription Model

A YANG model to collect network PM data and its publication between the MDSC and PNCs in Figure 3.b is depicted in [7]. The model allows for aggregated PM data collection for Virtual Network (VN) or Tunnel level as well as LSP, Link/Node level. Auto-scaling in and out mechanisms are set based on the PM data collected to trigger automated re-configuration or re-optimization.

4. CONCLUSIONS

Several innovations are presented in this paper. It provides a novel approach for an auto-scaling mechanism that is applied in the ICT converged cross stratum orchestration (CSO) architecture. Auto-scaling mechanism is recursive in nature from application/service down to DC level as well as Network level. This mechanism allows for end-to-end orchestration across Application orchestration, Data Center SDN orchestration, and WAN SDN orchestration so that application can not only monitor its service performance it sets but also configure dynamic scaling thresholds. The auto-scaling mechanism presented in this paper is built on active monitoring of both DC compute/storage/network resources and DCI Transport network resources and then provides dynamically configurable auto-scaling and resource re-allocation. This mechanism will help tremendously the operators to control and manage their networks in an automated fashion.

ACKNOWLEDGEMENTS

The research leading to these results has received funding from EC H2020 5GPPP project METRO-HAUL (761727) and MINECO project DESTELLO (TEC2015-69256-R).

REFERENCES

[1] S. Van Rossem, et al., "Nfv service dynamicity with a devops approach: demonstrating zerotouch deployment & operations," in IM2017, pp. 865–866.

[2] R. Vilalta, R. Casellas, R. Martinez, R. Muñoz, Y. Lee, H. Zheng, Y. Lin, Victor Lopez, Luis Miguel Contreras, "Fully Automated Peer Service Orchestration of Cloud and Network Resources Using ACTN and CSO", ONDM 2018, to appear in May 14, 2018.

[3] M. Betts, D. Hood, Y. Lee, R. Vilalta, "Mapping Cross Stratum Orchestration (CSO) to the SDN architecture", ONF TR-528, Issue 1, March 11, 2016.

[4] H. Yang, J. Zhang. Y. Zhao, Y. Ji, J. Han, Y. Lin, Y. Lee, "CSO: cross stratum optimization for optical as a service", IEEE Communications Magazine, Volume: 53, Issue: 8, August 2015.

[5] D. Cecarelli and Y. Lee, "Framework for Abstraction and Control of Traffic Engineered Networks", draft-ietf-teas-actn-framework, work in progress.

[6] OpenStack API. http://developer.openstack.org/api-guide/quick-start/

[7] Y. Lee (Editor), "YANG models for ACTN TE Performance Monitoring Telemetry and Network Autonomics", https://datatracker.ietf.org/doc/draft-lee-teas-actn-pm-telemetry-autonomics, work in progress.