FUTURE COMMUNICATION ARCHITECTURE FOR MOBILE CLOUD SERVICES
Acronym: MobileCloud Networking Project No: 318109
Integrated Project FP7-ICT-2011-8 Duration: 2012/11/01-2015/09/30

D4.5: Mobile Network Cloud Component Evaluation

Type	Prototype
Deliverable No:	D4.5
Workpackage:	WP4
Leading partner:	University of Twente
Author(s):	Luuk Hendriks (Editor) List of Authors overleaf
Dissemination level:	Public (P)
Status:	Final
Date:	30 June 2015
Version:	1.0

List of Authors (in alphabetical order):

Marius Corici	Fraunhofer
Luuk Hendriks	UTWENTE
Morteza Karimzadeh	UTWENTE
Thomas Magedanz	Fraunhofer
Carlos Marques	PTInS
Carlos Parada	PTInS
Aiko Pras	UTWENTE
Simone Ruffino	TI
Alexandru Russu	Fraunhofer
Zarrar Yousaf	NEC
Zhongliang Zhao	UBERN

List of Peer Reviewers (in alphabetical order):

Luis Cordeiro	ONESource
Erik Schyler	UBERN

List of GA Reviewers (in alphabetical order):

Santiago Ruiz	STT

Versioning and contribution history

Version	Description	Contributors
0.1	Initial structure of the document	Morteza Karimzadeh
0.2	Intermediate complete version	Luuk Hendriks, et al.
0.3	Version for Peer Review	Luuk Hendriks, et al.
0.5	Version for GA Review	Luuk Hendriks, et al.
1.0	Final version	Luuk Hendriks, et al.

Table of Contents

Table of Figures

Table of Tables

List of Acronyms

ANDSF	Access Network Discovery and Selection Function
BL	Service Backend Logic
BW	Bandwidth
CC	Cloud Controller
DNS	Domain Name Server
DRA	Diameter Router Agent
eNB	Evolved NodeB
EPC	Evolved Packet Core
EPCaaS	EPC as a Service
GPRS	General Packet Radio System
GTP	GPRS Tunnelling Protocol
GTP-C	GTP Control Plane
GTP-U	GTP User Plane
HSS	Home Subscriber Server
ICCID	Integrated Circuit Card ID
ICN	Information Centric Network
ICNaaS	ICN as a Service
JSON	Java Script Object Notation
LB	Load Balancer
MaaS	Monitoring as a Service
MME	Mobility Management Entity
MOBaaS	Mobility and Bandwidth prediction as a Service
NAS	Non Access Stratum
NAT	Network Address Translation
OSD	Object Storage Device
PGW	Packet Data Network Gateway
PGWC	PGW Control
PGWU	PGW User Plane
SCTP	Session Control Transport Protocol
SFC	Service Functions Chaining
SGW	Serving Gateway
SGWC	SGW Control
SGWU	SGW User Plane
SIC	Service Instance Component
SO	Service Orchestrator
SM	Service Manager
SPR	Subscriber Profile Repository
SW	Switch
TAU	Tracking Area Update
TEID	Tunnel Endpoint Identifier
UE	User Equipment
VM	Virtual Machine
WS	Web Server Front-end

Executive Summary

This Deliverable (D4.5) performs an evaluation and validation of the services developed under the WP4 scope, namely, the Evolved Packet Core (EPCaaS), the Access Network Discovery and Selection Function (ANDSFaaS) and the Mobility and Bandwidth availability as a Service (MOBaaS).

The development of these services followed the architecture described in D4.1 and implemented the algorithms and mechanisms described in D4.3. The software that implement the services as well as the service control, as defined in the MCN Architecture described in D2.5, was initially described in D4.2 and documented as the final version in D4.4. The evaluation work has triggered further component improvements on the services; for this reason, this Deliverable describes the updates from D4.4.

The testing work presented in this document, intends to evaluate and validate the functionalities and performance of the implemented services, focusing on the following list of aspects (although not all apply to all services):

- Comparison of multiple user patterns/profiles
- Dynamic configuration of SICs (EPCaaS specific)
- Deployment and Disposal
- Scalability and Elasticity
- Reliability and High Availability

This Deliverable evaluates the services behaviour in a standalone environment, in order to validate the performance of each service alone. This evaluation task concludes the WP4 work. However, the WP4 services will be integrated in an end2end environment, in order to provide a full service and mobile operator capabilities on top of the cloud. This work will be performed under the scope of WP6, where the integrated overall MCN services will be evaluated and validated, and further documented on D6.4.

1 Introduction

This section describes the motivation, objectives and scope of this deliverable.

1.1 Motivation, Objectives and Scope

The objective of this deliverable is to present the updates regarding algorithms and implementation of services, developed in the scope of WP4 [1]. Further, this document describes testing scenarios and settings applied for performance evaluation and validation of services, as well as, the obtained results.

The algorithms and mechanisms for an efficient placement of the EPCaaS and ANDSFaaS service components have been presented in D4.3 [2]. It also covered the user mobility and link bandwidth availability prediction algorithms, as an important key issue to prepare services to adapt to foreseen future conditions, in the MOBaaS. Following D4.3, the N:2, 1:N EPCaaS implementation architectures, the ANDSF implementation for the 1:N architecture model and the MOBaaS service developments, its service orchestrator and its integration with other services within cloud environment were presented in D4.4.

This deliverable mainly presents the approach for validating the functionality of service components, developed in WP4 and further evaluation of the performance of the architectures and the algorithms implemented for the services, in a standalone environment. During the validation and evaluation phases, various functional and performance measures have been defined for each service. Each individual service has been examined in the specified test beds and scenarios in terms of defined measures.

Further validation / evaluation of WP4 services, integrated with relevant MCN services, will be done in WP6, which mainly focuses on integration of all MCN services in an end-to-end test scenario. The integration scenario, implementation test beds, interfaces between the different services and the obtained result will be presented in D6.4.

1.2 Structure of the Document

This deliverable is structured as follows. Section 2 presents EPCaaS internal architecture, EPCaaS' SO and its evaluation in terms of functional and performance measures. Section 3 evaluates the ANDSFaaS performance and the obtained results. In Section 4, MOBaaS testing environment, evaluation scenarios and the results are described. Following Section 4, run time fine-grained resource-aware network service management are presented in Section 5. Finally, Section 6 concludes the deliverable.

2 EPCaaS Evaluation

2.1 EPCaaS internal architecture

The EPCaaS internal architecture was described in detail in MCN D4.4 [1]. Here, we provide a brief summary of its OpenEPC-based implementation used for evaluation.

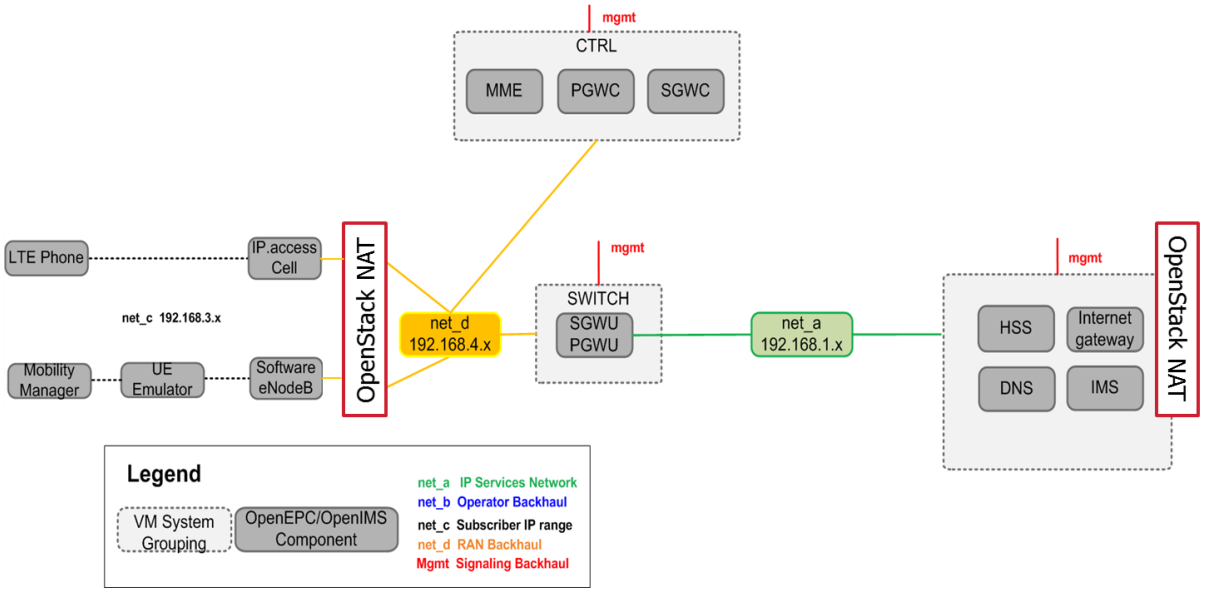

Figure 1 – N:2 Implementation Architecture

The current architecture is illustrated in Figure 1. It contains the following components:

- CTRL VM – includes the MME, SGWC and PGWC functions[1]. Although they run as a single program, for clarity of the presentation, they are still presented as three separate components;
- SWITCH – includes a single OpenFlow-based switch implementation, which is similar to a Local Gateway (having the functionality for both the SGWU and PGWU). Due to relative simplicity of OpenFlow protocol, it is easy to add more SWITCHes to increase the forwarding capacity of the EPCaaS. *Upon the evaluation, the SWITCH SIC was used for testing scaling-out and scaling-in capabilities of EPCaaS according to the number of users; a second SWITCH was created and disposed, dynamically*. Additional details on scaling-out/in mechanism are provided in sec. 2.2.3.
- Enablers – a component which includes the HSS and the Internet Gateway (including the NAT Gateway and the Firewall). The Enablers VM includes also, if needed, additional components such as DNS or IMS. It represents the termination of the EPC towards the applications and the Internet. *During the evaluation phase, the enablers were deployed as separate components, instead of an aggregate one. For example, the HSS was running on a VM together with an IMSaaS service instance.*
- Software eNB - UE Emulator – while preparing for the integration with Eurecom OpenAirInterface, the EPCaaS was tested with an eNB implementation currently in

[1] As specified by [[3GPP 23.401]]

development at Fraunhofer FOKUS. From the perspective of the EPCaaS, it behaves similarly to the physical LTE cell, although providing more flexibility such as the MME selection. *The UE Emulator, called also Benchmarking Tool, was used during evaluation phase to simulate an attachment and detachment for a large number of UEs, in such a way the scaling-out/in could be triggered.*

2.2 EPCaaS Service Orchestrator

This section describes the architecture and internal logic of the EPCaaS Service Orchestrator (SO)[2], used to perform the lifecycle management of OpenEPC entities, as described in the previous section. It complies with the general MCN SO architecture described in MCN D2.5 [3]:

- a SO Execution, which is responsible
    - for performing the actual service lifecycle operations, by calling the methods made available by the Cloud Controller and by interacting directly with the Service Instance Components (SICs) of OpenEPC, e.g. for provisioning or configuration
    - for interacting with other components, which enable the deployment of the service, e.g. the configuration module and the monitoring module, which act as interface towards the MaaS for the retrieval of the metrics needed by the SO Decision module to run its scaling algorithm
- a SO Decision entity, which is responsible for running the lifecycle management algorithm, in particular to take a decision to scale-out or scale-in.

2.2.1 Description of EPCaaS Classes

Figure 2 depicts the main classes of the EPCaaS SO.

[2] The EPCaaS SO was introduced in [[D4.4]]. This section provides a detailed description of the SO, which is needed to understand the evaluation process.

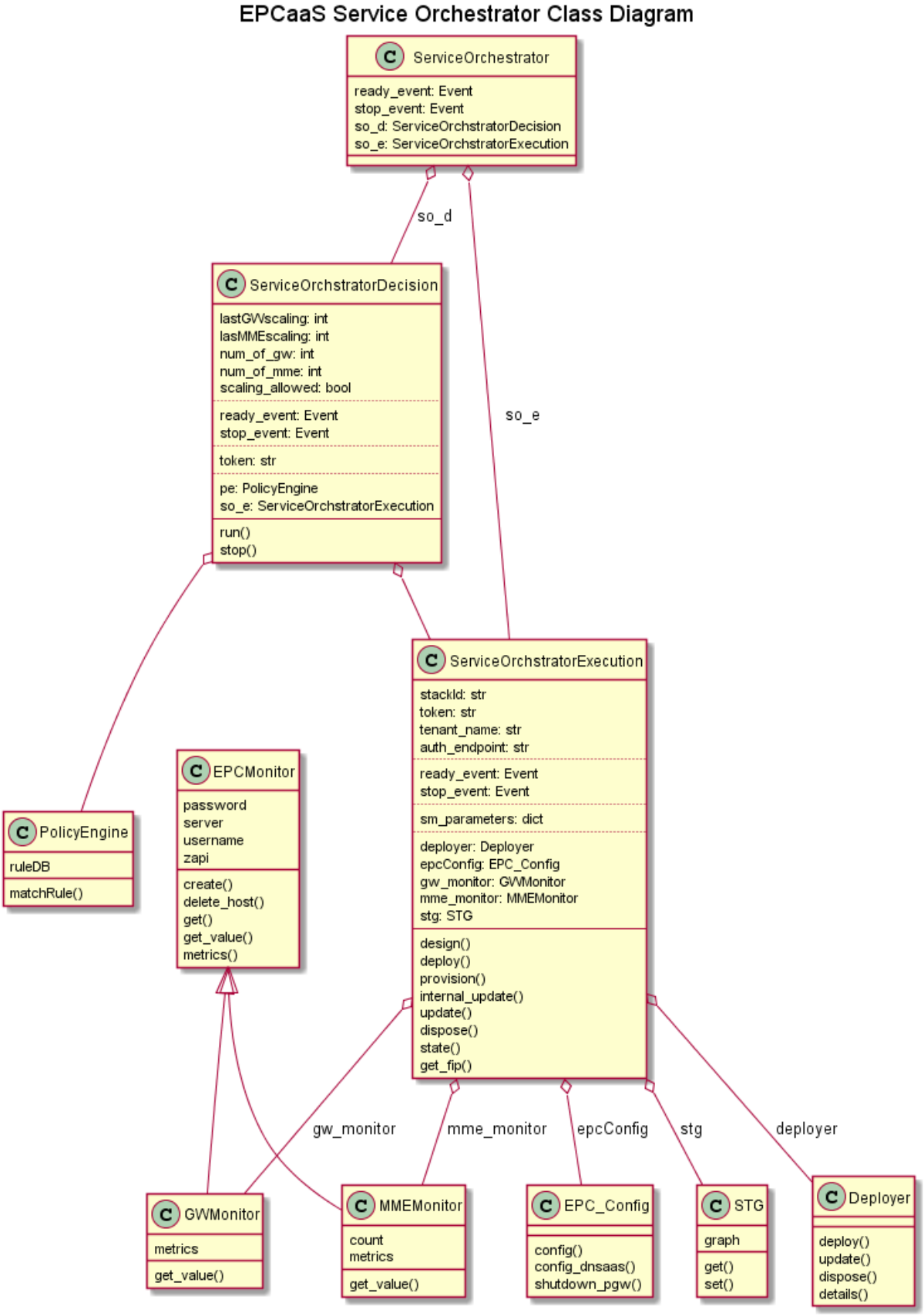

Figure 2 - EPCaaS Service Orchestrator class diagram

A brief description of each class is as follows:

- ServiceOrchestratorDecision: it contains the runtime management logic of the SO, to be activated upon the runtime management phase. It runs an infinite loop waiting for asynchronous events such as completion of provisioning phase or disposal of the service. It holds the state variables needed for scaling management (e.g. current number of deployed SWITCHes). During runtime phase, it gets metrics from SIC-specific EPCMonitor objects (handled by the SOExecution), queries the PolicyEngine class, which returns actions to be performed (e.g. scale-out, or None) and executes these actions by using ServiceOrchestratorExecution methods.
- PolicyEngine: it contains the actual decision logic, i.e. a set of rules, as in the exemplary table below:

Table 1 - example of ruleDB

Entity	Metric	Rule	Action
MME	Memory_usage	Memory_usage > TSO_ATTCH_USERS	SCALEOUT
GW	CPU_load	CPU_load < TSI_CPU_LOAD	SCALEIN
…	…		

The ruleDB columns have the following meaning:

- Entity is the SIC where the measurement of the Metric specified in the Metric column comes from,
- Metric is the metric measured through the SIC specified by the Entity column,
- Rule specifies activation conditions for the action specified in the Action column. Each rule simply compares one or more metrics to some thresholds (in the example and in the evaluation rules, only one metric per rule was used; the PolicyEngine provides flexibility to implement many complex rules),
- Action is a keyword, which indicates the action taken, when the rule in the Rule column is triggered; it is worth noting that in the current implementation, SODecision actually implements the scaling logic, i.e. knows the SIC to be created/deleted and the configuration to be changed on another SICs.

- ServiceOrchestratorExecution: it uses a Cloud Controller Deployer object to deploy or update the Service Template Graph (STG). It provides two EPCmonitor objects, one for the MME SIC (only one instance in our evaluation environment) and one for the SWITCH SIC, which can be used by the SODecision to retrieve the necessary metrics. Moreover it provides the EPCConfig object, which is used by the SODecision to perform configuration of the SICs, e.g. after a scale-in occurred. No decision is taken inside this class.
- EPCMonitor: it is an abstract class, which provides the basic methods to connect to MaaS and to retrieve all the metrics associated with a SIC. The service-specific monitors are actually implemented in the sub-classes, MMEMonitor and GWMonitor. They contain a list of metrics

to be monitored for that specific SIC (e.g. CPU_LOAD, ATTCH_USERS, NUM_SWITCHES from the MME), and a method to retrieve the measurements from the MaaS; i.e. they have the mapping between internal metric names (e.g. CPU_LOAD) and the name of the metrics configured on MaaS (e.g. "system.cpu.util[,idle]"). Multiple EPCMonitor objects can be instantiated, one for each SIC to be monitored.

- EPC_Config: it is used to configure the SICs in the provisioning phase and after an update occurs, e.g. after a scale-in. It also takes care of configuring support services, like DNSaaS. More details are provided in sec. 2.2.2.
- STG: it contains and manages the Service Template Graph in the form of a Heat template. The SODecision object creates the template and stores it in the STG object, which is used by the SOExecution, to actually update the Heat stack.

The code for EPCaaS resides in the MCN project GIT repository at [4]

2.2.2 Dynamic configuration of EPCaaS SICs

The EPC_config class (introduced in the previous section) takes care of the provisioning of EPCaaS SICs, i.e. configuration of the parameters needed for the correct activation of OpenEPC processes. Each SIC must be configured with a SIC-dependent set of parameters, some of which are only available at provisioning, i.e. only after the deployment phase.

Moreover, the provisioning step must also consider the dependencies between EPCaaS SICs, which need to be satisfied in a specific order. These dependencies are illustrated in Figure 3[3].

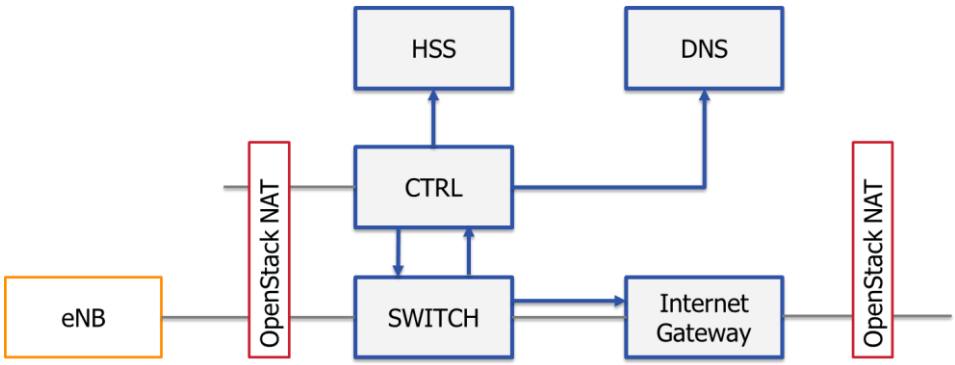

Figure 3 – EPCaaS N:2 Component Dependencies

The Table 2 provides an exemplary (small) subset of the parameters needed upon provisioning to setup dependencies between OpenEPC entities[4].

Table 2 - parameters needed for configuration

CTRL	DNS IP address

[3] An arrow from A to B means "A depends on B"

[4] Compared to the D4.2 version, a new dependency was added between the SWITCH and the Internet Gateway due to the forwarding chain. Additionally, a new dependency between the CTRL and the SWITCH was added due to the NAT limitations.

	DNS Domain configuration
	HSS name
	Public IP allocated to SWITCH
SWITCH	IP address of CTRL
	IP address of Internet GW

To cope with this requirement, all EPCaaS SICs are equipped with a convenient Configuration Module, which enables an external entity to send configuration commands by using a RESTful API. EPC_Config is being integrated with special purpose classes, called *"Configuration Adapters"*, which implement this API. This permits to dynamically configure all EPCaaS SICs at provisioning time, after all the other services (MCN and Support) have been deployed and the parameters are made available through the Service Manager(s)[5].

An example of an Adapter used to configure the CTRL SIC is described below:

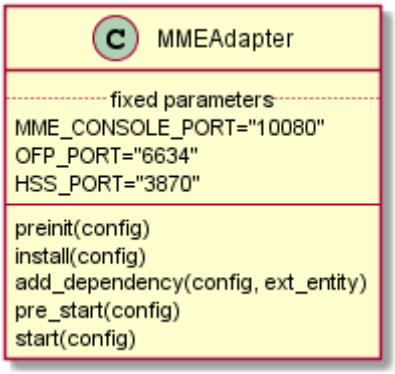

Figure 4 - MMEadapter class

The class methods (preinit, install, etc.) use the corresponding RESTful API available on each OpenEPC SIC. As an example, the http call to activate the install procedure on one mme-sgwc-pgwc SIC is:

```
curl -X POST -H "Content-Type:application/json" -d
"{\"parameters\":[\"$MME_MGMT_IP\",\"$MME_CONSOLE_PORT\",\"true,\"$MME_NET_
D_IP\",\"$OFP_PORT\",\"$OFP_PROTOCOL\"]}" http://127.0.0.1:8390/mme-sgw_c-
pgw_c-5G/install
```

Where MME_MGMT_IP, MME_CONSOLE_PORT etc. must be replaced with appropriate configuration parameters. It is worth noting that the Adapter methods have to be called in a precise order to correctly configure the SICs: this is the responsibility of the EPCConfig object[6].

[5] It is worth noting that without this feature, one should use other configuration mechanisms to configure VMs on OpenStack, which however seem to be more complicated and error-prone. Possible options include SSH to access VMs or heat SoftwareConfig resources [24], which enable us to change the configuration of a VM after it gets deployed.

[6] Figure 4 already lists the methods in the correct activation order.

2.2.3 Runtime Scaling logic

In the current implementation of the EPCaaS SO, a simple scaling policy has been implemented and evaluated. The scaling rules are included in the following table.

Table 3 - scaling policies implemented in EPCaaS SO

Entity	Metric	Rule	Action
MME	Num_Attached_Users	Num_Attached_Users > TSO_ATTCH_USERS	SCALEOUT
MME	Num_Attached_Users	Num_Attached_Users < TSI_ATTCH_USERS	SCALEIN

A **scaling-out** occurs when the number of attached users, read from CTRL SIC (i.e. mme-sgwc-pgwc) goes above the TSO_ATTCH_USERS threshold. When a scale-out is triggered, a new SWITCH SIC is created. Since SWITCH is a stateless component[7] only handles UE data traffic, it can be easily added and deleted, with no impact on the user's ongoing sessions, which store states in the mme-sgwc-pgwc component (CTRL). When a SWITCH is added to the running instance of OpenEPC, the total forwarding (i.e. traffic processing) capacity increases. *During evaluation, we tested scaling-out starting with one SWITCH and adding only one additional SWITCH.*

As detailed in D4.4 [1], the PGW-C functionality in the controller (CTRL) maintains different address spaces for each SWITCH, thus allowing continuous address allocation to UEs. However, the same rules are placed in all available SWITCHes; this ensures that the forwarding state information is available in all the data path components.

For the uplink (eNB → Internet direction, see also Figure 1), the traffic is routed through the OpenStack NAT (at this moment emulated as the NAT/Gateway in the enablers) at the entry of the system: this ensures the appropriate forwarding to the proper SWITCH. This NAT component is equivalent to a load balancer, which handles all the communication with external networks. Same for the downlink, the data traffic will pass through the NAT and then forwarded to the appropriate SWITCH. This mechanism allow both the scale-out and scale-in of the switches.

A **scaling-in** occurs when the number of attached users, read from CTRL SIC (i.e. mme-sgwc-pgwc) goes below the TSI_ATTCH_USERS threshold.

An additional control was implemented, which prevents Scaling-out/in to occur X seconds after the last scaling event. This control was inserted to avoid dangerous ping-pong effects with respect to the number of attached users. We use SCALING_GUARD_TIME parameter to control this delay.

For the evaluation, the following configuration for thresholds and parameters was used:

TSO_ATTCH_USERS	40.0
TSI_ATTCH_USERS	20.0
SCALING_GUARD_TIME	300

[7] For a more complete description of the functional split between CTRL and SWITCH, please refer to D4.3([2]) and D4.4 ([1])

2.3 EPCaaS Orchestration evaluation

This section contains results of the EPCaaS SO evaluation. It must be noted that the goal of this evaluation was not to evaluate performance of the EPCaaS itself. I.e. we did not mean to stress OpenEPC implementation to verify its scalability limits in terms of e.g. Simultaneously Attached Users (SAU) or maximal user throughput. Instead, to evaluate the SO, we performed two types of tests:

- Functional tests (sec. 2.3.2), aimed at verifying the correct implementation of
  - the runtime of the scaling-out/in logic,
  - the dynamic configuration modules,

  both implemented in the Service Orchestrator
- Performance tests (sec. 2.3.3), to measure a set of metrics, reported below, which are not related to the performance of the service itself, but rather to the performance of the environment, in which the service is deployed; they give an indication of reactiveness, we can reasonably expect from a "cloudified" service.

2.3.1 Environment for evaluation

2.3.1.1 Testbed description

The testbed is located at Zürich University of Applied Sciences in Winterthur/Switzerland. It is running OpenStack Juno. Resources available to test EPCaaS are:

40 Instances, 40 VCPUs, 40 GB of RAM, 20 Public IPs, 1TB Storage

2.3.1.2 Configuration of VMs

An instance of EPCaaS used upon the evaluation phase consisted of the following SICs

Table 4 - configuration of EPCaaS SICs, in terms of virtual resources

Hostname	Image	vCPU	vRAM	Disk
pgwu-sgwu-1	epc-base	1	2048 MB	20GB
pgwu-sgwu-2[8]	*epc-base*	1	2048 MB	20GB
mme-pgwc-sgwc	epc-base	1	2048 MB	20GB
dns	Dns	1	2048 MB	20GB
nat	epc-base	1	2048 MB	20GB

The network connectivity was configured as summarized in the following table:

[8] The second SWITCH is not included in the initial template deployed, but it is added after scale-out is triggered.

Table 5 - networking configuration for evaluation of EPCaaS

name	**Hostname**	**Mgmt**	**net_a**	**net_d**
SWITCH 1	pgwu-sgwu-1	192.168.9.45	172.19.5.45	172.19.8.45
SWITCH 2	*pgwu-sgwu-2*	192.168.9.120	172.19.5.4	172.19.8.4
CTRL	mme-pgwc-sgwc	192.168.9.47		172.19.8.47
DNS	Dns	192.168.9.49	172.19.5.49	
NAT	Nat	192.168.9.46	172.19.5.46	

2.3.2 Functional tests

This section contains the test descriptions and results of functional tests, used for verifying the basic functionalities of the runtime management phase of EPCaaS SO, namely the scaling-out/in logic and the dynamic configuration mechanism.

2.3.2.1 Test case#1: scaling out

Test Scenario	The EPCaaS is running; we check that a new SWITCH is created when the scaling-out conditions are met
SWITCH Scaling Rules	**IF** NUMBER_ATTCH_USERS > TSO_ATTCH_USERS **THEN** SCALE-OUT
Pre-condition	SWITCH – 1 VM CTRL – 1 VMs DNS – 1 VMs NAT – 1 VMs
Test Execution	Increase the number of users attached to the EPCaaS
Expected result	**SWITCH – 2 VM** CTRL – 1 VMs DNS – 1 VMs NAT – 1 VMs
Testing conditions	User attach rate: ca. 1 user/sec

2.3.2.2 Test case#2: scaling in

Test Scenario	After a scale-out occurred, the traffic decreases and the scale-in conditions are met; the previously added SWITCH is deleted
SWITCH Scaling Rules	**IF** NUMBER_ATTCH_USERS < TSI_ATTCH_USERS **THEN** SCALE-IN

Pre-condition	SWITCH – 2 VM CTRL – 1 VMs DNS – 1 VMs NAT – 1 VMs
Test Execution	Decrease the number of users attached to the EPCaaS
Expected result	**SWITCH – 1 VM** CTRL – 1 VMs DNS – 1 VMs NAT – 1 VMs
Testing conditions	User detach rate: ca. 1 user/sec

2.3.2.3 Test case #3: configuration of EPCaaS entities

Test Scenario	Configuration of EPCaaS entities after deployment or update (scale-out/in).
Pre-condition	SWITCH – 1 VM CTRL – 1 VMs DNS – 1 VMs NAT – 1 VMs All the VMs are not configured, meaning they are deployed from the same image
Test execution	Call the Configuration Adapter methods in the correct order and parameters
Expected result	SWITCH – 1 VM CTRL – 1 VMs DNS – 1 VMs NAT – 1 VMs All the VMs are correctly configured and specialized, i.e. they now host different SICs

2.3.2.4 Results of functional tests

2.3.2.4.1 SCALE-OUT/SCALE-IN

This section summarizes the results of the tests described in sec. 2.3.2.1 and 2.3.2.2. Table 6 contains an excerpt of the EPCaaS SO log file. The log contains a trace of the test run, aimed at triggering a scale-out, followed by a scale-in. In particular, during a scale-in, the dynamic configuration of a EPCaaS SIC occurs, namely the shutdown of a to-be-deleted SWITCH[9].

In particular, each line show:

- Timestamp of the event,

[9] This operation must be done to gracefully shutdown the SWITCH, and permit the CTRL to be notified of this event and take appropriate actions, e.g. by deleting the SWITCH from the list of the available SWITCHes and reclaim its addressing ranges.

- Three metrics: CPU load of the CTRL (labeled with '0'), the number of attached users ('1') and the number of active SWITCHES ('1'), as reported by the MaaS.

Some rows contain additional information, like

- state of the Heat stack, as provided by OpenStack, where detailed information of the running SIC can be found,
- an indication whether the scaling is allowed in this particular moment; it can be noted that, as explained above, after a scaling occurs, scaling is prohibited for SCALING_GUARD_TIME seconds (300 sec.)

As can be seen in the log, the scaling logic correctly triggers a scale-out when the MaaS reports a number of number of attached users higher than 40 (TSO_ATTCH_USERS) and correctly triggers a scale-in when the users go below 20 (TSI_ATTCH_USERS).

Moreover, it can be verified that the EPCconfig module actually performs a graceful shutdown of the SWITCH that will be deleted after the scale-in has been triggered.

Table 6 - example test-run of a scaling-out/in

```
...
[Thu Feb 12 11:16:44 2015]  {0: u'95.0167', 1: u'0', 3: u'1'}
[Thu Feb 12 11:16:44 2015]  Thread-1 \t DEBUG 02/12/2015 11:16:44 AM: \tSCALING ALLOWED: True
...
[Thu Feb 12 11:28:04 2015]  {0: u'99.0487', 1: u'1', 3: u'1'}
[Thu Feb 12 11:28:04 2015]  (u'UPDATE_COMPLETE', u'b4063f78-edd3-4178-9c8a-f73cf9b5aa42',
[{u'output_value': u'192.168.9.49', u'description': u'IP address of DNSaaS', u'output_key':
u'mcn.endpoint.dnsaas'}, {u'output_value': u'172.19.5.49', u'description': u'mgmt IP address
of dns', u'output_key': u'mcn.endpoint.dns'}, {u'output_value': u'172.19.5.45',
u'description': u'mgmt IP address of sgw_u', u'output_key': u'mcn.endpoint.pgwu-sgwu-1'},
{u'output_value': u'172.19.5.46', u'description': u'mgmt IP address of hss', u'output_key':
u'mcn.endpoint.nat'}, {u'output_value': u'160.85.4.45', u'description': u'IP address of
MaaS', u'output_key': u'mcn.endpoint.maas'}, {u'output_value': u'172.19.8.47',
u'description': u'mgmt IP address of mme-pgw_c-sgw_c', u'output_key': u'mcn.endpoint.mme-
pgwc-sgwc'}])
[Thu Feb 12 11:28:18 2015]  {0: u'98.5309', 1: u'1', 3: u'1'}
[Thu Feb 12 11:28:30 2015]  {0: u'97.3759', 1: u'3', 3: u'1'}
[Thu Feb 12 11:28:42 2015]  {0: u'96.4704', 1: u'2', 3: u'1'}
[Thu Feb 12 11:28:53 2015]  {0: u'95.5604', 1: u'2', 3: u'1'}
[Thu Feb 12 11:29:04 2015]  {0: u'94.5180', 1: u'2', 3: u'1'}
[Thu Feb 12 11:29:15 2015]  {0: u'92.7492', 1: u'2', 3: u'1'}
[Thu Feb 12 11:29:26 2015]  {0: u'91.6387', 1: u'2', 3: u'1'}
[Thu Feb 12 11:29:38 2015]  {0: u'90.9427', 1: u'2', 3: u'1'}
[Thu Feb 12 11:29:49 2015]  {0: u'90.1092', 1: u'6', 3: u'1'}
[Thu Feb 12 11:30:00 2015]  {0: u'89.5214', 1: u'16', 3: u'1'}
[Thu Feb 12 11:30:11 2015]  {0: u'89.4755', 1: u'27', 3: u'1'}
[Thu Feb 12 11:30:22 2015]  {0: u'89.4003', 1: u'33', 3: u'1'}
[Thu Feb 12 11:30:33 2015]  {0: u'89.6169', 1: u'40', 3: u'1'}
[Thu Feb 12 11:30:44 2015]  {0: u'89.8168', 1: u'46', 3: u'1'}
Scale-out THRESHOLD crossed
[Thu Feb 12 11:30:44 2015]  Thread-1 \t DEBUG 02/12/2015 11:30:44 AM: \tSCALING ALLOWED: True
[Thu Feb 12 11:30:44 2015]  Thread-1 \t DEBUG 02/12/2015 11:30:44 AM: \tExecuting internal
update logic
[Thu Feb 12 11:30:55 2015]  Thread-1 \t DEBUG 02/12/2015 11:30:55 AM: \tState:
UPDATE_IN_PROGRESS
Service update started
[Thu Feb 12 11:30:56 2015]  {0: u'89.8456', 1: u'57', 3: u'1'}
[Thu Feb 12 11:31:06 2015]  {0: u'89.6124', 1: u'65', 3: u'1'}
[Thu Feb 12 11:31:17 2015]  {0: u'89.4914', 1: u'69', 3: u'1'}
[Thu Feb 12 11:31:27 2015]  {0: u'89.1275', 1: u'78', 3: u'1'}
[Thu Feb 12 11:31:38 2015]  {0: u'88.9877', 1: u'87', 3: u'1'}
[Thu Feb 12 11:31:48 2015]  {0: u'87.3637', 1: u'91', 3: u'1'}
[Thu Feb 12 11:31:59 2015]  {0: u'87.1179', 1: u'100', 3: u'1'}
[Thu Feb 12 11:32:09 2015]  {0: u'76.6806', 1: u'101', 3: u'1'}
```

```
[Thu Feb 12 11:32:19 2015]  {0: u'66.7951', 1: u'100', 3: u'1'}
[Thu Feb 12 11:32:30 2015]  {0: u'61.1260', 1: u'97', 3: u'1'}
[Thu Feb 12 11:32:41 2015]    Thread-1 \t DEBUG 02/12/2015 11:32:41 AM: \tState:
UPDATE_COMPLETE
[Thu Feb 12 11:32:41 2015]   (u'UPDATE_COMPLETE', u'b4063f78-edd3-4178-9c8a-f73cf9b5aa42',
[{u'output_value': u'192.168.9.49', u'description': u'IP address of DNSaaS', u'output_key':
u'mcn.endpoint.dnsaas'}, {u'output_value': u'172.19.5.49', u'description': u'mgmt IP address
of dns', u'output_key': u'mcn.endpoint.dns'}, {u'output_value': u'172.19.5.45',
u'description': u'mgmt IP address of sgw_u', u'output_key': u'mcn.endpoint.pgwu-sgwu-1'},
{u'output_value': u'160.85.4.56', u'description': u'mgmt IP address of sgw_u 2',
u'output_key': u'mcn.endpoint.pgwu-sgwu-2'}, {u'output_value': u'172.19.5.46',
u'description': u'mgmt IP address of hss', u'output_key': u'mcn.endpoint.nat'},
{u'output_value': u'160.85.4.45', u'description': u'IP address of MaaS', u'output_key':
u'mcn.endpoint.maas'}, {u'output_value': u'172.19.8.47', u'description': u'mgmt IP address of
mme-pgw_c-sgw_c', u'output_key': u'mcn.endpoint.mme-pgwc-sgwc'}])
```

Service update completed – a new SWITCH is available

```
[Thu Feb 12 11:32:41 2015]  {0: u'60.3183', 1: u'94', 3: u'1'}
[Thu Feb 12 11:32:41 2015]   Thread-1 \t DEBUG 02/12/2015 11:32:41 AM: \tSCALING ALLOWED:
False
[Thu Feb 12 11:32:52 2015]  {0: u'62.3492', 1: u'89', 3: u'1'}
[Thu Feb 12 11:32:52 2015]  Thread-1 \t DEBUG 02/12/2015 11:32:52 AM: \tSCALING ALLOWED: True
[Thu Feb 12 11:33:03 2015]  {0: u'65.4375', 1: u'84', 3: u'1'}
[Thu Feb 12 11:33:15 2015]  {0: u'75.5537', 1: u'72', 3: u'1'}
[Thu Feb 12 11:33:26 2015]  {0: u'87.1769', 1: u'68', 3: u'1'}
[Thu Feb 12 11:33:37 2015]  {0: u'88.1484', 1: u'60', 3: u'1'}
[Thu Feb 12 11:33:48 2015]  {0: u'89.2599', 1: u'52', 3: u'1'}
[Thu Feb 12 11:34:00 2015]  {0: u'89.2653', 1: u'46', 3: u'2'}
[Thu Feb 12 11:34:11 2015]  {0: u'90.1427', 1: u'36', 3: u'2'}
[Thu Feb 12 11:34:22 2015]  {0: u'90.9686', 1: u'29', 3: u'2'}
[Thu Feb 12 11:34:34 2015]  {0: u'91.4080', 1: u'22', 3: u'2'}
[Thu Feb 12 11:34:46 2015]  {0: u'91.4626', 1: u'9', 3: u'2'}
```

Scale-in THRESHOLD crossed

```
[Thu Feb 12 11:34:46 2015]  Thread-1 \t DEBUG 02/12/2015 11:34:46 AM: \tSHUTTING DOWN PGW2
```

Dynamic re-configuration of SWITCH

```
[Thu Feb 12 11:34:47 2015]  PGW2 IP: 160.85.4.56
[Thu Feb 12 11:34:50 2015]  [u'sgwu-pgwu stop/waiting\\n']
[Thu Feb 12 11:34:50 2015]   Thread-1 \t DEBUG 02/12/2015 11:34:50 AM: \tExecuting internal
update logic
[Thu Feb 12 11:35:02 2015]    Thread-1 \t DEBUG 02/12/2015 11:35:02 AM: \tState:
UPDATE_IN_PROGRESS
[Thu Feb 12 11:35:02 2015]  {0: u'91.2490', 1: u'1', 3: u'1'}
[Thu Feb 12 11:35:02 2015]   Thread-1 \t DEBUG 02/12/2015 11:35:02 AM: \tSCALING ALLOWED:
False
[Thu Feb 12 11:35:13 2015]    Thread-1 \t DEBUG 02/12/2015 11:35:13 AM: \tState:
UPDATE_COMPLETE
[Thu Feb 12 11:35:13 2015]   (u'UPDATE_COMPLETE', u'b4063f78-edd3-4178-9c8a-f73cf9b5aa42',
[{u'output_value': u'192.168.9.49', u'description': u'IP address of DNSaaS', u'output_key':
u'mcn.endpoint.dnsaas'}, {u'output_value': u'172.19.5.49', u'description': u'mgmt IP address
of dns', u'output_key': u'mcn.endpoint.dns'}, {u'output_value': u'172.19.5.45',
u'description': u'mgmt IP address of sgw_u', u'output_key': u'mcn.endpoint.pgwu-sgwu-1'},
{u'output_value': u'172.19.5.46', u'description': u'mgmt IP address of hss', u'output_key':
u'mcn.endpoint.nat'}, {u'output_value': u'160.85.4.45', u'description': u'IP address of
MaaS', u'output_key': u'mcn.endpoint.maas'}, {u'output_value': u'172.19.8.47',
u'description': u'mgmt IP address of mme-pgw_c-sgw_c', u'output_key': u'mcn.endpoint.mme-
pgwc-sgwc'}])
```

Service update completed – a SWITCH has been removed

```
[Thu Feb 12 11:35:13 2015]  {0: u'90.9289', 1: u'1', 3: u'1'}
[Thu Feb 12 11:35:24 2015]  {0: u'90.9289', 1: u'2', 3: u'1'}
...
...
```

Figure 5 illustrates the test run of the previously attached log file. The two red indicators show the moments, in which scale-out and scale-in are triggered. The purple line show the actual number of SWITCHes (taken from the OpenStack heat stack state), while the green line shows the number of SWITCHes available on the MaaS.

As can be observed, there is a delay between the actual instantiation of the SWITCH and the moment when its metrics become available on the MaaS. This is due to the time needed for a new SWITCH

VM to actually boot up and the time required by the MaaS server to read the metrics and make them available.

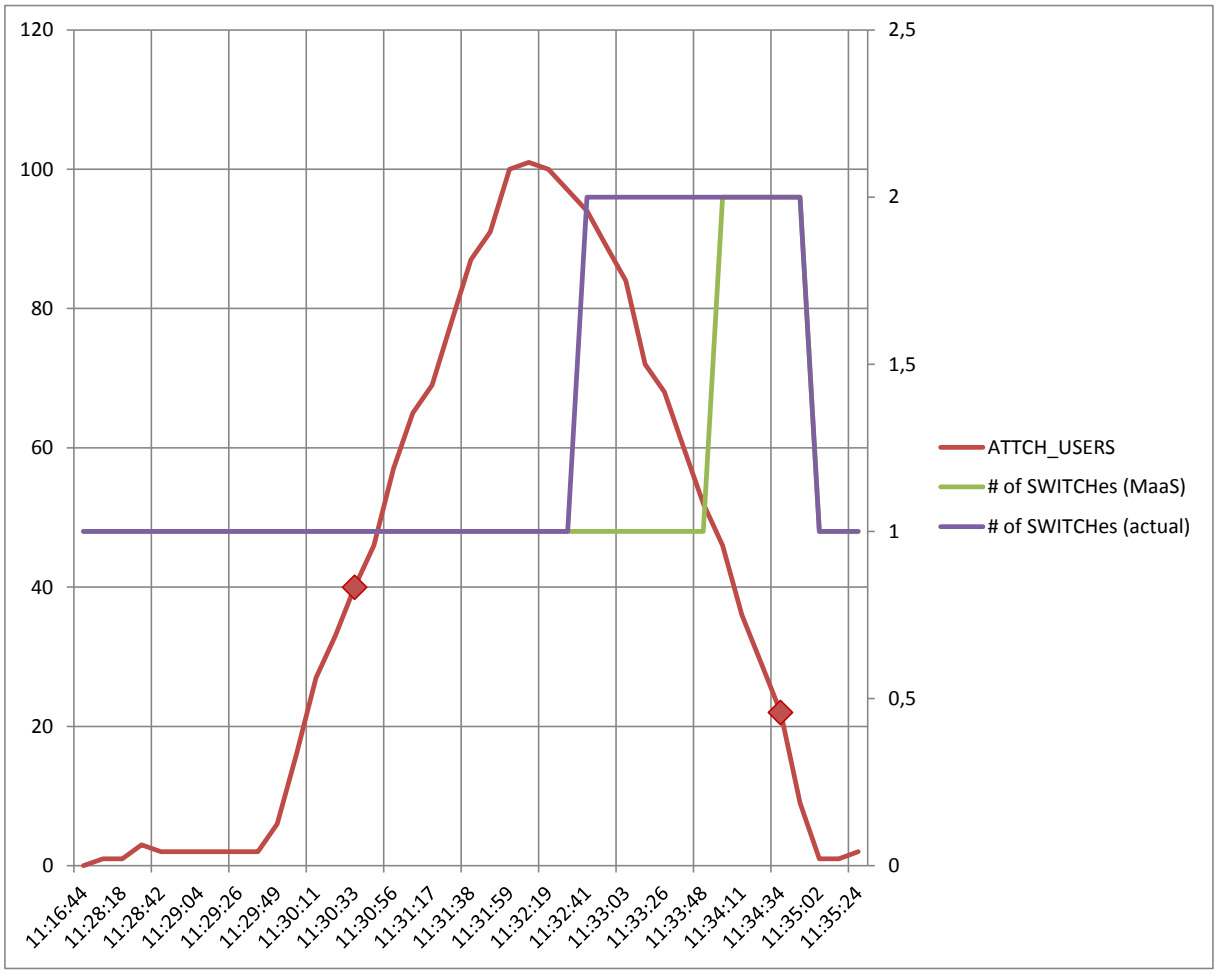

Figure 5 - example of scale-out/in procedure (left axis-users, right axis-#of Switches)

2.3.2.4.2 CONFIGURATION

This section summarizes the results of the tests described in sec.2.3.2.3. An example of the proper Configuration Adapter operation is illustrated in Table 8. We report the result of the configuration method call (namely the MMEAdapter() "install" method). As one can observe, when the method calls the corresponding API, a set of configuration operations (local configurations, package downloading etc.) is performed automatically on the SIC. The successful configuration result is reported at the bottom of the table ([200]).

Table 7 - Example of correct configuration of an MME-SGWc-PGWc SIC

```
I'm the mme-pgwc-sgwc adapter, install mme-pgwc-sgwc service, parameters ['192.168.85.47',
'10080', 'true', '172.19.8.47', '6634', 'tcp']
+ mgmt_addr_file=mgmt
+ net_d_addr_file=net_d
+ '[' 6 -ne 6 ']'
+ getopts h OPTION
+ OwnIPv4_ADDR=192.168.85.47
+ mme_console_port=10080
+ upstart_on=true
+ Own_net_d_IPv4_ADDR=172.19.8.47
```

```
+ ofp_port=6634
+ ofp_transport=tcp
+ NETD_FLOATING_NETWORK_CIDR=
+ sgw_c_s11s4_ip_addr=127.0.0.2
+ echo 192.168.85.47
+ echo 172.19.8.47
+ '[' '!' -d /opt/Open5GCore ']'
++ sed 's/\([^:]*\):/\1/'
++ awk '{ print $2 }'
++ head -1
++ ip addr show to
Command line is not complete. Try option "help"
+ IPv4_DEV=
++ sed -e 's/\/.*$//'
++ awk '-F ' '{print $2}'
++ grep inet6
++ ip addr show dev
Command line is not complete. Try option "help"
+ IPv6_ADDR=
+ echo 'using network interface '
using network interface
+ create_internal_networking
+ ip addr add 127.0.0.2/8 dev lo
RTNETLINK answers: File exists
+ ip addr add ::2 dev lo
RTNETLINK answers: File exists
+ ip addr add 127.0.0.3/8 dev lo
RTNETLINK answers: File exists
+ ip addr add ::3 dev lo
RTNETLINK answers: File exists
+ NETD_FLOATING_NETWORK_CIDR=
+ '[' 6 -eq 7 ']'
+ mme/install 192.168.85.47 127.0.0.2 mme 10080 true 6634 tcp
Installing packages
Ign http://security.ubuntu.com trusty-security InRelease
Ign http://ppa.launchpad.net trusty InRelease
Get:1 http://security.ubuntu.com trusty-security Release.gpg [933 B]
Hit http://ppa.launchpad.net trusty Release.gpg
Get:2 http://security.ubuntu.com trusty-security Release [63.5 kB]
Hit http://ppa.launchpad.net trusty Release
Hit http://ppa.launchpad.net trusty/main amd64 Packages
Hit http://ppa.launchpad.net trusty/main Translation-en
Get:3 http://security.ubuntu.com trusty-security/main Sources [83.9 kB]
Ign http://nova.clouds.archive.ubuntu.com trusty InRelease
Get:4 http://security.ubuntu.com trusty-security/universe Sources [25.2 kB]
Get:5 http://security.ubuntu.com trusty-security/main amd64 Packages [284 kB]
Ign http://nova.clouds.archive.ubuntu.com trusty-updates InRelease
Get:6 http://security.ubuntu.com trusty-security/universe amd64 Packages [107 kB]
Hit http://nova.clouds.archive.ubuntu.com trusty Release.gpg
Hit http://security.ubuntu.com trusty-security/main Translation-en
Hit http://security.ubuntu.com trusty-security/universe Translation-en
Get:7 http://nova.clouds.archive.ubuntu.com trusty-updates Release.gpg [933 B]
Hit http://nova.clouds.archive.ubuntu.com trusty Release
Get:8 http://nova.clouds.archive.ubuntu.com trusty-updates Release [63.5 kB]
Hit http://nova.clouds.archive.ubuntu.com trusty/main Sources
Hit http://nova.clouds.archive.ubuntu.com trusty/universe Sources
Hit http://nova.clouds.archive.ubuntu.com trusty/main amd64 Packages
Hit http://nova.clouds.archive.ubuntu.com trusty/universe amd64 Packages
Hit http://nova.clouds.archive.ubuntu.com trusty/main Translation-en
Hit http://nova.clouds.archive.ubuntu.com trusty/universe Translation-en
Get:9 http://nova.clouds.archive.ubuntu.com trusty-updates/main Sources [207 kB]
Get:10 http://nova.clouds.archive.ubuntu.com trusty-updates/universe Sources [118 kB]
Get:11 http://nova.clouds.archive.ubuntu.com trusty-updates/main amd64 Packages [532 kB]
Get:12 http://nova.clouds.archive.ubuntu.com trusty-updates/universe amd64 Packages [283 kB]
Hit http://nova.clouds.archive.ubuntu.com trusty-updates/main Translation-en
Hit http://nova.clouds.archive.ubuntu.com trusty-updates/universe Translation-en
Fetched 1770 kB in 4s (403 kB/s)
Reading package lists...
Reading package lists...
Building dependency tree...
Reading state information...
libxslt1.1 is already the newest version.
python-lxml is already the newest version.
python-requests is already the newest version.
```

```
0 upgraded, 0 newly installed, 0 to remove and 183 not upgraded.
creating database for MME
Setting Service Upstart mme on
+ sgw_c/install
Installing packages
Ign http://nova.clouds.archive.ubuntu.com trusty InRelease
Ign http://ppa.launchpad.net trusty InRelease
Ign http://nova.clouds.archive.ubuntu.com trusty-updates InRelease
Hit http://ppa.launchpad.net trusty Release.gpg
Hit http://nova.clouds.archive.ubuntu.com trusty Release.gpg
Hit http://ppa.launchpad.net trusty Release
Hit http://nova.clouds.archive.ubuntu.com trusty-updates Release.gpg
Hit http://ppa.launchpad.net trusty/main amd64 Packages
Hit http://nova.clouds.archive.ubuntu.com trusty Release
Hit http://nova.clouds.archive.ubuntu.com trusty-updates Release
Hit http://ppa.launchpad.net trusty/main Translation-en
Hit http://nova.clouds.archive.ubuntu.com trusty/main Sources
Ign http://security.ubuntu.com trusty-security InRelease
Hit http://nova.clouds.archive.ubuntu.com trusty/universe Sources
Hit http://nova.clouds.archive.ubuntu.com trusty/main amd64 Packages
Hit http://nova.clouds.archive.ubuntu.com trusty/universe amd64 Packages
Hit http://nova.clouds.archive.ubuntu.com trusty/main Translation-en
Hit http://nova.clouds.archive.ubuntu.com trusty/universe Translation-en
Hit http://security.ubuntu.com trusty-security Release.gpg
Hit http://nova.clouds.archive.ubuntu.com trusty-updates/main Sources
Hit http://nova.clouds.archive.ubuntu.com trusty-updates/universe Sources
Hit http://nova.clouds.archive.ubuntu.com trusty-updates/main amd64 Packages
Hit http://nova.clouds.archive.ubuntu.com trusty-updates/universe amd64 Packages
Hit http://security.ubuntu.com trusty-security Release
Hit http://nova.clouds.archive.ubuntu.com trusty-updates/main Translation-en
Hit http://nova.clouds.archive.ubuntu.com trusty-updates/universe Translation-en
Hit http://security.ubuntu.com trusty-security/main Sources
Hit http://security.ubuntu.com trusty-security/universe Sources
Hit http://security.ubuntu.com trusty-security/main amd64 Packages
Hit http://security.ubuntu.com trusty-security/universe amd64 Packages
Hit http://security.ubuntu.com trusty-security/main Translation-en
Hit http://security.ubuntu.com trusty-security/universe Translation-en
Reading package lists...
Reading package lists...
Building dependency tree...
Reading state information...
libxslt1.1 is already the newest version.
python-lxml is already the newest version.
python-requests is already the newest version.
0 upgraded, 0 newly installed, 0 to remove and 183 not upgraded.
creating local SGW-C database
+ pgw_c/install
Installing packages
Ign http://ppa.launchpad.net trusty InRelease
Hit http://ppa.launchpad.net trusty Release.gpg
Hit http://ppa.launchpad.net trusty Release
Hit http://ppa.launchpad.net trusty/main amd64 Packages
Hit http://ppa.launchpad.net trusty/main Translation-en
Ign http://security.ubuntu.com trusty-security InRelease
Hit http://security.ubuntu.com trusty-security Release.gpg
Hit http://security.ubuntu.com trusty-security Release
Hit http://security.ubuntu.com trusty-security/main Sources
Hit http://security.ubuntu.com trusty-security/universe Sources
Hit http://security.ubuntu.com trusty-security/main amd64 Packages
Hit http://security.ubuntu.com trusty-security/universe amd64 Packages
Hit http://security.ubuntu.com trusty-security/main Translation-en
Hit http://security.ubuntu.com trusty-security/universe Translation-en
Ign http://nova.clouds.archive.ubuntu.com trusty InRelease
Ign http://nova.clouds.archive.ubuntu.com trusty-updates InRelease
Hit http://nova.clouds.archive.ubuntu.com trusty Release.gpg
Hit http://nova.clouds.archive.ubuntu.com trusty-updates Release.gpg
Hit http://nova.clouds.archive.ubuntu.com trusty Release
Hit http://nova.clouds.archive.ubuntu.com trusty-updates Release
Hit http://nova.clouds.archive.ubuntu.com trusty/main Sources
Hit http://nova.clouds.archive.ubuntu.com trusty/universe Sources
Hit http://nova.clouds.archive.ubuntu.com trusty/main amd64 Packages
Hit http://nova.clouds.archive.ubuntu.com trusty/universe amd64 Packages
Hit http://nova.clouds.archive.ubuntu.com trusty/main Translation-en
Hit http://nova.clouds.archive.ubuntu.com trusty/universe Translation-en
```

```
Hit http://nova.clouds.archive.ubuntu.com trusty-updates/main Sources
Hit http://nova.clouds.archive.ubuntu.com trusty-updates/universe Sources
Hit http://nova.clouds.archive.ubuntu.com trusty-updates/main amd64 Packages
Hit http://nova.clouds.archive.ubuntu.com trusty-updates/universe amd64 Packages
Hit http://nova.clouds.archive.ubuntu.com trusty-updates/main Translation-en
Hit http://nova.clouds.archive.ubuntu.com trusty-updates/universe Translation-en
Reading package lists...
Reading package lists...
Building dependency tree...
Reading state information...
libxslt1.1 is already the newest version.
python-lxml is already the newest version.
python-requests is already the newest version.
0 upgraded, 0 newly installed, 0 to remove and 183 not upgraded.
creating local PGW-C database

I'm the mme-pgwc-sgwc adapter, installing mme-pgwc-sgwc service, received resp <Response
[200]>
```

2.3.3 Performance tests

This section contains the description and the results of performance tests. It is worth noting, the objective of performance tests was to measure a set of metrics, which are not related to the performance of the service itself, but to the performance of the environment, in which the service is deployed; they give an indication on reactiveness that we can expect from a "cloudified" service.

2.3.3.1 Metrics for the performance evaluation

The following table contains the metrics used to evaluate the EPCaaS SO. In particular, scale-out and scale-in latencies have a direct impact on the overall service quality: the service must know how much time the creation of a new VM requires, in order to adapt the logic accordingly.

Table 8 - metrics used for the performance evaluation of EPCaaS SO

Deploy latency	The time for the complete deployment of an EPCaaS instance, i.e. all the VMs needed by OpenEPC
Dispose latency	The time for a complete disposal of an EPCaaS instance
Scale-out latency	The time for adding one SWITCH to a running instance of EPCaaS
Scale-in latency	The time for removing one SWITCH from a running instance of EPCaaS

As an additional note, it can be observed that, though these metrics are relatively simple, they are very interesting from the Operator's perspective, because they can be compared to operations (e.g. deploy a network function, adding capacity to a running network function etc.), which nowadays are executed manually and completed in days/weeks.

2.3.3.2 Results of performance tests

2.3.3.2.1 DEPLOYMENT LATENCY

The following table contains the deployment time statistics. In particular:

- in the first column, the time needed for an EPCaaS heat template to be completely deployed on OpenStack with pre-configured virtual networks and routers.
- in the second column, the time needed for an EPCaaS heat template to be completely deployed on OpenStack, with no network pre-configuration, i.e. the heat template includes the creation of virtual network resources.

We present this comparison, because it can be observed that the latency when a network is created increases ca. 24% in comparison to the situation when networks are pre-configured.

Table 9 - Deploy latencies [s]

	EPCaaS (4 VMs) - network pre-created	**EPCaaS (4 VMs) – w/ network creation**
Mean	**22,80**	**28,40**
SD[10]	1,13	1,39
Min	21,09	26,59
Q1[11]	22,08	27,47
Median	22,41	28,18
Q3[12]	23,29	28,95
Max	24,94	32,81

The following graph summarizes the statistics.

[10] Standard Deviation

[11] First Quartile

[12] Third Quartile

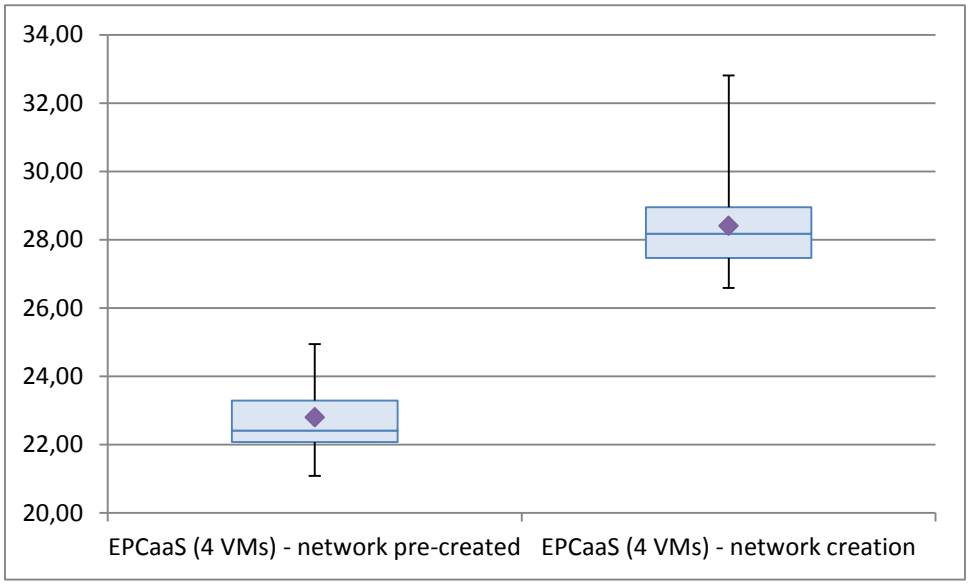

Figure 6 – Deploy latencies [s]

Overall, the statistics show that the deployment times in all cases are far below 1 minute, which is indeed, a very promising result.

2.3.3.2.2 DISPOSE LATENCY

The following tables and graph present disposal latencies of the EPCaaS again in two cases: when virtual network components have to and do not have to be deleted. The difference in latencies is more evident than in the deployment case (ca. 58% more time is required in the second case), but in absolute values both creation and deletion times of virtual network resources are around 5-6 seconds.

Table 10 - Dispose latencies [s]

	EPCaaS (4 VMs) - network pre-created	**EPCaaS (4 VMs) - network creation**
Mean	**8,38**	**13,31**
SD	0,38	0,60
Min	7,88	11,96
Q1	8,18	13,05
Median	8,33	13,43
Q3	8,55	13,55
Max	9,63	14,33

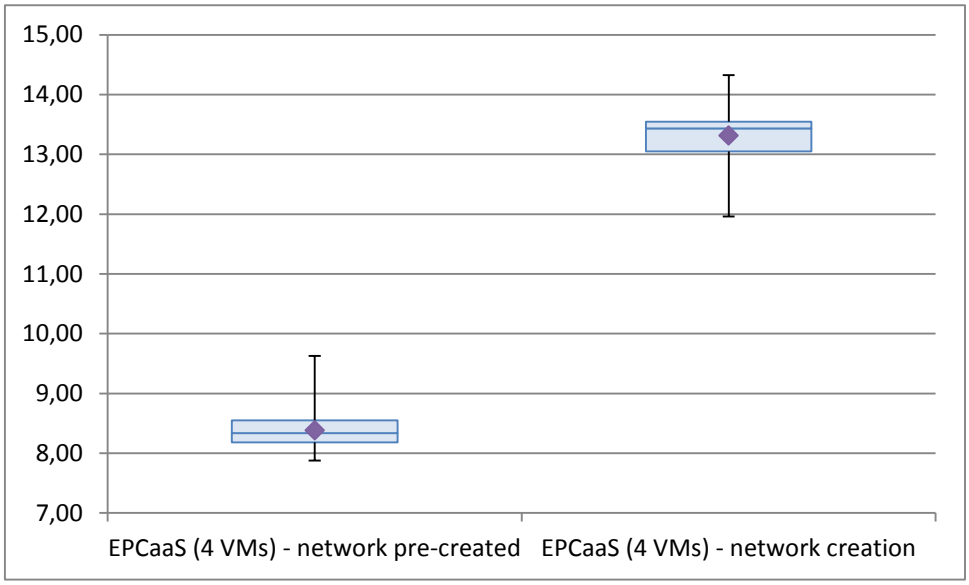

Figure 7 – Dispose latencies [s]

2.3.3.2.3 SCALING-OUT/IN LATENCY

In the following table and graphs, the statistics for scaling-out/in latencies are reported. The figures illustrate the time needed to deploy an additional SWITCH SIC by updating the running heat template with a new one.

Note that these measurements do NOT include any virtual network creation, so they account for the actual creation of additional SICs and the update of running heat templates.

Table 11 - Scaling performances [s]

	1 → 2 SWITCHes (scale-out)	**2 → 1 SWITCHes (scale-in)**
Mean	**25,89**	**18,44**
SD	2,90	0,76
Min	22,85	17,50
Q1	24,55	17,69
Median	25,48	18,22
Q3	26,34	19,15
Max	37,27	19,57

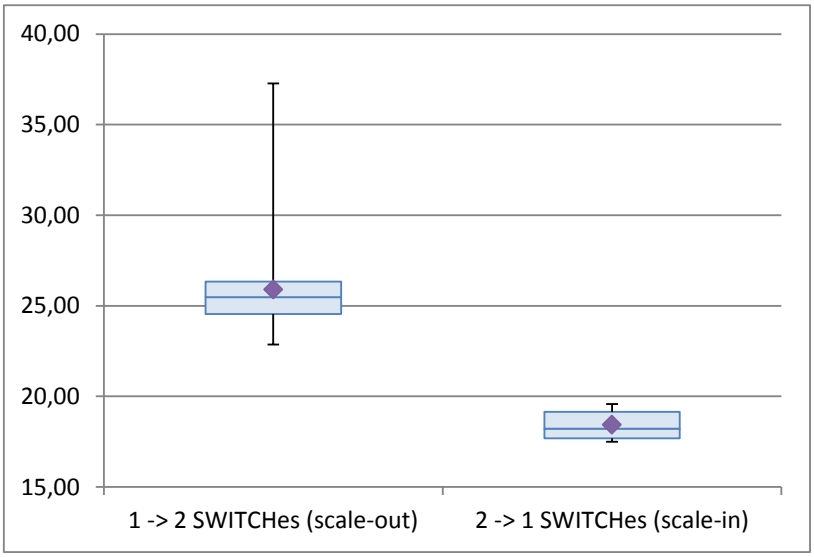

Figure 8 – Scaling out/in latencies [s]

It can be observed (from the box on the left) that the time needed for adding a new SWITCH VM to the running EPCaaS instance has some "spike", which depends on the underlying test-bed load at the moment of scaling.

2.4 Auto-Evaluation/Planning Tool for EPCaaS Deployment

2.4.1 Background

In D4.3 [2], we presented a detailed cost study on the initial deployment strategies for the SICs provisioning EPCaaS in a large scale 3-tier DC. The deployment cost was measured in terms of the Data Centre Infrastructure (DCI) resource utilization. Here, we focus on a detailed analytical mode developed for NEC's vEPC system [5] capturing the interfaces between various SICs and the (anti)affinity between the constituent SICs. Two deployment strategies, referred to as VSD and HSD in D4.3, were tested in terms of the impact on the DC resource utilization cost. A valuable insight provided by the results motivated us towards extending the study into developing a practical auto-evaluation/planning tool. It will enable a user (i.e., DC operator and/or EPCaaS provider) to perform quick DevOps style pre-deployment evaluation of the impact of the deployment strategy on the DC resources when deploying the SICs providing EPCaaS. This tool will enable the user to input relevant parameters (including workload profiles) and quickly provide the graphical view of the output results. The user will thus be able to perform a quick series of tests with different configurations and get a visual overview of the cost incurred in terms of DC resource consumption. Therefore, the user will be able to better perform DC resource planning and management. Due to its modular design, the developed tool is scalable and can be extended towards other nature of network services.

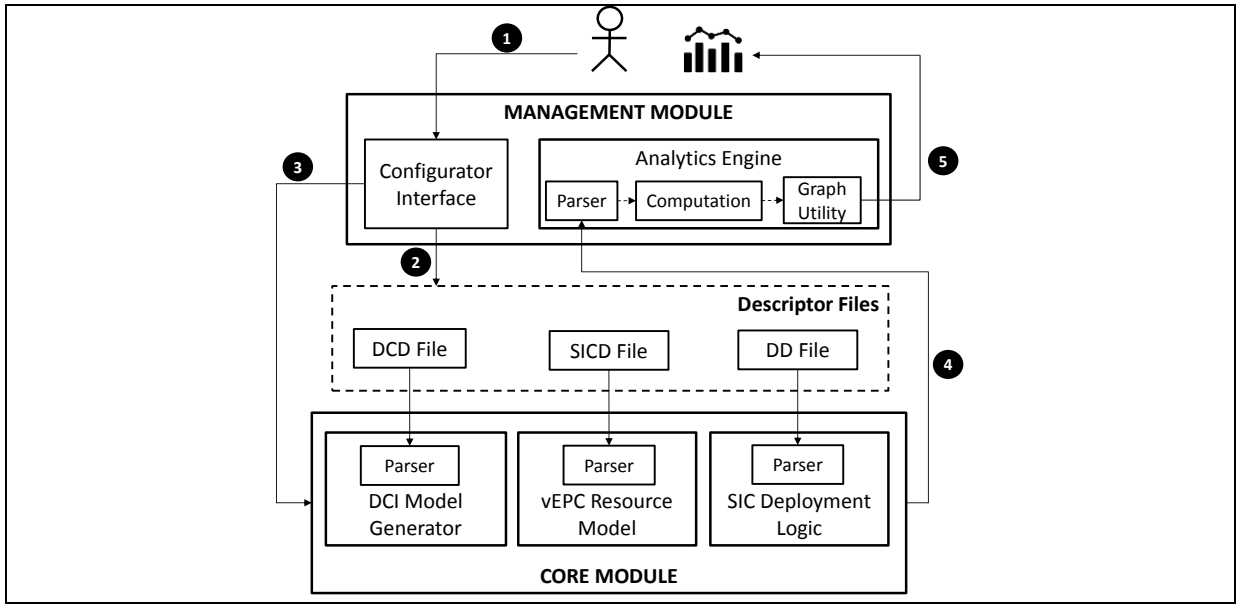

Figure 9 - Functional Overview of the Auto-evaluation/Planning tool for EPCaaS Deployment

2.4.2 Functional Overview

Figure 9 shows a functional overview of the auto-evaluation/planning tool of a modular design. It consists of two main modules namely the "Core module" and the "Management module". Each module is composed of multiple interacting functional blocks. The functional blocks inside the Core module are developed in C++, while those within the Management module are developed using Python.

As labelled in Figure 9, following the main sequence of operation:

1. The user will provide values for relevant parameters via the "Configurator interface".

2. The "configurator interface" will write these values into the relevant "Descriptor files", where each Descriptor file is associated with a relevant functional block of the Core module.

3. The management module will then send an execute command towards the main function of the Core module.

4. The core module will provide the output results composed of multiple metrics, which are sent towards the "Analytics Engine" within the "Management Module".

5. Based on the various performance metrics, the "Analytics Engine" will perform necessary analytics to derive useful information, which are then provided as a graphical output to the user.

The details of the Core/Management module along with their respective functional blocks are described below.

2.4.3 Core Module

The core module is composed of the following three sub-modules:

1. DC Infrastructure (DCI) Model Generator

2. vEPC resource model

3. SIC deployment logic

The above sub-modules are developed in C++ and each sub-module is associated with a configuration/descriptor file that enables a user to set various configuration parameters via the Management module. Each sub-module also has a parser for reading parametric values. A brief description of the above three sub-modules is given below.

2.4.3.1 DC Infrastructure (DCI) Model Generator:

This module embodies the logic of creating a DCI model based on user specifications. The DC topology and its resources (i.e., compute, network, and memory/storage) is specified by the user via the configuration files referred to as "DC Descriptor" (DCD) file. The DCD file specifies the following key user configurable parameters:

- DC topology type (`dc_type`)
- Number of aggregation switches (`n_aggr_switches`)
- Number of core switches (`n_core_switches`)
- Number of racks (`n_racks`)
- Number of servers per racks (`rack_capacity`)
- Number of cores per server (`n_cores_per_server`)
- Amount of memory per server in GB (`n_memory_per_server`)
- Number of NICs per server (`n_nics_per_server`)

In addition to the above parameters, the DCD file also specifies aggregate link capacities between different segments/entities of the DC. Two top-of-rack (ToR) switches (primary and standby) are always realized whenever a rack is created. In the present configuration, only the primary ToR switch is activated as failure scenarios are not yet considered, but necessary provisions are in place for this purpose.
The built-in parser of the DC model configurator module parses the DCD file and validates the parameters by performing sanity check on input parameters. The core logic then creates nodes and the hierarchical DC model by establishing interconnectivity between various nodes (i.e., interconnecting servers to ToR switches, the ToR switches to the aggregate switch(es) and the aggregate switches to the core switches).

As mentioned, the DCI model generator is developed in C++ and is composed of the following Classes:

- `Node` – the base class
- `Server` – models server entity; derived from Node,
- `SwitchAggr` – models Aggregation Layer Switch; derived from Node,
- `SwitchCore` – models Core Layer Switch; derived from Node,
- `SwitchToR` – models Top of Rack switch; derived from Node

2.4.3.2 vEPC Resource Model:

This module embodies the vEPC system model based on NEC vEPC system solution for providing EPCaaS. The details and architecture of this model is described in [6] and D4.3 [2]. This is a very complex module, which very accurately models and characterizes the individual SICs (MMP, S/PGW-C and S/PGW-U) in terms of the control plane (CP) and user-plane (UP) load that they process. The model also captures the interfaces between different relevant SICs. The details of the model are given in [6] and [2]. This model is associated with a SIC descriptor (SICD) file that enables the user to configure various parameters for the different SICs. The following are key user configurable parameters described in the SICD file:

- total connected eNBs – `n_enbs`
- control plane demand (in events/h) – `Ccp`
- control plane packet size (in bytes) – `cp_packet_size`
- user plane demand (in Gbps) – `Cup`
- user plane packet size (in bytes) – `up_packet_size`
- max load at S11 (S/PGW-C) – `max_S11`
- max load at S11 (S/PGW-U) – `max_S11_GWU`
- max load at S1C per MMP (ev/h) – `max_S1C`
- max S1U load per S/PGW-U (in Gbps)– `max_S1U`
- number of CPU cores per MMP – `n_cores_per_mmp`
- number of CPU cores per S/PGWC – `n_cores_per_gwp`
- number of CPU cores per S/PGW-U – `n_cores_per_gwu`
- memory required per SIC (in GB) – `n_mem_per_vnfc`
- storage required per SIC (in GB) – `n_storage_per_vnfc`

The built-in parser of the vEPC resource model parses the SICD file and validates the parameters by performing sanity check on input parameters. The core logic, which is implemented in C++, then creates the SICs, calculates the required number of SIC types to handle the specified CP/UP load, instantiates them and inserts them in a vector. It then creates and instantiate links between different SICs, calculates the respective load and inserts them in a vector.

The vEPC resource model and is composed of the following classes:

- `VNF` and `VNFC` – for modeling different types of SIs and SICs
- `Link` – Connects two SICs
- `RMG` – Models Resource Model logic; has vector of SICs and vector of Links

2.4.3.3 SIC Deployment Logic:

This module implements the “deployment strategies” using C++. The main Class is the VNFCDeployer that models the SIC deployment logic. In the present implementation two deployment strategies namely the VSD and HSD are available, the details of which are available in [2] and [6]. There is a built-in parser that reads the Deployer Descriptor (DD) file to select the specified deployment strategy. This module also receives inputs from the connectivity graphs generated by the DCI model generator and the vEPC resource model. All the SICs generated by the vEPC resource model are then deployed and instantiated onto the servers of the DCI model based on the selected

deployment strategy. The SICs are deployed in the servers of different racks based on the (anti)affinity rules between different SICs. After the SICs are deployed, links are instantiated by overlaying the vEPC links onto DCI links by searching the instantiated SIC's server and calculating the shortest path (Dijkstra algorithm) between the SICs and the external network.
This module provides load statistics at different layers of the DCI and provides CPU core utilization at servers. Figure 10 shows the internal interactions between the functional blocks of the Core module and what key parameters are exchanged between them.

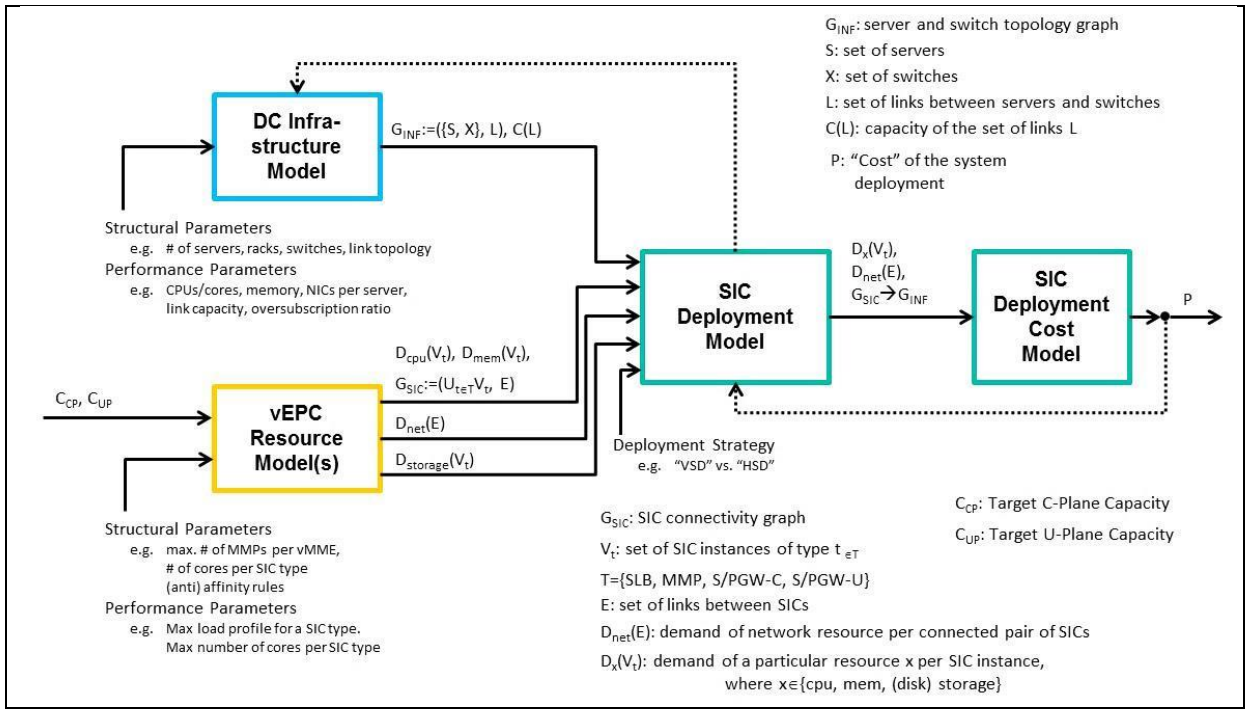

Figure 10 - Internal Interaction between the functional blocks of the Core Module

2.4.4 Management module:

As depicted in Figure 9, this module is composed of 2 main sub-modules; each sub-module is implemented in Python. The Management module enables the user to manage, control, configure and run the Core module. The output of the Core module, that contains key parameters, is fed back to the Management module, based on which the Management module derives the various cost factors and presents a graphical output to the user.

The following are the three main sub-modules of the Management module:

- Configurator interface
- Analytics engine

2.4.4.1 Configurator interface

Through this interface the user is able to specify and set parameters for the configuration files belonging to the sub-modules of the Core module, such as SICD, DCD and DD files as described above. After the parameters have been specified and written to the respective configuration/descriptor files, an execute command is initiated from the Management module towards the Core module. This will initiate the execution of the code in the sub-modules of the Core module. The Management module can also be extended to run automated batch test runs.

2.4.4.2 Analytics Engine

The output generated by the Core Module is fed back to the Analytics Engine that also contains an elaborate Parser for parsing the output file for necessary performance metrics. The Parser is implemented using the RegEx module of Python. The Analytics Engine derives performance metrics based on the values extracted by the Parser from the Core module's output file and arranges the data into matrices and lists for further processing by the graphing tool. Example performance metrics are:

- Total throughput per rack
- Average throughput per active server per rack
- Number of active servers per rack
- Total number of active (CPU) cores
- Average number of active (CPU) cores per rack

The performance metrics derived by the Analytics engine are then fed to a graphing function, which provides the performance and impact of the deployment strategy for the user specified configuration. The graphs provide the user with a quick overview and immediate insight. The user can perform necessary measures to design an optimum deployment configuration.

2.5 Virtualized EPC evaluation

The main part of the EPCaaS is the implementation of the EPC functionality as a software on top of virtual machines. This section provides an initial evaluation of the virtualized EPC functionality assessing different parameters.

Although a proper attention was given to optimize the processing of the different procedures within the EPC, the evaluation here presented remains tributary to the specific Fraunhofer OpenEPC / Open5GCore implementation, which was developed specifically for being deployed on top of COTS hardware as part of testbeds, missing a large amount of carrier-grade features which should be available in a product. Additionally, the results here presented should be seen as a rather volatile measurement, as the platform is in continuous development. New features may introduce additional delay, while other older features may be optimized for faster processing.

Ultimately, the measurements were performed on a single computer running the complete virtualization environment. Therefore they are accurate only for that specific computer combination. While porting the same EPC software to another data center, a different set of results may be obtained. These results as well as the further extension of the ones presented here will be presented in the evaluation deliverable.

A proper attention was given to the performance measurements within the benchmarking tool. Although there is no certainty that the delay perceived at the end subscriber is appropriately measured, the administrator of the testbed has made multiple checks using network monitoring tools to double-check that the delay is not introduced by the benchmarking tool itself. We assume that the delay introduced by the benchmarking system is up to 1ms which ultimately is the error of the end to end system.

The measurements here presented were acquired on a single virtualization environment with no orchestration. This solution was chosen to avoid interference with other components of the MCN project. Additionally, it provided a very short development loop for both the benchmarking tool, as well as, for the EPC components, enabling the fast feedback and optimization of the software. Additionally, the benchmarking tool removes the complete delay of the transmission over the radio link. Thus the scope of these measurements should be seen as limited to ONLY core network.

2.5.1 Evaluation environment

All the measurements were performed on a single system with a dual hexa-core Intel x64 and with 16GB of RAM memory. As illustrated in Figure 11. the EPC N:2 implementation option was installed together with the benchmarking tool. The benchmarking tool included a pool of 100.000 subscribers, for which subscription profiles were also introduced in the HSS. The benchmarking tool includes a set of 5 emulated eNBs which are connected to the same control entity.

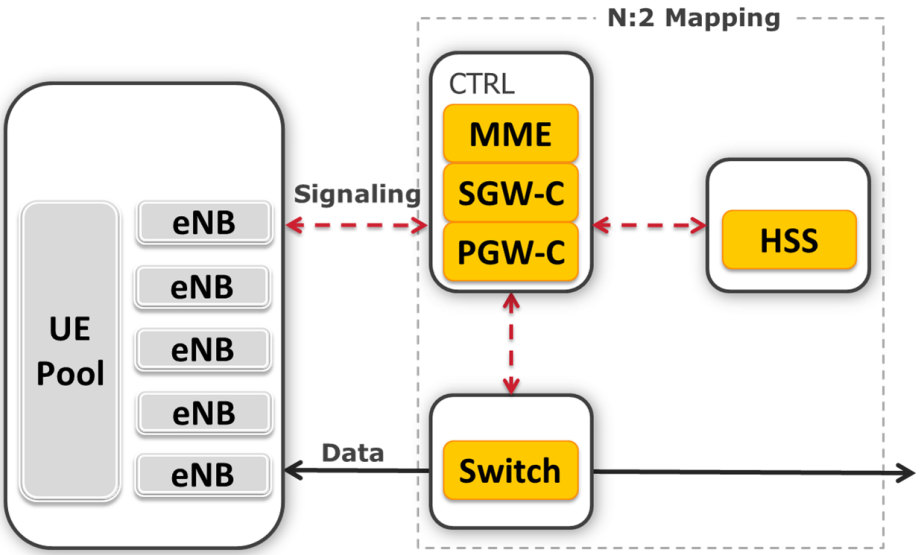

Figure 11 – Evaluation Environment

Different subscriber traces were used as input including a specific number of operations per-second. The operations include procedures for attachment, detachment or handover. As the scope of the measurements is to assess the performance of the service, the procedures are independent of the actual subscriber identity and of the actual mobility trace. For all the operations, the next user available in the list which can be used was always selected (e.g. for the attachments, the first unattached user, for the detachments the first attached user and for handovers the first attached user). Additionally for the handovers, the movement of subscribers is considered random and in a circular order. A subscriber can handover only to a next cell. This ensures that the processing at the Benchmarking Tool level is minimal and that the respective number of operations per second is respected, while not influencing the processing capabilities in the network.

2.5.2 Evaluation Results

The results here presented represent a very small sub-set of the operations that were performed. A more comprehensive evaluation including the comparison between physical and virtual infrastructures, the comparison between EPC 1:1 and N:2 implementation architecture and elasticity will be provided in the evaluation deliverable.

2.5.2.1 Required Cloud Resources

Regarding the performance of the system, a set of measurements were executed with 5 to 50 operations per second with the following percentage: 48% attachments, 47% detachments and 5% handovers. The results are depicted in Figure 12 in the form of percentage.

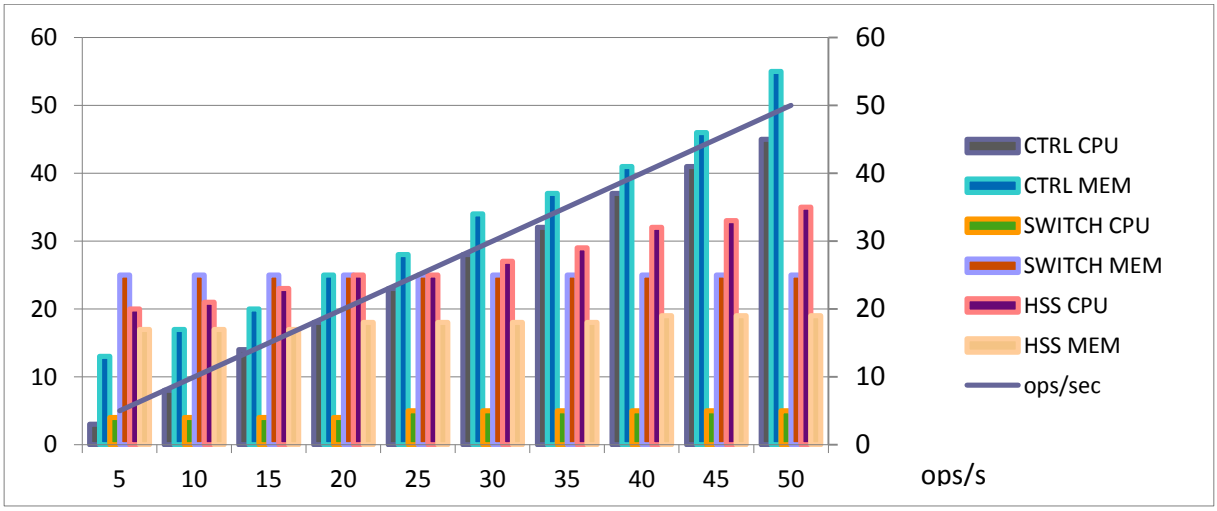

Figure 12 – CPU and Memory Usage for different levels of operations/second

As the operations are control only operations, there is a direct correlation between the number of operations per second and the CPU and Memory usage in the Controller entity. Additionally, as the switch receives only the pre-configured OpenFlow rules in order to install them into the specific forwarding table, the usage of the resources is rather uniform. A further check will be done to determine why there is a high level of memory usage in the switch. As the number of attachements and detachments highly overpasses the number of handovers, HSS is used in almost all the procedures. Thus, the HSS has to also scale according to the number of operations per second. However, as the operations related to the HSS are only request/reply based and the HSS does not have to take any decision, the resources required are smaller.

As expected, if there is enough capacity available and enough memory, then the system behaves rather uniformly i.e. there is a specific need of compute capacity to be available in order to process the requests correctly.

Please note that the system was developed to serve all the subscribers with the available resources (implicit operation). Thus when the limit of the available resources for computing is reached, the system will react slower and will not drop the requests.

2.5.2.2 EPC signaling delay

Regarding the visible delay at the subscriber side, coming from the core network, a large number of measurements were performed. An attachment takes 40-42 ms, a detachment 12-14 ms while a handover takes 17-19 ms. Therefore, through a back of the envelope basic computation, a single virtual CPU can process a maximum of 25 attachments/second. The charts are part of Annex C including the CPU consumption for the Benchmarking tool itself.

The low load assessment was required in order to be able to determine the duration of the procedures within the OpenEPC/Open5GCore. As additional results, the behavior is completely uniform up to 20 ops/sec. Afterwards, the behaviour between 20 ops/sec up to 60 ops/sec has a larger delay rather stable followed by a very high delay up to 200-300 ops/sec. This is mainly due to the software implementation and the internal queues.

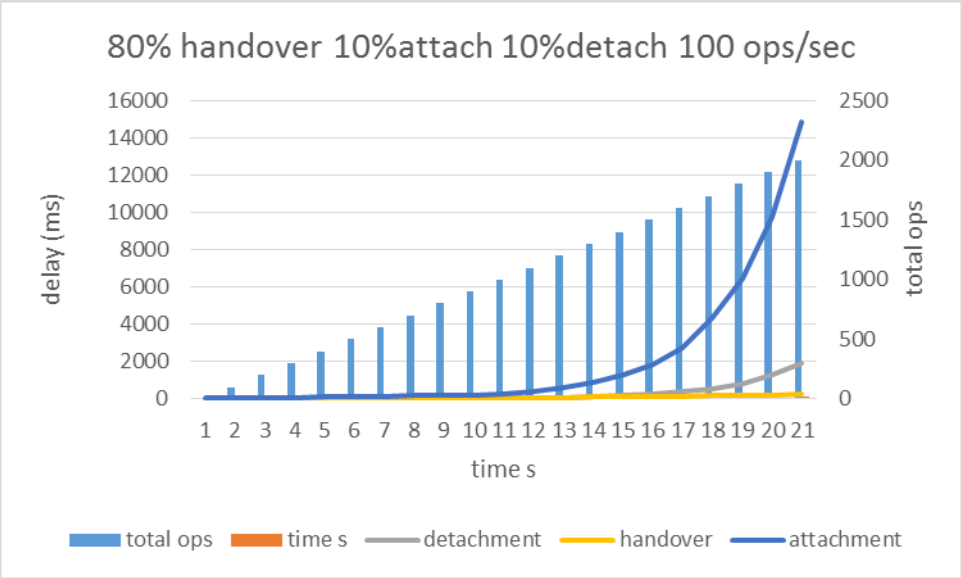

Figure 13 – Operations delay evolution with 100 ops/sec

However, the same single virtual CPU can hold up to 50 operations/second the system will respond with a larger delay up to 400 ms and will drop part of the session establishment messages. Afterwards, the delay increases to very high levels in the order of seconds. In terms of stability, we have not managed to reach a breach point for the entities even when running with 1000 attachments/second.

For the evaluation phase, different other subscriber patterns will be considered including more or less handovers as well as considering also the amount of subscribers active in the system at a specific moment in time.

2.5.2.3 Performance of the Attachment Procedures

Another set of measurements tried to evaluate the capacity of a single virtual machine with two processors to support attachment operations only. The results are depicted in Figure 14. We have observed that the system is becoming unstable after around 1500 attachments (i.e. the delay highly increases for the procedures over and goes over the limit of session failure or 10 sec at 1900 attachments).

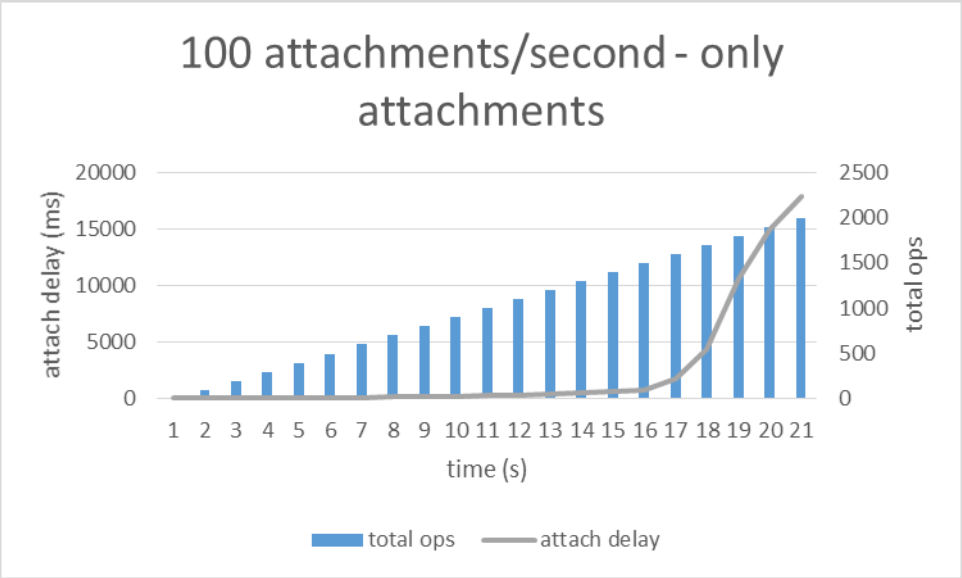

Figure 14 – 100 attachments per second

The current assumption is that the system is becoming unstable due to the number of requests received, which increase the processing queue. A similar measurement for 50 attachments per second is very stable for the duration of the test (up to 20 minutes tests).

From this we can conclude that 2 processors are enough in a typical cloud environment to support 50 attachments per second with 4 processors for 100 attachments and we foresee that with 8 processors we will support 200 attachments / second (comparative measurement will be provided in the evaluation deliverable). With this, we can conclude that a virtual EPC can support a rather large number of subscribers – in the magnitude of Millions – with a single machine with 8 cores. Note, for high availability 16 cores can offer the expected hardware redundancy.

2.5.2.4 Performance of the Handover Procedure

Another set of measurements tried to evaluate the delay of the handover procedure as seen from the subscriber side (excluding the RAN delay). The results are depicted in Figure 15. The system has two levels of stability – for low amount of operations the delay of the signaling procedure remains in the area of 20 ms. A second stable layer is after 1500 handovers, when the system remains stable at around 80ms/handover.

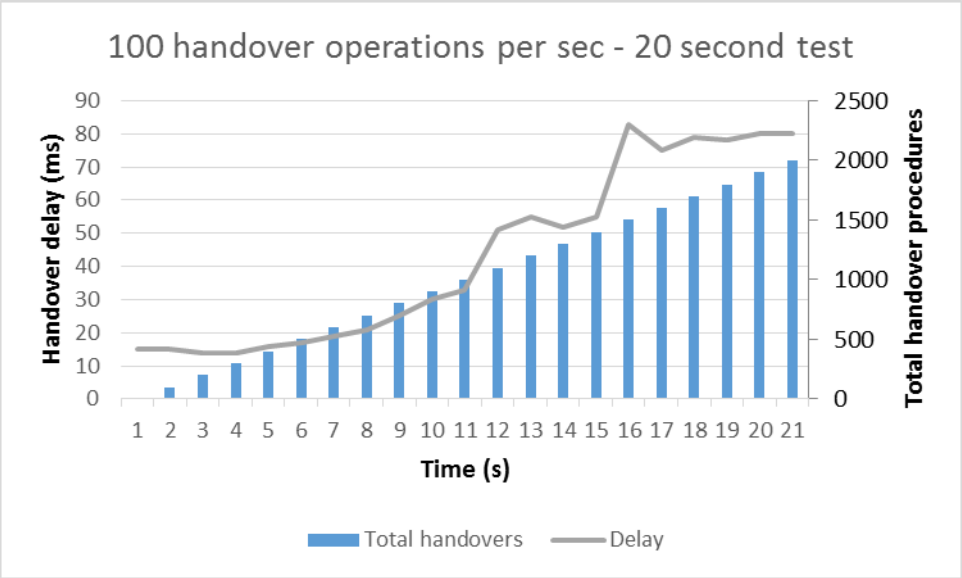

Figure 15 – 100 handovers per second

The current assumption is that the system is becoming stable in time at less than 20 ms per operation for low amount of operations which does not require any queuing and at 80 ms per operations for a larger amount of operations in which queuing is needed.

Additionally, the same test was performed for 50 handovers per second, as illustrated in Figure 16. The system is stable under 20 ms per operation. Additional to the provided figure, the same results are obtained for a very long duration of time e.g. for 2000s.

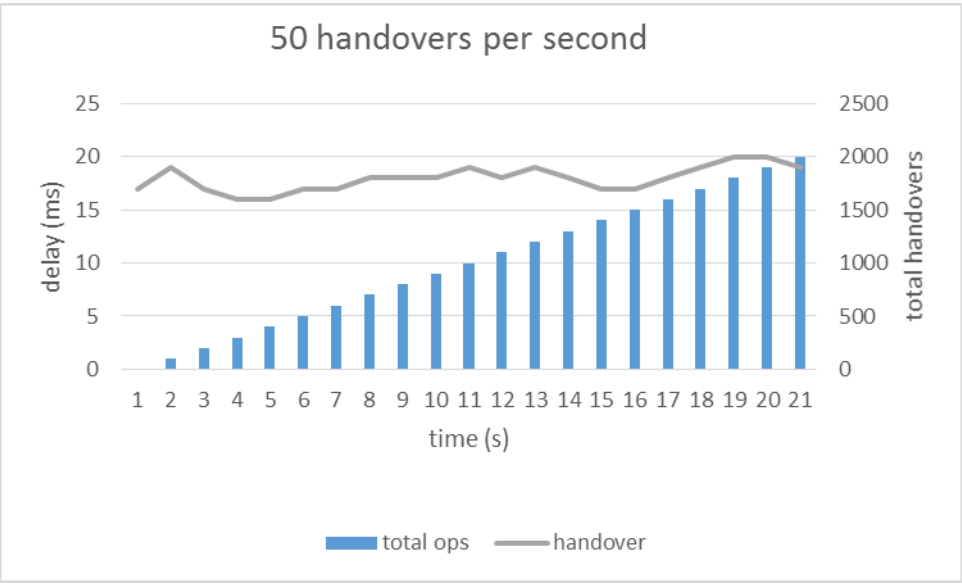

Figure 16 – 50 handovers per second

From this, we can conclude that for the signaling side a single server with 8 processors is able to maintain 200 handovers per second, which is equivalent to server 100.000 – 1.000.000 users.

2.5.2.5 Data path evaluations

The data path evaluation was implemented by using a huge number of ICMP messages started from the subscriber side and transmitted over the network.

Although a very low delay in communication was achieved, in the order of milliseconds, a further benchmarking of this functionality is still needed.

2.5.2.6 Conclusions

With the initial set of measurements in this section, it is presumed that the current network core evaluated has rather high performance in terms of delays perceived by the end subscribers, as well as, from the perspective of the networking resources consumed.

The delay introduced by network forwarding is rather small, mainly due to the virtualization of the benchmarking tool. An immediate next step is the test of data center capabilities with the external benchmarking tool when the delay of the radio responding to the procedures is also considered. This will provide a more accurate measurement of the virtualized EPC capabilities.

With rather fixed estimation of the specific LTE procedures for attachment, detachment and handover, it is simple to compute, at this moment, the capacity required for specific subscriber traces. Due to the flat structure of the code where different messages can be treated in parallel and where the subscriber state is maintained isolated, the degree of parallelization is very high, keeping a common serialized part very small. Due to this, the proposed system scales almost linearly with available processors.

Furthermore in the evaluation phase, the 1:1 and N:2 architectures will be compared adding also high capacity data path support based on RTP-like messages. Following, the elasticity procedures will be assessed in the term of network resources consumed.

3 ANDSFaaS Evaluation

This section describes the evaluation results of the ANDSFaaS service.

3.1 Updates from D4.4

The implementation of the ANDSF as a Service (ANDSFaaS) described in deliverable D4.4 [1] has been improved. The reason for this is the following. We decided to go further with flexibility extending the scaling capability to all tiers, i.e. beyond IRP (worker) as initially planned. This will increase agility, fostering the most efficient ANDSFaaS service management.

In D4.4 [1], and according to the 1:N architecture, the ANDSFaaS was implemented using 3 tiers: WS (webserver), IRP (core/worker) and DB (database); where only the IRP tier was able to scale (see Figure 17). Although this is already an interesting feature, some other components may suffer from heavy load as well, requiring increasing or decreasing resources. For this reason, we decided to carefully look into the remaining tiers to understand whether they could scale too to provide a fully scalable ANDSFaaS.

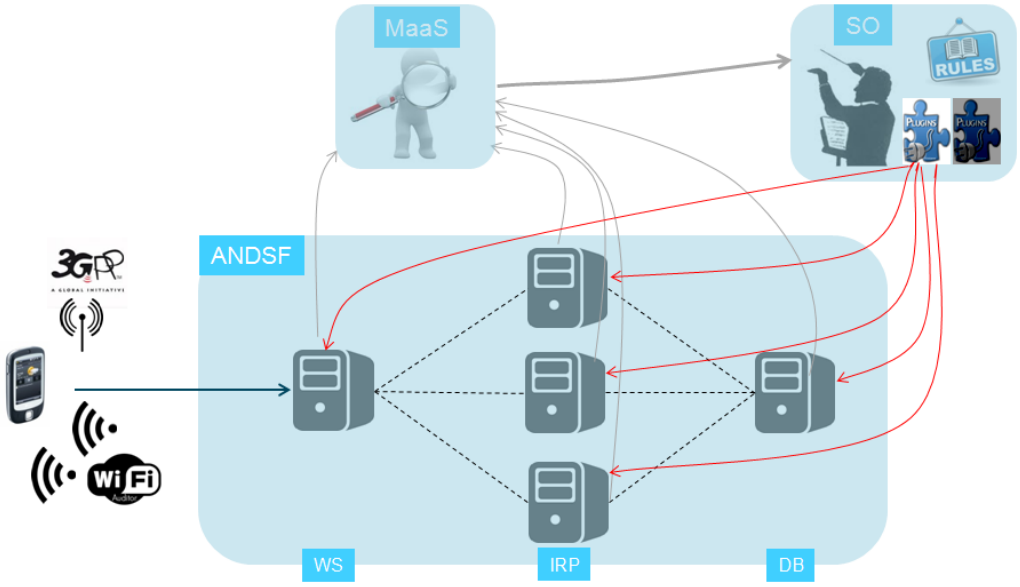

Figure 17 - ANDSFaaS architecture as described in [1]

The Figure 18 epicts the architecture of the final ANDSFaaS service implementation. This architecture includes an additional tier: the LB (Load Balancer). The LB is responsible to balance UEs (User Equipments) requests among multiple webservers (WSs). The UEs are the ANDSFaaS customers, using the 3GPP standard S14 interface. This tier enables the ANDSFaaS to balance the WSs, by allowing them to scale, and increasing the ANDSFaaS flexibility. This way, the new LB tier is the only one that remains not scalable, although, if needed, it could scale too by using traditional DNS load balancing solutions; i.e. the same FQDN is associated to multiple IPs, and the DNS server performs a round robin name resolution.

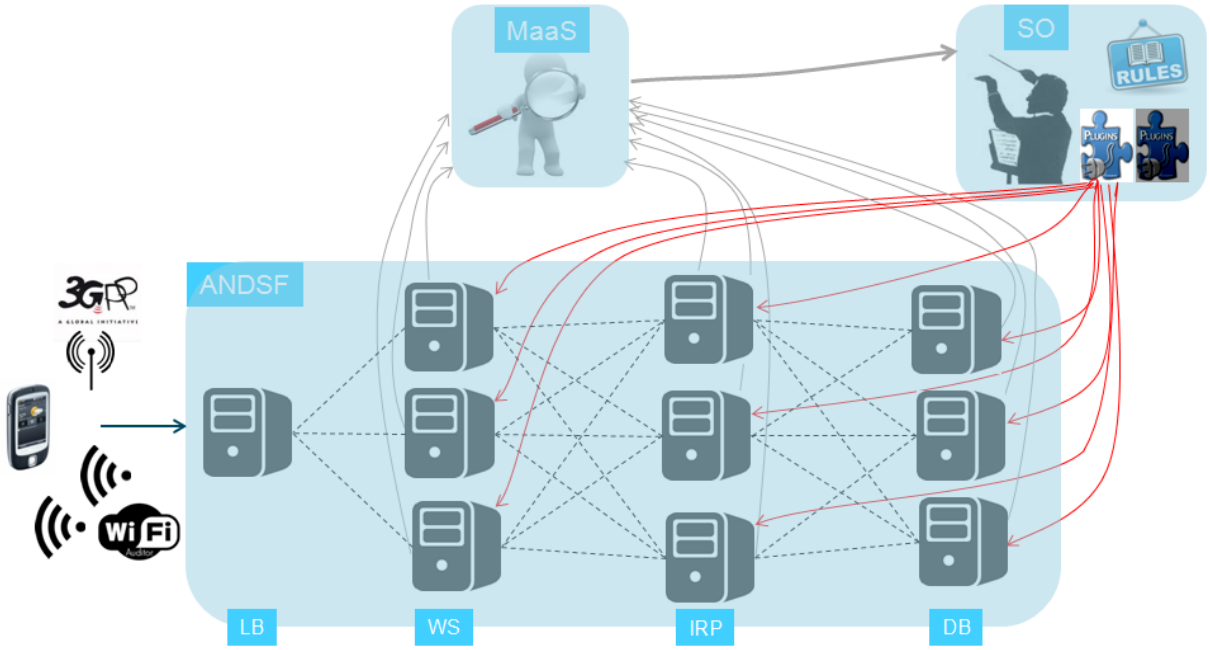

Figure 18 – Updated ANDSFaaS service architecture

The Service Orchestrator (SO) has been also upgraded in comparison to the version from D4.4 [1] in order to monitor additional resources, as well as, to take actions on additional components. The SO is now able to monitor metrics on multiple tiers, store multiple rules per tier, and take deployment and scaling actions over them.

The scaling of WS and IRP tiers relies on the creation of component pools by configuring load balancing mechanisms in upstream tiers. As those tiers are stateless, scaling these tiers is just a matter of creating additional components and balance them. The Database (DB) is a different issue, since this approach does not apply. For the DB tier scaling, the mongodb sharding solution [7] is used, splitting data and processing among multiple shards.

The basic software implementation design does not change because of these enhancements. Only additional features are coded in order to monitor, decide and actuate over additional tiers, using new metrics, rules and actions. Regarding the rest, all aspects remain the same as described in D4.4.

3.2 Definition and Scope

The ANDSFaaS component developed and documented in deliverable D4.4 [1] showed that a virtualized ANDSDFaaS service can be run on top of a cloud. It demonstrated that it is possible to on-demand instantiate a complete software-based ANDSFaaS service and, based on monitoring data, scale it automatically in order to adjust the resources consumption to the ongoing requirements, saving costs and providing high service agility.

3.3 Innovation

This section highlights the main innovations that ANDSFaaS brings to the virtualization of network functions (VNFs). In short, the pioneering of this architecture and implementation, the flexibility and agility of the solutions are key aspects to consider, making this work worldwide innovative.

Pioneering: First of all, it is important to underline that this project started in November 2012, when virtualization of network functions was not yet the big hype it is today. Remember it is

exactly in November 2012 when the ETSI NFV Group was created, and only one year later when the first drafts were released. By that time, the MCN has already the architecture fully specified and starting the development work. For this reason, the MCN achievements, namely ANDSFaaS, are highly valuable. By the beginning of 2014, the ANDSFaaS was already able to be deployed on top of an Openstack based cloud, through the Cloud Controller (CC), by using the SDK (Software Development Kit) mediation.

Flexibility: The ANDSFaaS provides a high level of flexibility, by allowing the component to automatically adapt the resources he is using to the required level of performance, making this way an efficient use of resources and saving costs. In the final implementation stage, the ANDSF is able to scale-in and out in all its tiers, being able to be adapted to any performance situation. The scaling capability is triggered by monitoring (resource or service metrics), but it could also be triggered by prediction, or any other source.

Simplicity: The ANDSFaaS deployment and scaling can be easily configured by using a rules engine, where the scaling in and out thresholds are easily configured, using (*IF condition THEN action*) simple rules format. The SO is the piece responsible to receive the monitoring data and check the thresholds, in order to decide the actions to take. In case a new monitoring metric is added, there is no need to make new developments; it is only a matter to configure the monitoring service (MaaS) and use this metric in the rules configuration file. Rules can be changed during ANDSFaaS execution and will take effect immediately as the new monitoring data comes. This model provides high levels of agility to the service.

Modularity: The implementation of the SO is modular (and based on SDK), so that a new service (other than ANDSFaaS) can be orchestrated with minor effort. It is only required to implement the specific aspects of the service, like particular configurations, graphs, service monitoring metrics and actions to perform an initial deployment and scaling. This allows the service owners to easily create SOs for multiple services, reusing the same software and experiencing a similar approach.

Integration: The ANDSFaaS is able to integrate with other MCN services, providing high value added to the service. In particular, the ANDSFaaS can be integrated with the monitoring service (MaaS) and the rating, charging, billing service (RCBaaS). They allow the ANDSFaaS service to be provided with additional features, which are relevant for production-level environments.

Although the innovative aspects, there are some limitations that may be mentioned, which should be subject of improvement in the future, in order to allow this kind of solutions to move to production. The most significant limitation is related to the capability of the service to follow the SO orchestration agility and avoid scaling downtimes.

Scaling Downtime: The ANDSFaaS is able to perform on-the-fly scaling according to a set of predetermined rules. However, the provisioning of such changes (add new WS or IRP components) requires that the different SICs (service instance component) can handle those changes without service loss. In the current ANDSFaaS version, and despite the effort devoted to solve these questions, there is still some downtime associated to scaling procedures, which relates to the time elapsed to recreate the connectivity between tiers. Nevertheless, considering the time measured (see results in section below), the impacts are not significant, considering typical scaling operations.

Delay: Usually, operator network components use to be inside the operator's network, close to UEs and benefitting this way from a small delay. In an ANDSFaaS deployment on a central (large) Datacenter, delays can increase, degrading the quality experienced (QoE) by UEs.

Maturity: This kind of solutions are not yet mature, requiring some time to ensure that this can work in a production environment, where users are high availability levels (99.999%). Operator must gain some confidence that this is gone a work all the time.

3.4 Evaluation Scenario

This section describes the scenario used to evaluate and validate the ANDSFaaS service. This general scenario serves the purpose for all tests, as was used to obtain the results described below in section 3.5. Specific parameterizations and configurations will be set in a per test basis. Those are specified together with the test description.

3.4.1 Physical Machine

The overall scenario runs on top of a physical machine (PM) with the following main characteristics.

Table 12 System Under Test Machine Characteristics

Resource	Value
Machines	1
CPU	Intel Xeon E5-2640v2 16 Cores (2 GHz) / 32 Processors
Memory	96GB
Disk	4TB
Network	4 x 1GigabitEthernet

3.4.2 Evaluation Scenario

The Figure 19 depicts the general evaluation scenario. The parameterizations and configurations described below are common for all tests. Specific parameterizations/configurations are referred in a per test basis in the results section 3.5.

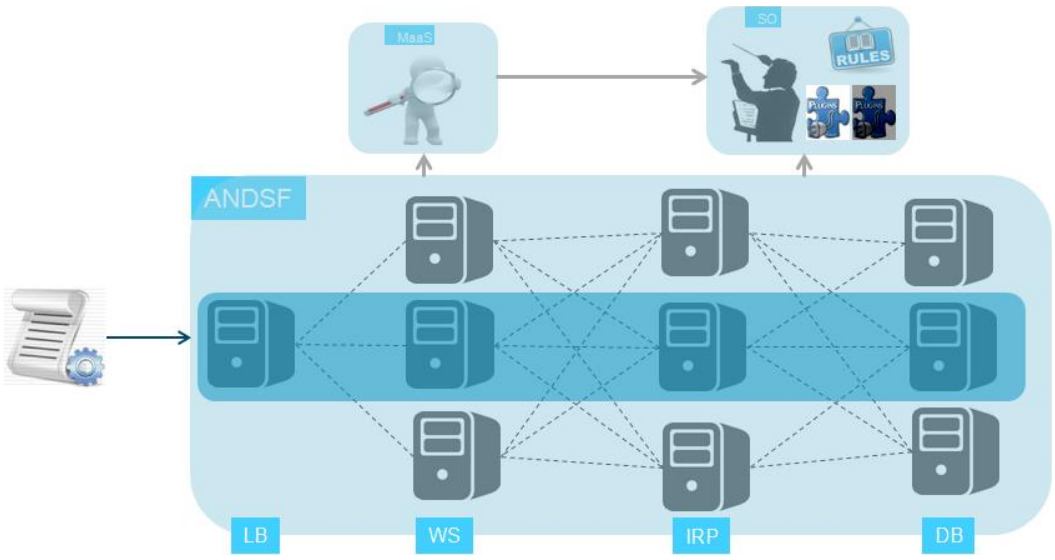

Figure 19 – ANDSFaaS overall evaluation scenario

In this scenario, the WS, IRP and DB tiers may be composed by multiple (variable) number of components (SICs) – the back boxes, depending on the performance required. For the case of the WS and IRP tiers, SICs are all the same function, while for the DB case, different components with different features are deployed (according to the sharding technology).

At instantiation time, the configuration dictates a fixed number of components to deploy per tier. Once the instantiation process is completed, the service is ready for use. From this moment on, the SO will get monitoring data and compare it with the scaling rules (thresholds), in order to decide whether they dictate a change in the current ANDSFaaS deployment, scaling out or in the service, accordingly. After a decision is taken, the SO enforces that decision by managing the required (re)configurations.

3.4.3 Common Parameterization/Configuration

For the overall scenario, some parameterization/configurations will be constant for all tests.

- The VM's profile, for simplicity, is the same for all machines (VMs) in all tiers.

Resource	Value
CPU	2 vCPUs
Memory	1 GB
Network	2 x GigabitEthernet

- In order to ensure consistency, only one tier will scale at a time, locking scaling operations for the others.
- In order to ensure service stability, after any scaling operation, a guard time of 30 seconds is awaited with no scaling operations.

3.4.4 Variable Parameterization/Configurations

On a per test basis, some parameterization/configuration will change for the different tests.

- On instantiation, a predefined set of machines are placed by tier (usually, the minimum resources are used).
- On scaling in/out, a configurable number of VMs are incremented/decremented per scaling event.
- Scaling in/out rules can be triggered by any metric; however, for simplicity, only the number of services requests and the CPU utilization metrics will be used.
- In order to emulate custom service requests from UEs, the JMETER [8] tool was used, allowing a flexible definition of the requests profiles. That make possible to put some load on the service and trigger scaling out and in decisions.

3.5 Results

This section describes the results of the evaluation work related to the ANDSFaaS service.

This evaluation results focuses on management and orchestration aspects of the ANDSFaaS, mainly regarding deployment, scaling and reliability operations. These results should be considered in order to evaluate the performance of management, namely how much time it takes to fully instantiate an ANDSFaaS instance, to scale any tier or to receiver from a component crash. Also important is to get aware about the impacts of those operations on the physical machine.

3.5.1 Test 1 – ANDSFaaS Deployment

This test intends to evaluate the deployment performance, relating to the job performed by the SO. It measures the time elapsed to deploy and release a fully operational ANDSFaaS service instance using multiple configurations. That is an important issue, which provides a measure about how agile the service management is. It also measures the resources impacts on the physical server.

3.5.1.1 Test Description

This test performs 3 different deployment configurations. For each deployment, a different number of VMs per tiers is configured (see below). This intends to evaluate the behaviour for deployments with a different number of initial VMs.

Table 13 – Deployment Configurations

Configuration	VMs per Tier
Deployment 1	LB – 1 VM WS – 1 VMs IRP – 1 VMs DB – 2 VMs *(*minimum DB cluster size)*
Deployment 2	LB – 1 VM WS – 2 VMs IRP – 2 VMs DB – 3 VMs
Deployment 3	LB – 1 VM WS – 3 VMs IRP – 3 VMs DB – 4 VMs

3.5.1.2 Test Results

The sections below show the results obtained for the multiple deployment configurations.

3.5.1.2.1 Deployment 1

The following sections depict the main results obtained for the Deployment 1.

3.5.1.2.1.1 Deployment

The Figure 20 shows the time elapsed to deploy an ANDSFaaS instance fully operational and ready to respond to service requests from UEs. The chart shows the tiers under deployment and the times taken to be in the following 3 steps:

- OS) VM created in OpenStack, before OS starting;
- VM Ready) the OS has fully started;
- Service Ready) the ANDSFaaS service is configured and ready to use.

As a result, this ANDSFaaS deployment takes 320 seconds, 5 minutes and 20 seconds, to be fully deployed. This value is very reasonable; however, with a better hardware, this can be still significantly improved.

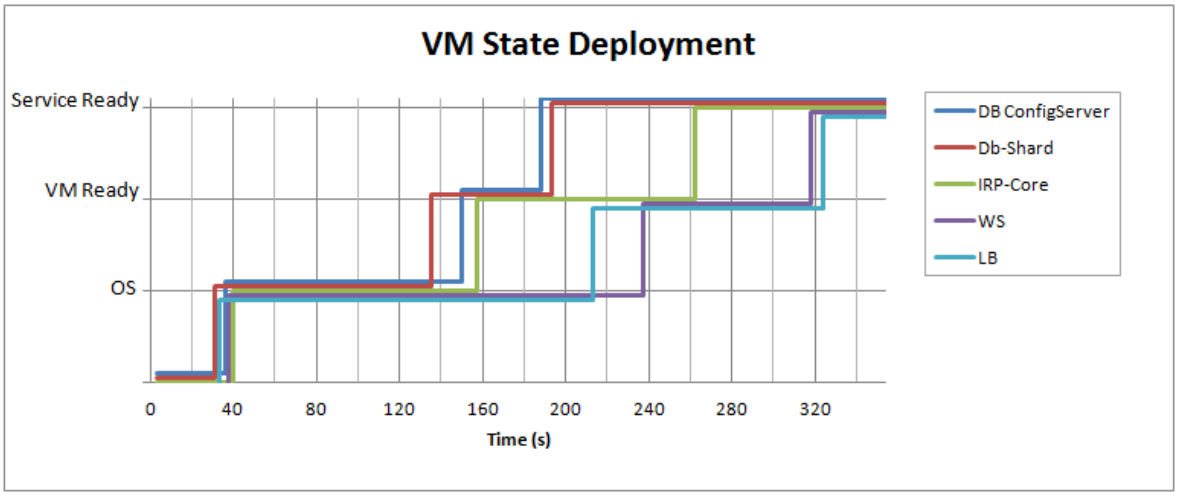

Figure 20 – ANDSFaaS: Deployment 1 (deployment time)

3.5.1.2.1.2 Release

The Figure 21 shows the time elapsed to release an ANDSFaaS instance, freeing all resources associated. The chart shows the tiers being released, when OS, VM Ready and Service Ready have the same meaning as described above.

As a result, this ANDSFaaS release takes 60 seconds, 1 minute, to be fully disposed. This value is very good, although with a better hardware, this can be still significantly improved.

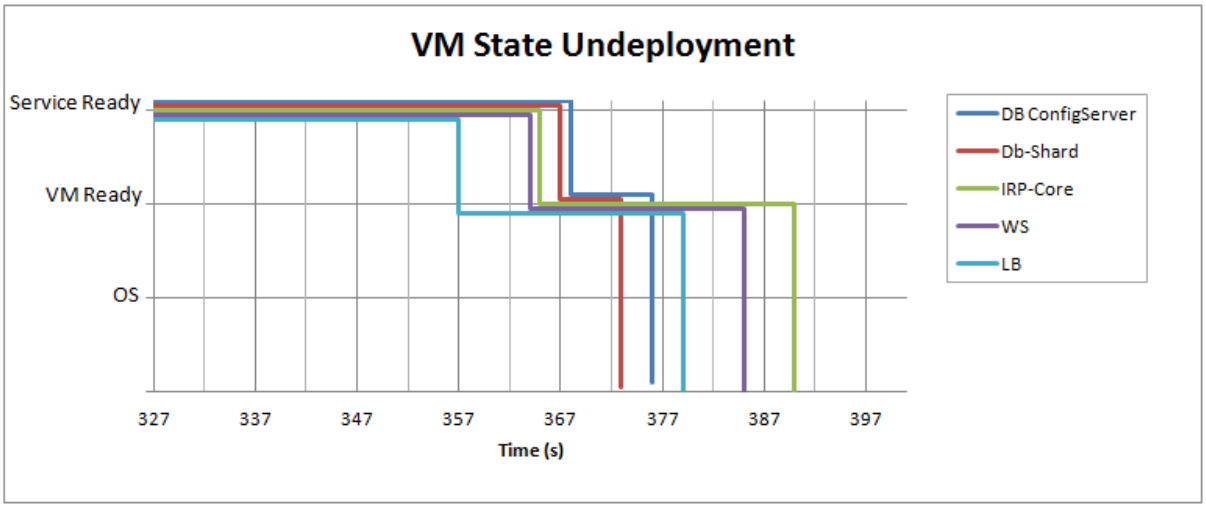

Figure 21 – ANDSFaaS: Deployment 1 (release time)

3.5.1.2.1.3 Deployment and Release

The Figure 22 shows the combination of deployment and release of an ANDSFaaS instance, as already shown above. Both operations are concluded in 390 seconds, 6 minutes and 30 seconds.

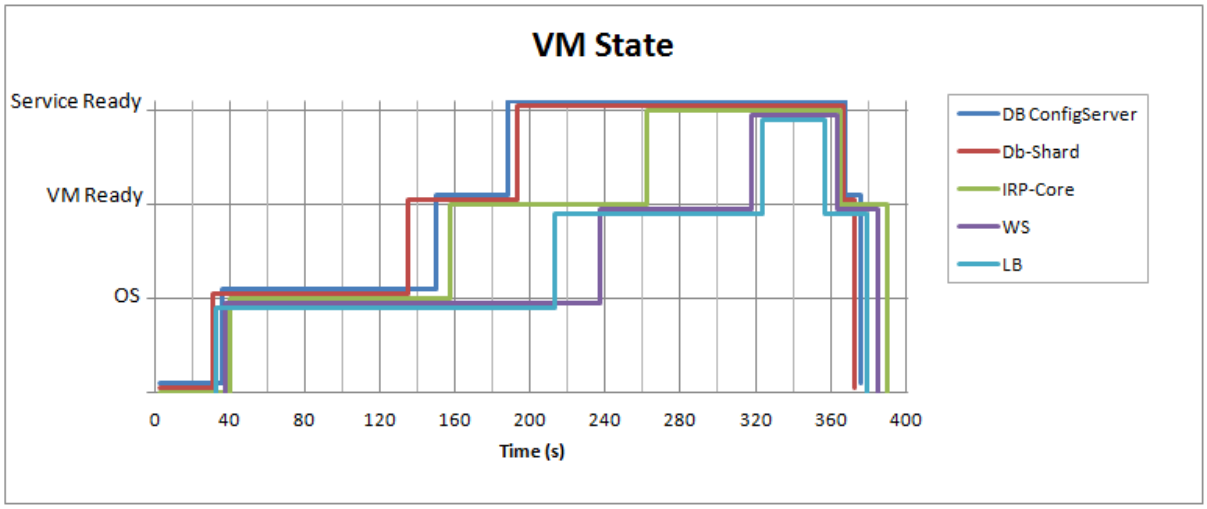

Figure 22 – ANDSFaaS: Deployment 1 (deployment and release times)

3.5.1.2.1.4 Impacts on Physical Machine

The Figures below show the impact of the ANDSFaaS deployment and release operations on the physical machine resources. In particular, the Figure 23, Figure 24 and Figure 25 show the impacts on CPU, RAM and I/O utilization along the deployment and release periods.

As a result, it can be seen that the most relevant impact on the physical machine relates to I/O utilization. The CPU faces a significant increase, while RAM utilization suffers minor changes.

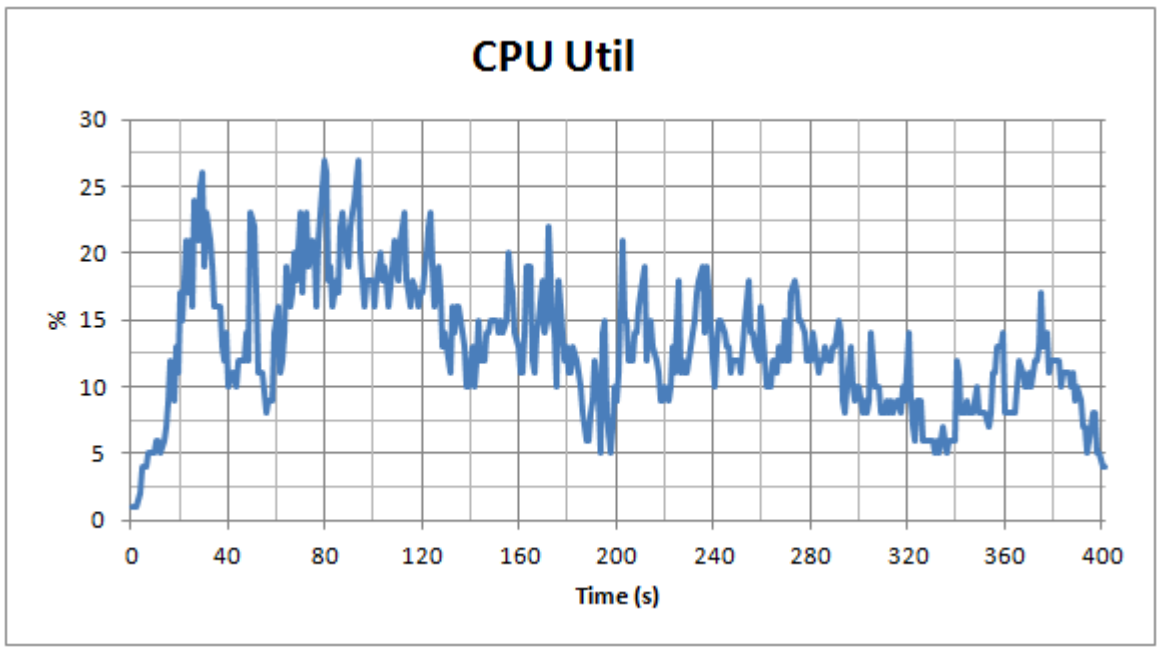

Figure 23 – ANDSFaaS: Deployment 1 (PM CPU utilization)

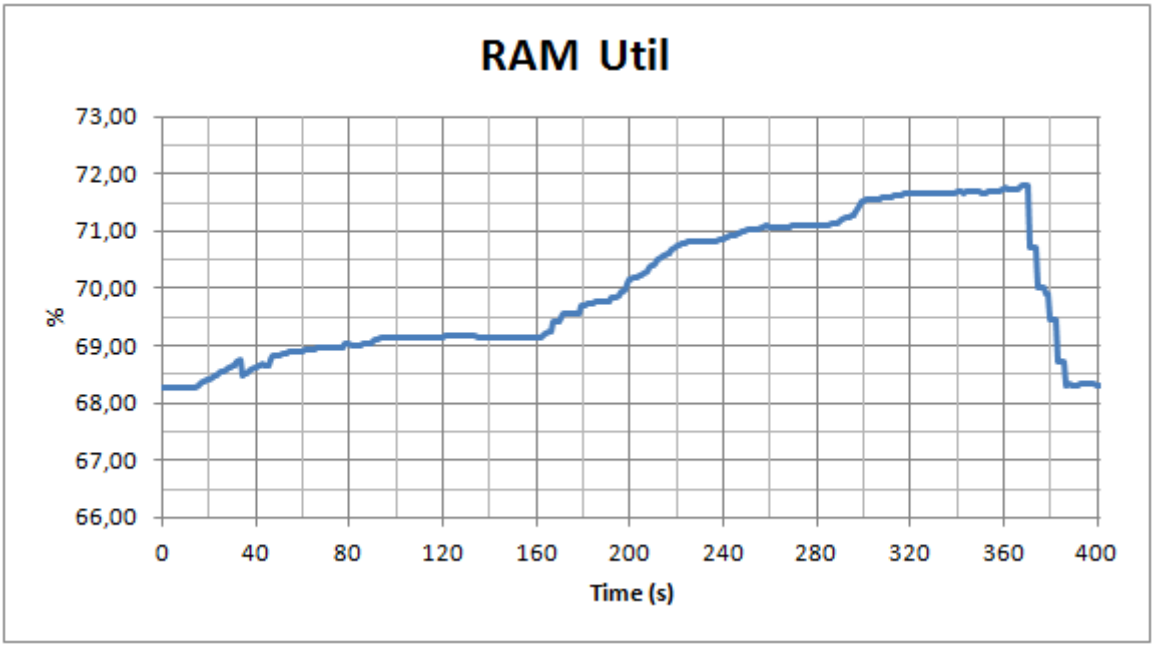

Figure 24 – ANDSFaaS: Deployment 1 (PM RAM utilization)

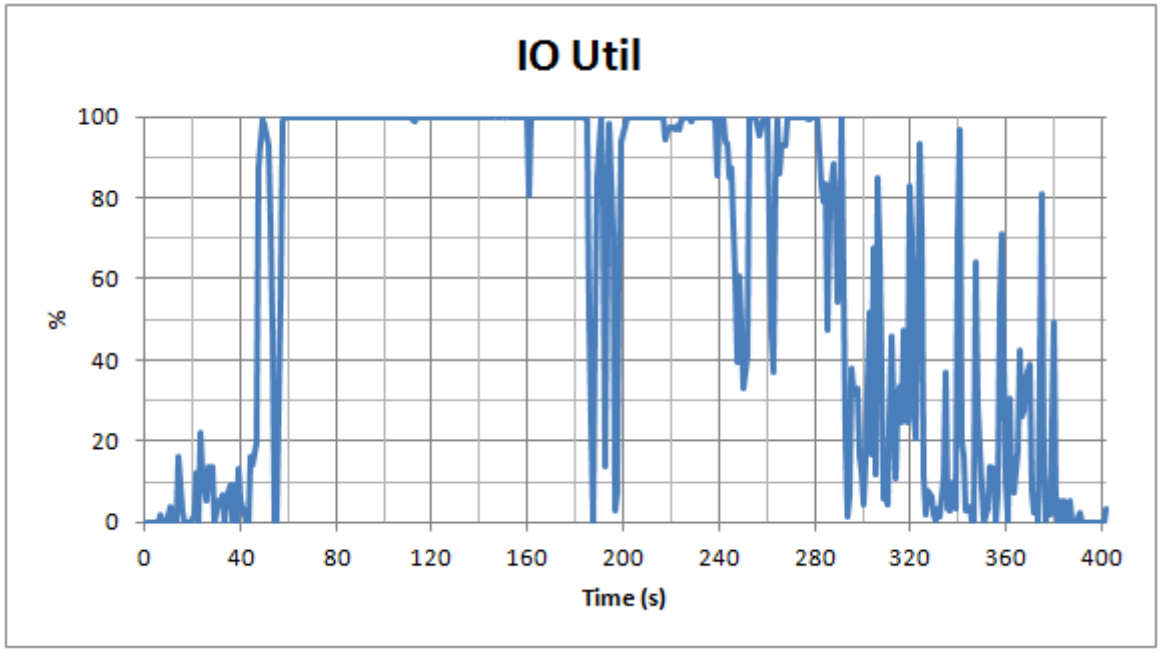

Figure 25 – ANDSFaaS: Deployment 1 (PM I/O utilization)

3.5.1.2.2 Deployment 2

The following charts depict the main results obtained for the Deployment 2.

3.5.1.2.2.1 Deployment

The Figure 26 shows the time elapsed to deploy an ANDSFaaS instance fully operational and ready to respond to service requests from UEs.

As a result, this ANDSFaaS deployment takes 490 seconds, 8 minutes and 10 seconds, to be fully deployed. This value is around 50% higher than for Deployment 1, as a result of 3 (~50%) additional components (VMs) to deploy, which seems a reasonable and linear behaviour.

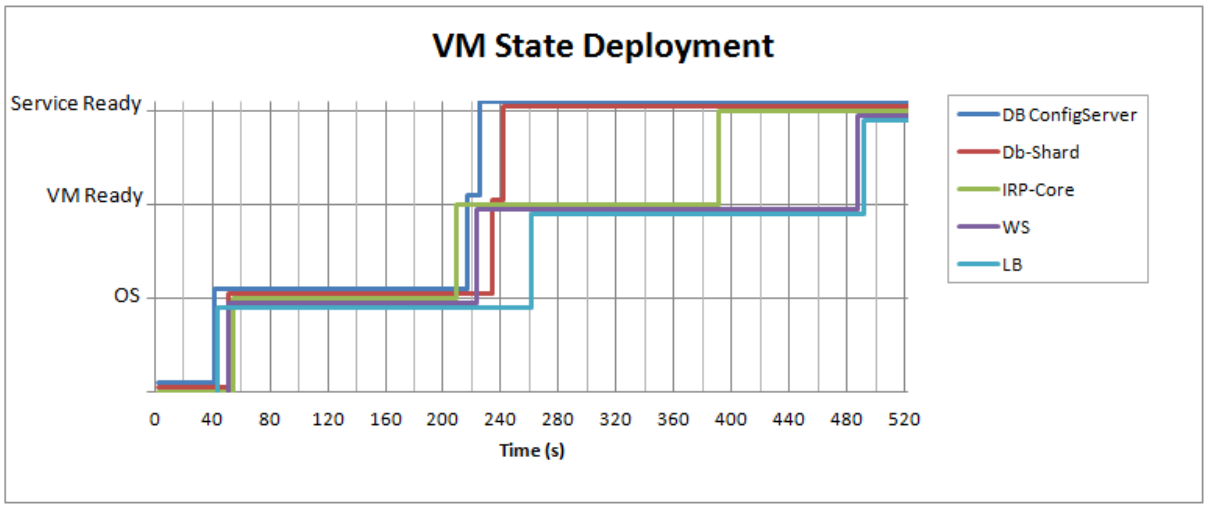

Figure 26 – ANDSFaaS: Deployment 2 (deployment time)

3.5.1.2.2.2 Release

The Figure 27 shows the time elapsed to fully release the resources of an ANDSFaaS instance.

As a result, this ANDSFaaS deployment takes 80 seconds, 1 minute and 20 seconds, to be released. This value is around 50% higher than for Deployment 1, as a result of 3 (50%) additional components (VMs) to release, which seems a reasonable and linear behaviour.

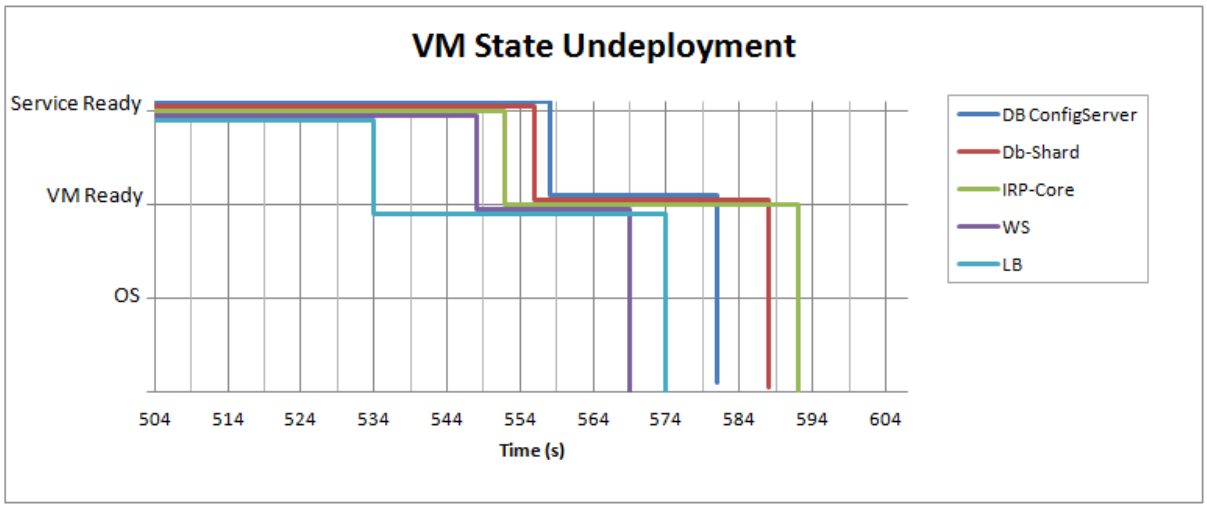

Figure 27 – ANDSFaaS: Deployment 2 (release time)

3.5.1.2.2.3 Deployment and Release

The Figure 28 shows the combination of deployment and release of an ANDSFaaS instance, as already shown above. Both operations are concluded in 590 seconds, 9 minutes and 50 seconds.

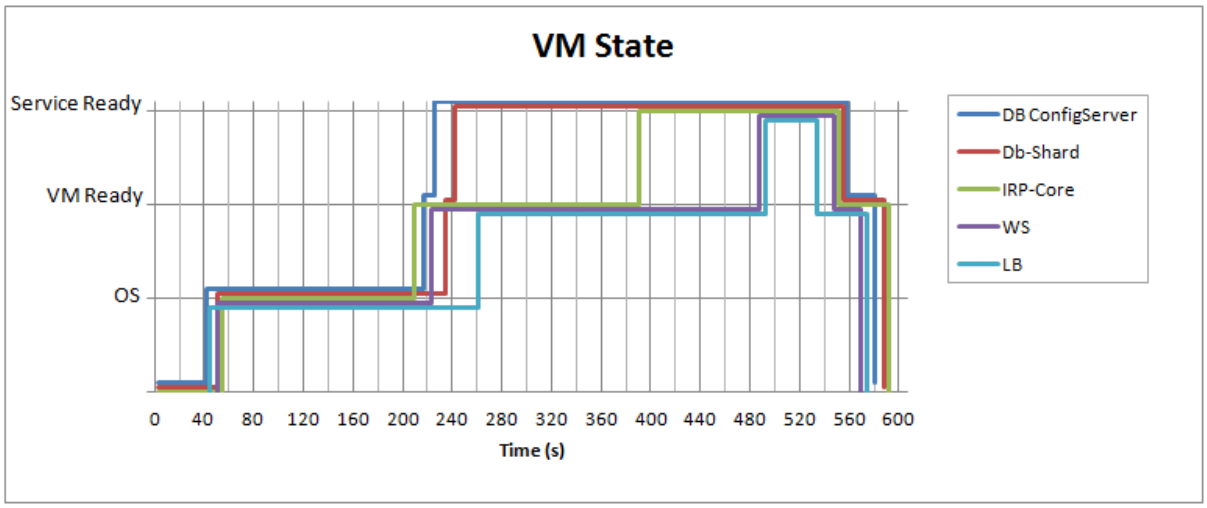

Figure 28 – ANDSFaaS: Deployment 2 (deployment and release times)

3.5.1.2.2.4 Impacts on Physical Machine

The Figures below show the impact of the ANDSFaaS deployment and release operations on the physical machine resources. In particular, the Figure 29, Figure 30 and Figure 31 show the impacts on CPU, RAM and I/O utilization along the deployment and release periods.

Compared to the Deployment 1, the results are similar, with a high impact on I/O, some on CPU and a few on RAM. There is no increase of resources consumption, although it occurs during more time.

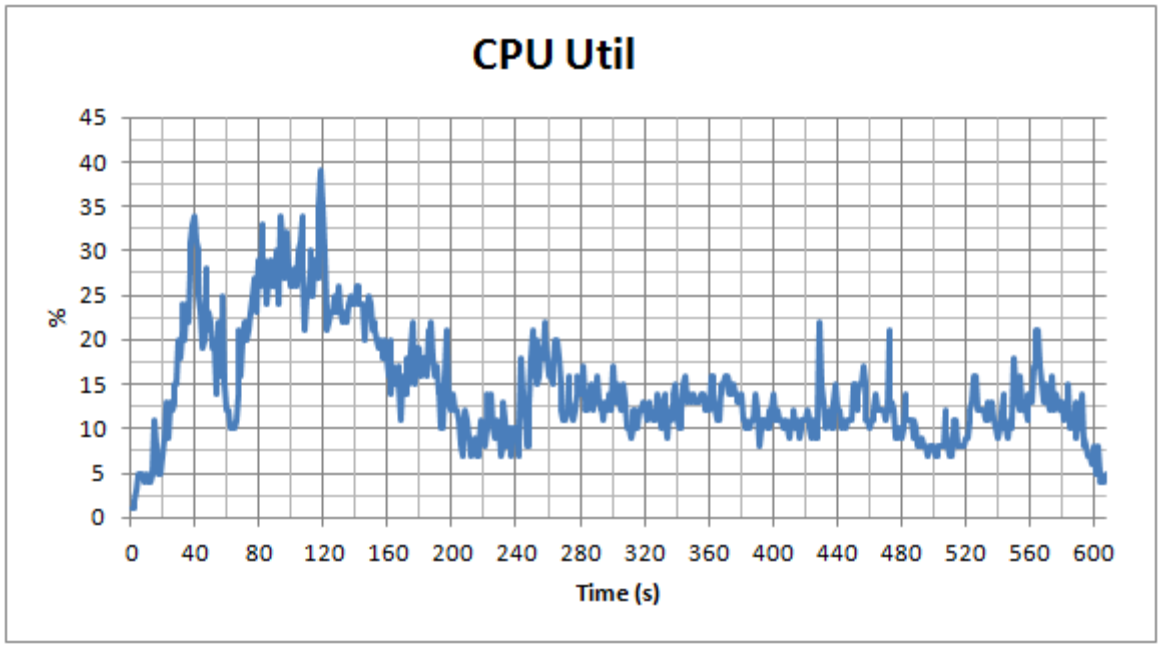

Figure 29 – ANDSFaaS: Deployment 2 (PM CPU utilization)

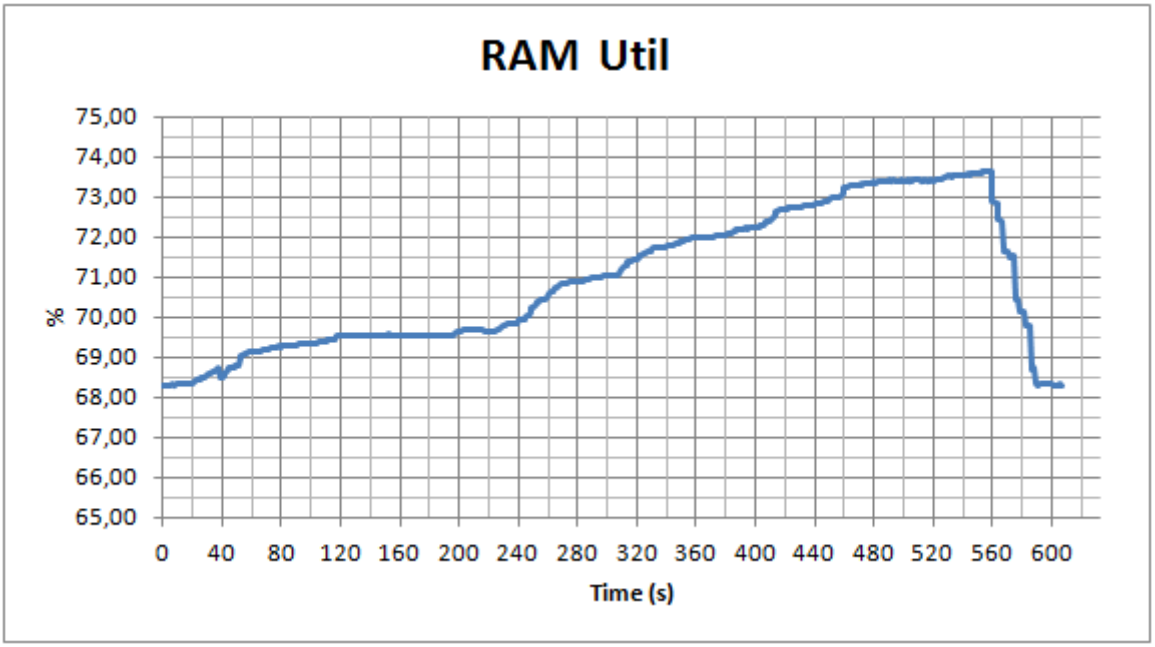

Figure 30 – ANDSFaaS: Deployment 2 (PM RAM utilization)

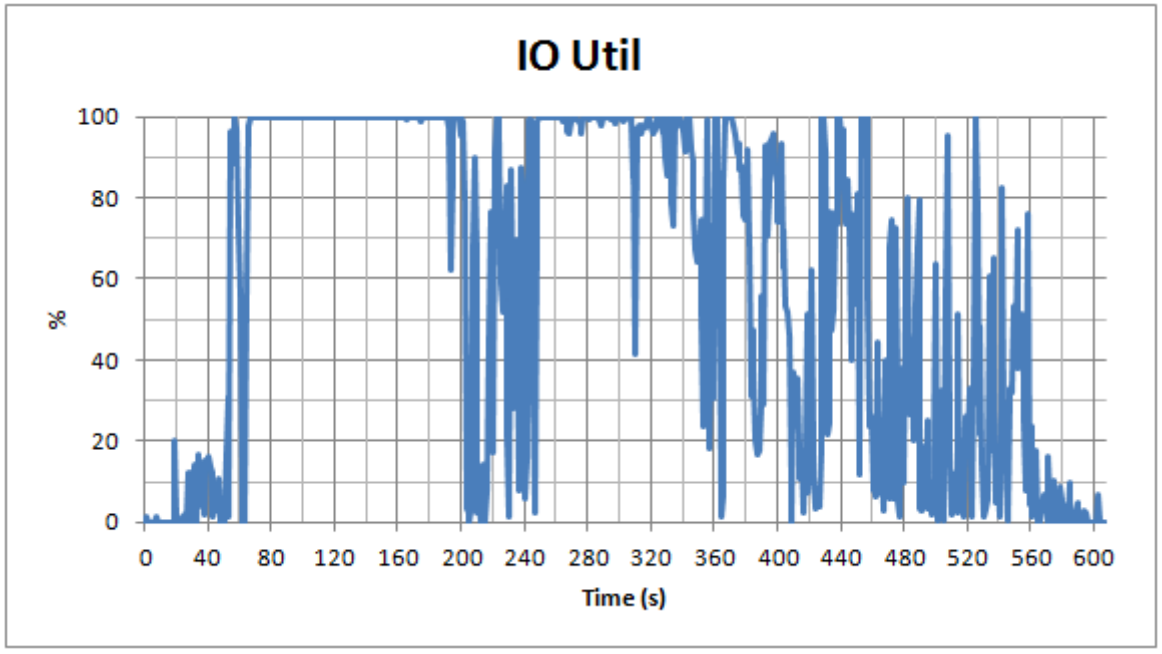

Figure 31 – ANDSFaaS: Deployment 2 (PM I/O utilization)

3.5.1.2.3 Deployment 3

The following charts depict the main results obtained for the Deployment 3.

3.5.1.2.3.1 Deployment

The Figure 32 shows the time elapsed to deploy an ANDSFaaS instance fully operational and ready to respond to service requests from UEs.

As a result, this ANDSFaaS deployment takes 660 seconds, 11 minutes, to be fully deployed. This value is around 30% higher than for Deployment 2 and 100% higher than Deployment 1, as a result of additional components (VMs) to deploy. Again, this seems reasonable and linear behaviour.

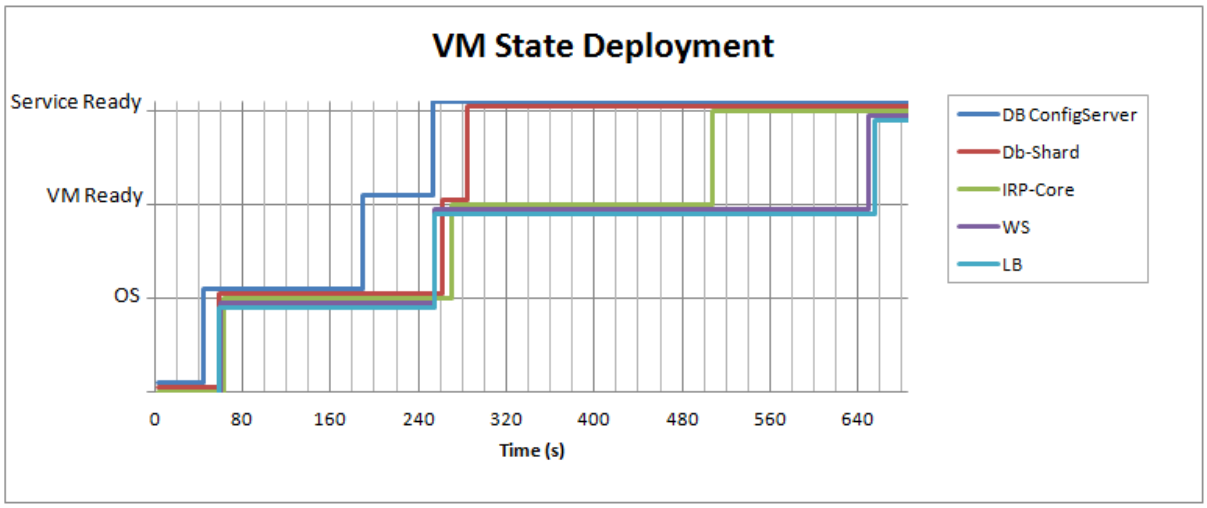

Figure 32 – ANDSFaaS: Deployment 3 (deployment time)

3.5.1.2.3.2 Release

The Figure 33 shows the time elapsed to fully release the resources of an ANDSFaaS instance.

As a result, this ANDSFaaS deployment takes 110 seconds, 1 minute and 50 seconds, to be released. This value is around 30% higher than for Deployment 2 and 100% higher than Deployment 1, as a result of additional components (VMs) to release. Again, this seems a reasonable and linear behaviour.

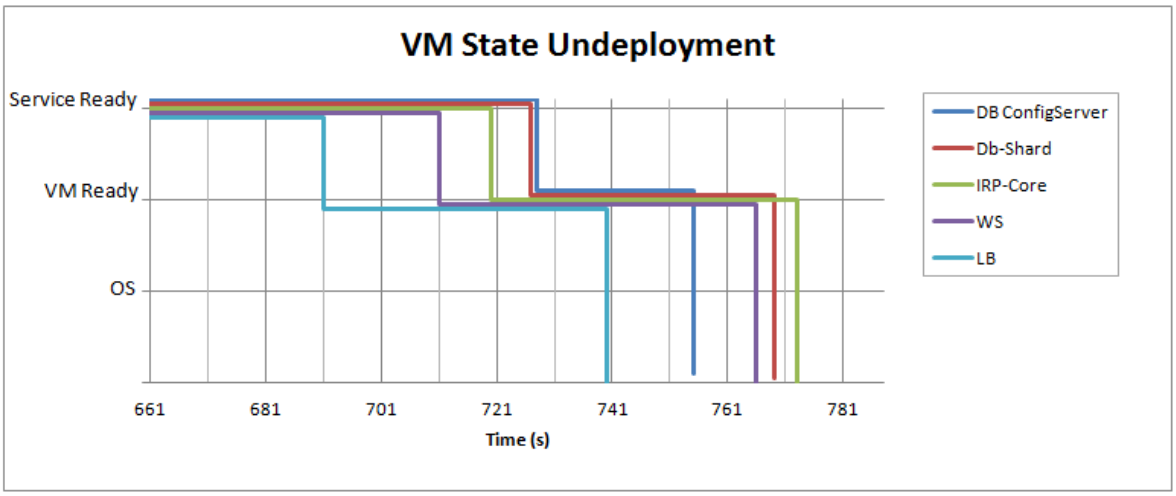

Figure 33 – ANDSFaaS: Deployment 3 (release time)

3.5.1.2.3.3 Deployment and Release

The Figure 34 shows the combination of deployment and release of an ANDSFaaS instance, as already shown above. Both operations are concluded in 780 seconds, 13 minutes.

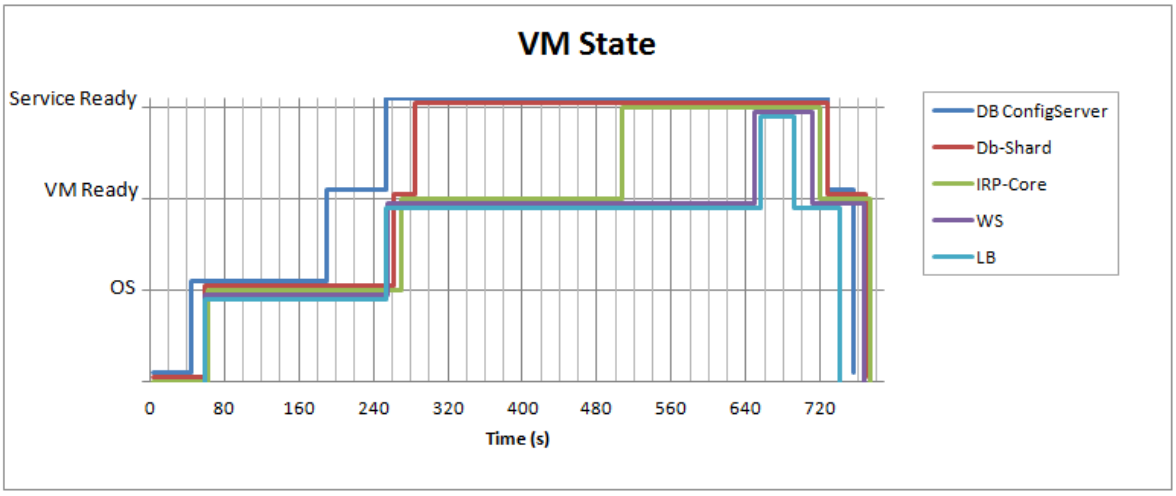

Figure 34 – ANDSFaaS: Deployment 3 (deployment and release times)

3.5.1.2.3.4 Impacts on Physical Machine

The Figures below show the impact of the ANDSFaaS deployment and release operations on the physical machine resources. In particular, the Figure 35, Figure 36 and Figure 37 show the impacts on CPU, RAM and I/O utilization along the deployment period.

Compared to the Deployment 2 and Deployment 1, the results are similar, with a high impact on I/O, some on CPU and a few in RAM. There is no increase of resources consumption, although it occurs during more time.

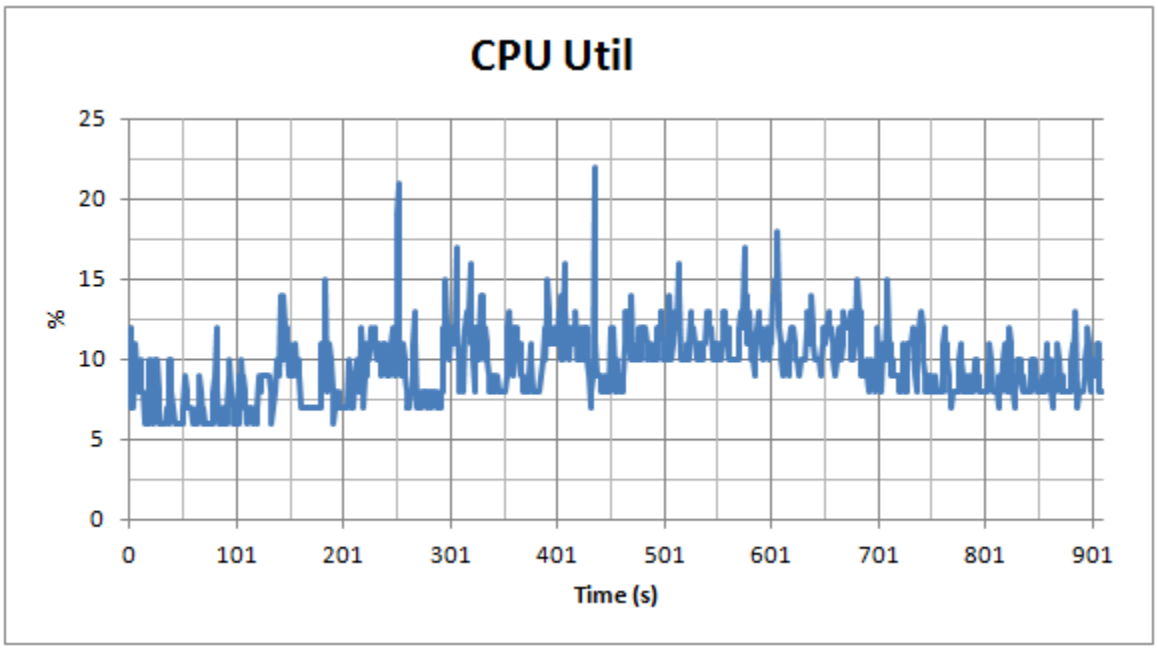

Figure 35 – ANDSFaaS: Deployment 3 (PM CPU utilization)

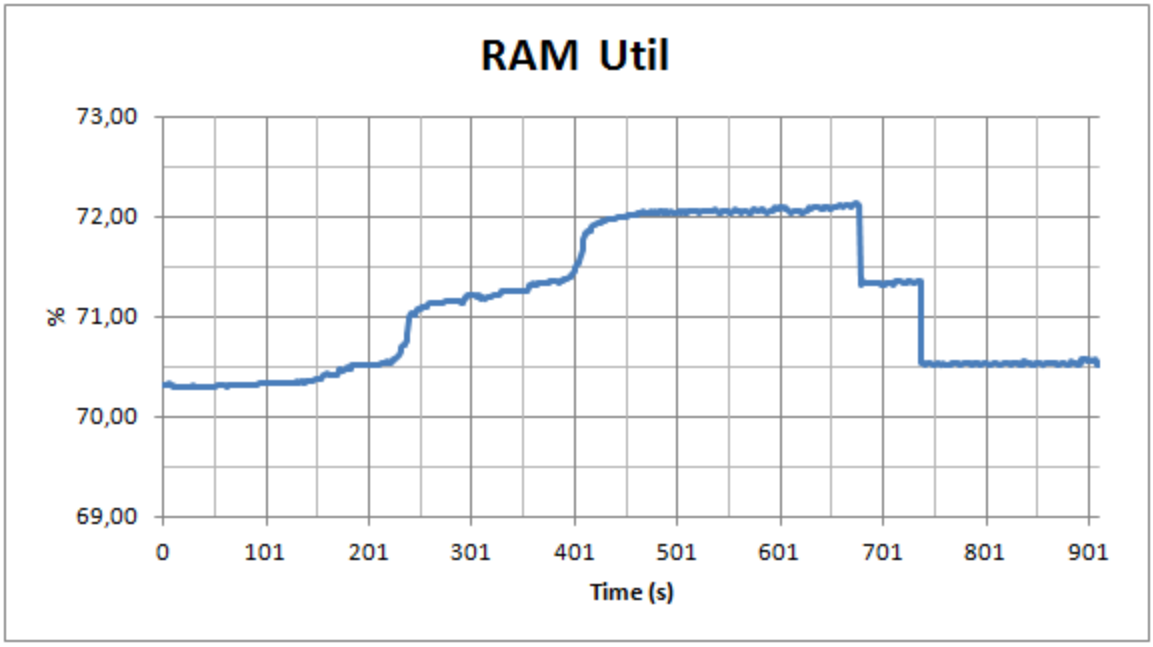

Figure 36 – ANDSFaaS: Deployment 3 (PM RAM utilization)

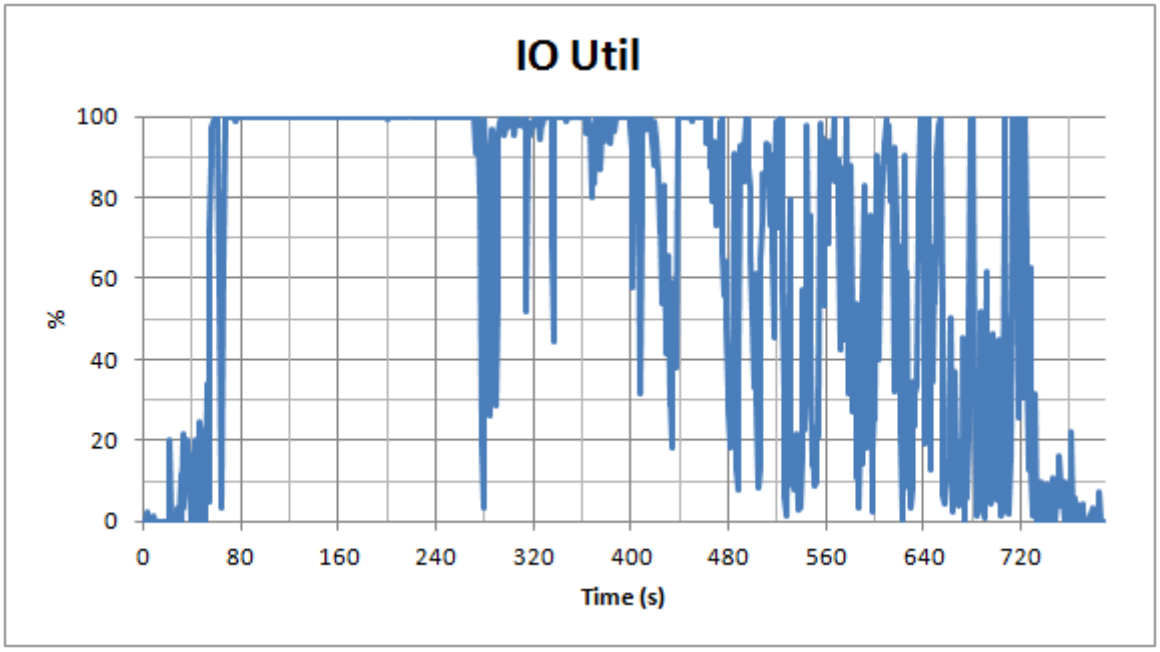

Figure 37 – ANDSFaaS: Deployment 3 (PM I/O utilization)

3.5.1.3 Conclusions

As overall conclusions, the deployment test results show that the number of nodes used for an initial instantiation has impacts on the required time to deploy an ANDSFaaS service-ready instance. Instantiations with more nodes take more time; however, the good news is that this time seems linear. That means that it was not noticed any exponential increase of time for large instantiations, keeping the time per node more or less constant.

Regarding the overall time, for the 3 deployment configurations, the instantiation time varied from around 5 to 10 minutes, which is reasonable, considering that the physical machine used is below average hardware. Anyway, it is possible to implement agile services with no significant constraints.

For ANDSFaaS service release operations, the times were, as expected, considerably lower than for instantiation. ANDSFaaS release seemed to have a linear behaviour too, ranging from 1 to 2 minutes for the 3 deployment configurations considered.

Regarding the impacts on the physical machines, it was noticed that the I/O utilization is the most impacted resource on instantiation, having significant impacts. The CPU and RAM have noticed a slight impact. For ANDSFaaS disposal, the impacts are significantly reduced on I/O, while the CPU and especially RAM get significant resource utilization reductions as resources are released.

3.5.2 Test 2 – ANDSFaaS Scaling

These tests intend to evaluate the management and orchestration performance of the SO, considering scaling operations for the 3 scalable tiers: WS, IRP and DB. Scaling operations are triggered by metrics monitored using the MaaS. The metrics here considered are the number of requests (Req) – a service related metric –, and the CPU utilization – an infrastructure related metric. In fact, scaling can be easily triggered by any other metrics, by just configuring it on the scaling rules configurations file.

The scaling out operations creates additional components of a given tier, in order to increase the load capacity, reacting to periods of more load on service requests. Scaling in operations intend to dispose unnecessary resources, when the number of requests decreases, in order to save resources and costs.

In these tests, the base configuration for each tier is evaluated, introducing changes regarding:

- The traffic generation profile, more/less aggressive (slope), Profile 1/Profile 2;
- The number of components incremented/decrements per scaling event: +-1 VM, +-2 VMs;
- The metric used as trigger, i.e. number of requests (Req) or CPU utilization (CPU);

3.5.2.1 Test Description

The scaling tests start all from a basic deployment (Deployment 1), as described in the table below.

Table 14 – Test configuration

Configuration	VMs per Tier
Deployment 1	LB – 1 VM WS – 1 VMs IRP – 1 VMs DB – 2 VMs

The scaling rules used are as follows for the different tiers, both for the number of requests (Req) and CPU utilization (CPU). They indicate the thresholds for scaling out and scaling in.

Table 15 – Scaling policies

Tier	Nº of Requests (*Req*)	CPU Utilization (CPU)
WS	IF (Req >= 10) THEN scale-out IF (Req <= 4) THEN scale-in	IF (CPU >= 30%) THEN scale-out IF (CPU <= 12%) THEN scale-in

IRP	IF (Req >= 10) THEN scale-out IF (Req <= 4) THEN scale-in	IF (CPU >= 30%) THEN scale-out IF (CPU <= 12%) THEN scale-in
DB	IF (Req >= 20) THEN scale-out IF (Req <= 8) THEN scale-in	IF (CPU >= 6%) THEN scale-out IF (CPU <= 2%) THEN scale-in

Anytime a scaling operation is performed, the number of Components (VMs) to be instantiated or released varies. For those tests, increments of 1 and 2 VMs were considered for all tiers.

Table 16 – Scaling Operation Actions

Tier	Scale-in 1	Scale-out 1	Scale-in 2	Scale-out 2
WS	-1	+1	-2	+2
IRP	-1	+1	-2	+2
DB	-1	+1	-2	+2

In order to test the scaling flexibility, a different number of requests (Req) profiles were used. Two different profiles are here considered, the more aggressive (Profile 1) and less aggressive (Profile 2).

Table 17 - Load profiles

Time	Profile 1 (Req/s)	Profile 2 (Req/s)
$T_{Initial}$ Rate	0 Req/s	0 Req/s
Increment Rate	+1 Req/s every 10s	+1 Req/s every 15s
T_{Max} Rate	35 Req/s	35 Req/s
Decrement Rate	-1 Req/s every 10s	-1 Req/s every 15s
T_{Final} Rate	0 Req/s	0 Req/s

3.5.2.2 Test Results

3.5.2.2.1 WS Scaling

3.5.2.2.1.1 Base Configuration

This section considers the scaling tests with the following conditions, which is the configuration base for further comparison.

- **Scaling trigger: number of requests (Req);**
- **Scaling increments: +-1 VM;**
- **Traffic generation: Profile 1 (high increasing and decreasing rate).**

3.5.2.2.1.1.1 Scale out

The Figure 38 depicts the time elapsed to scale out an ANDSFaaS service instance on the WS tier, showing the WS2 and WS3 instances (WS1 was already instantiated) and the 3 instantiation steps (OS, VM Ready and Service Ready).

As a result, it can be seen that each scaling out operation takes 150 seconds, 2 minutes 30 seconds, which seems a reasonable result considering the performance of the hardware used.

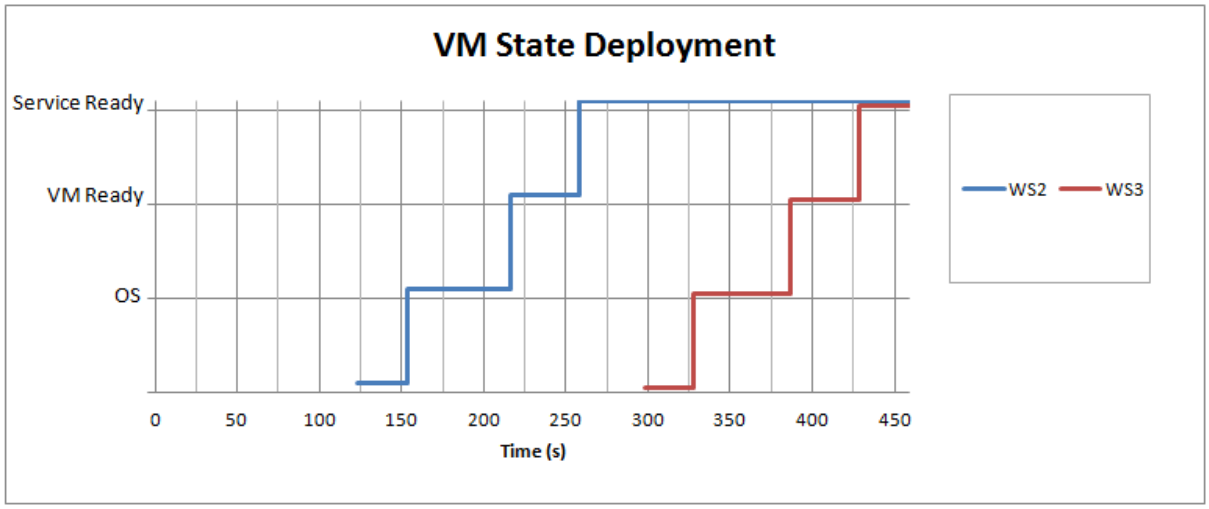

Figure 38 – ANDSFaaS: Scale WS, Trigger Req, Increment 1, Profile 1 (scale out)

3.5.2.2.1.1.2 Scale in

The Figure 39 depicts the time elapsed to scale in an ANDSFaaS instance on the WS tier, showing the WS3 and WS2 instances being released (WS1 was already instantiated).

As a result, it can be seen that scaling in operations take 20 seconds which is, as expected, significantly lower compared to the scaling out.

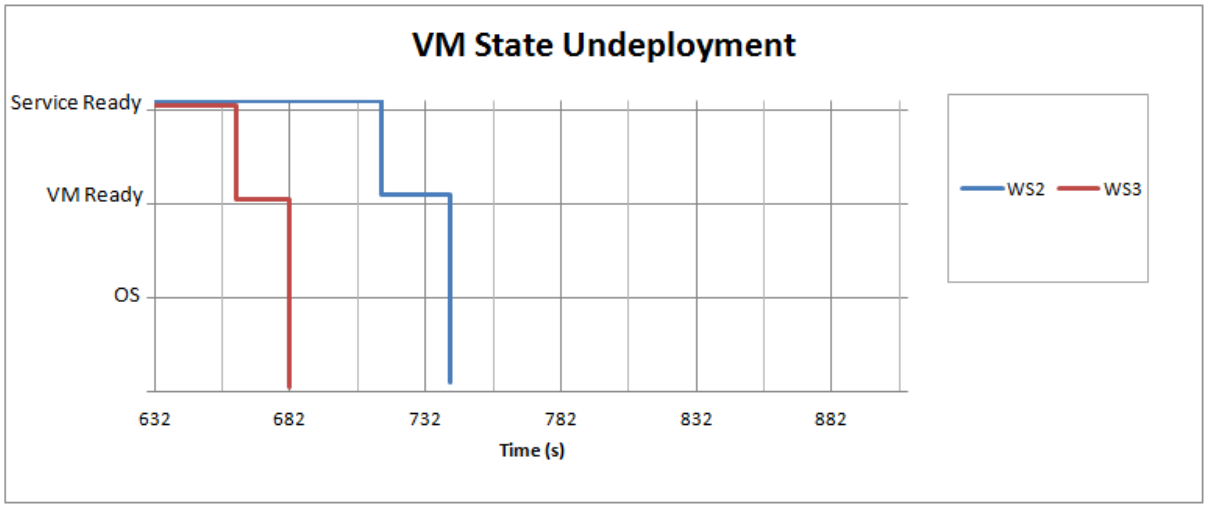

Figure 39 – ANDSFaaS: Scale WS, Trigger Req, Increment 1, Profile 1 (scale in)

3.5.2.2.1.1.3 Scale out and in

The Figure 40 shows the combination of scaling out and in an ANDSFaaS instance on the WS tier, as already shown above.

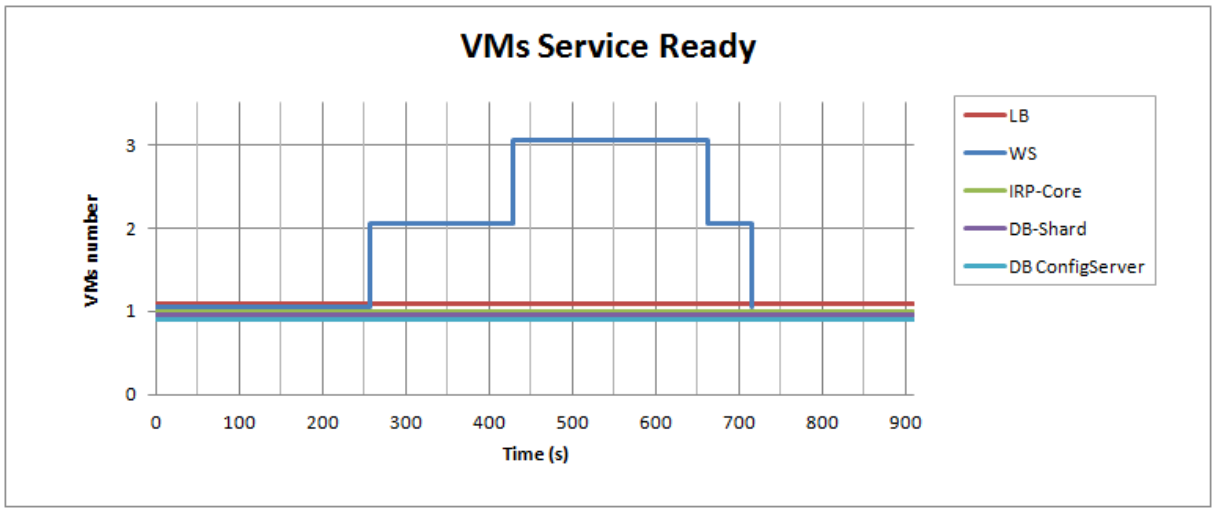

Figure 40 – ANDSFaaS: Scale WS, Trigger Req, Increment 1, Profile 1 (scale out and in)

3.5.2.2.1.1.4 Number of Requests

The Figure 41 depicts the service behaviour during ANDSF scaling in and out operations on the WS tier, showing the total number of requests and the number of requests per WS component.

As a result, it can be seen that service loss is very low and the traffic is split among the WS VMs quite balanced, improving the overall performance.

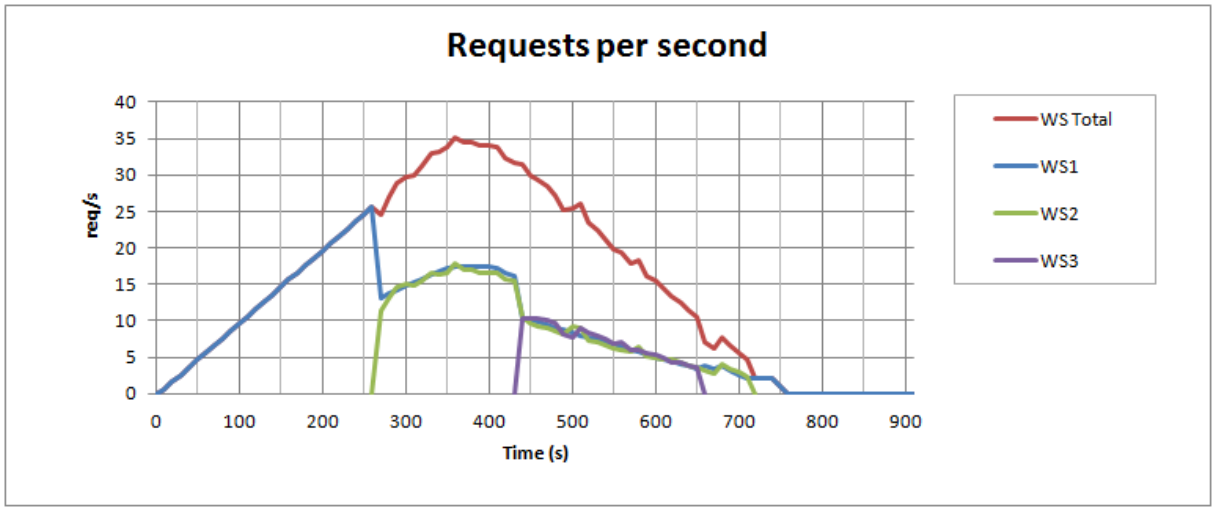

Figure 41 – ANDSFaaS: Scale WS, Trigger Req, Increment 1, Profile 1 (number of requests)

3.5.2.2.1.1.5 Impacts on Physical Machine

The Figures below show the impact of the ANDSFaaS scaling operations on the physical machine resources. In particular, the Figure 42, Figure 43 and Figure 44show the impacts on CPU, RAM and I/O utilization, respectively, along the deployment and release periods.

As a result, it can be seen that the most relevant impact on the physical machine relates to I/O utilization, while the CPU and RAM suffer minor increases. It can be seen that scale-in operations require significantly less resources than scale out.

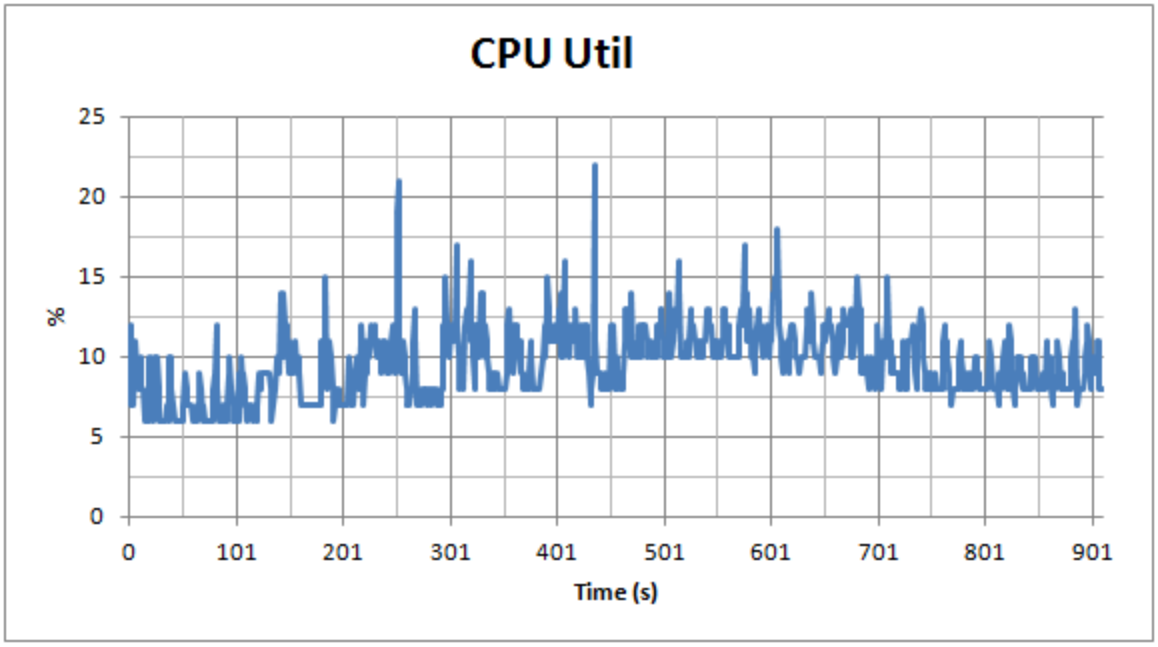

Figure 42 – ANDSFaaS: Scale WS, Trigger Req, Increment 1, Profile 1 (PM CPU utilization)

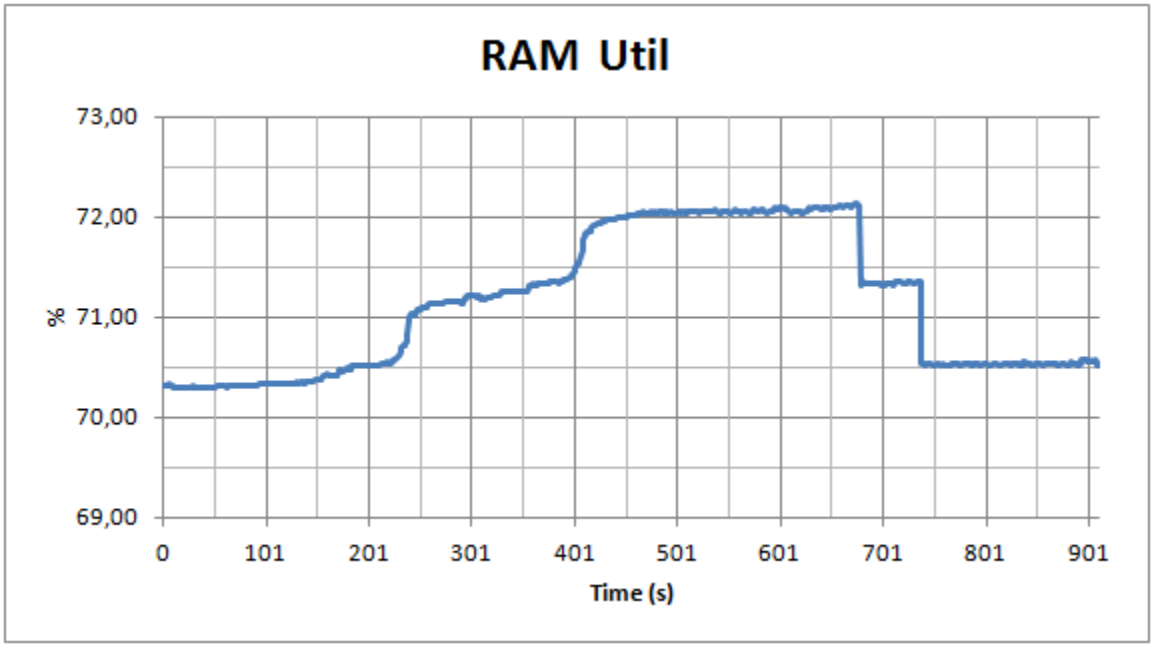

Figure 43 – ANDSFaaS: Scale WS, Trigger Req, Increment 1, Profile 1 (PM RAM utilization)

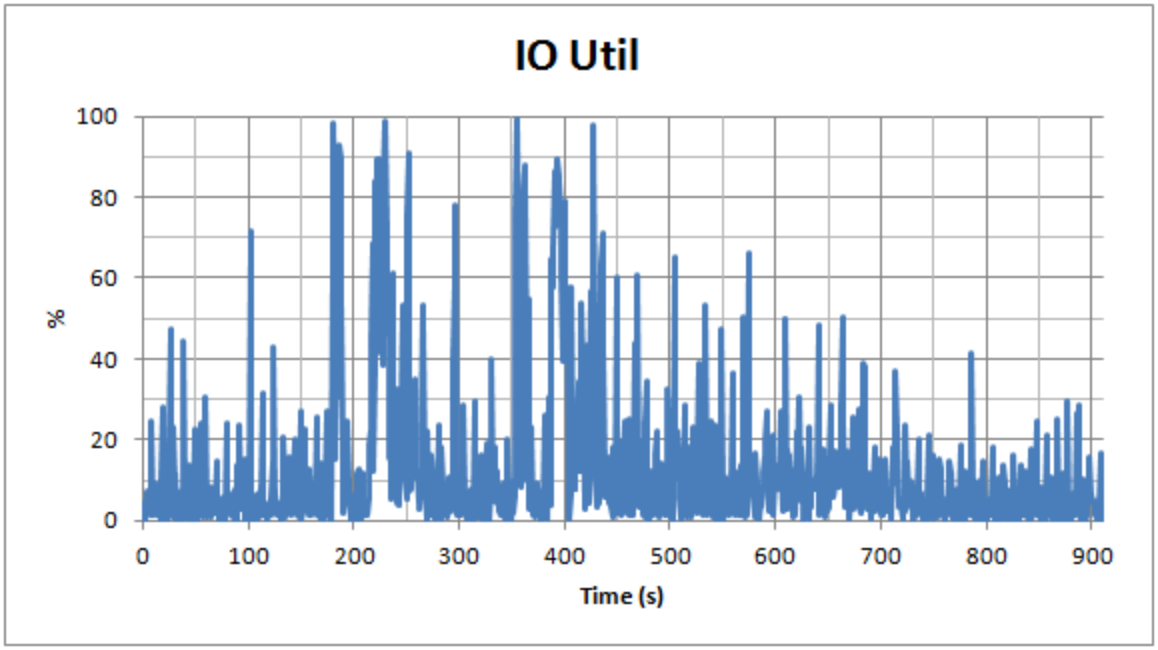

Figure 44 – ANDSFaaS: Scale WS, Trigger Req, Increment 1, Profile 1 (PM I/O utilization)

3.5.2.2.1.2 Other configurations

The Annex B contains the detailed test results obtained by making some parameterization changes to the Base Configuration above.

In particular, the following three changes are evaluated:

- The traffic generation with Profile 2 (less aggressive slope);
- Number of components incremented/decrements per scaling event: +-2 VMs;
- The metric used as trigger is the CPU utilization (CPU);

3.5.2.2.2 IRP Scaling

3.5.2.2.2.1 Base Configuration

This section considers the scaling tests with the following conditions, which is the configuration base for further comparison.

- **Scaling trigger: number of requests (Req);**
- **Scaling increments: +-1 VM;**
- **Traffic generation: Profile 1 (high increasing and decreasing rate).**

3.5.2.2.2.1.1 Scale out

The Figure 45 depicts the time elapsed to scale out an ANDSFaaS instance on the IRP tier, showing the IRP2 and IRP3 instances (IRP1 was already instantiated) and the 3 instantiation steps (OS, VM Ready and Service Ready).

As a result, it can be seen that each scaling out operation takes 125 seconds, 2 minutes and 5 seconds, which seems reasonable. This also means that it takes slightly less time to scale when compared to the WS tier (150 seconds).

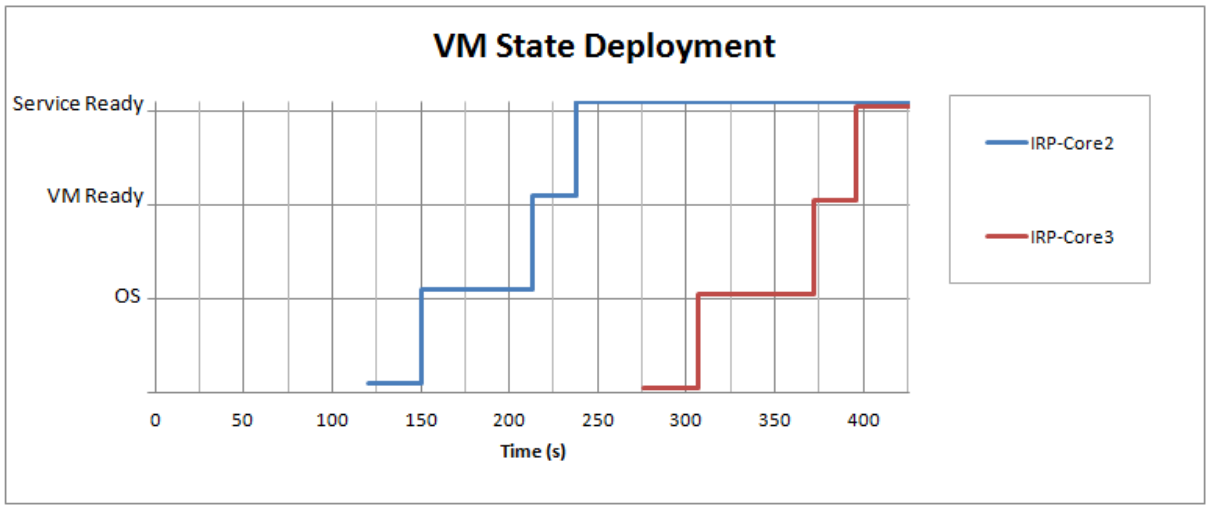

Figure 45 – ANDSFaaS: Scale IRP, Trigger Req, Increment 1, Profile 1 (scale out)

3.5.2.2.2.1.2 Scale in

The Figure 46 depicts the time elapsed to scale in an ANDSFaaS instance on the IRP tier, showing the IRP3 and IRP2 instances being released (IRP1 was already instantiated).

As a result, it can be seen that scaling in operations take 20 seconds which, as expected, is significantly lower compared to the scaling out. It is the same time taken to scale in the WS tier.

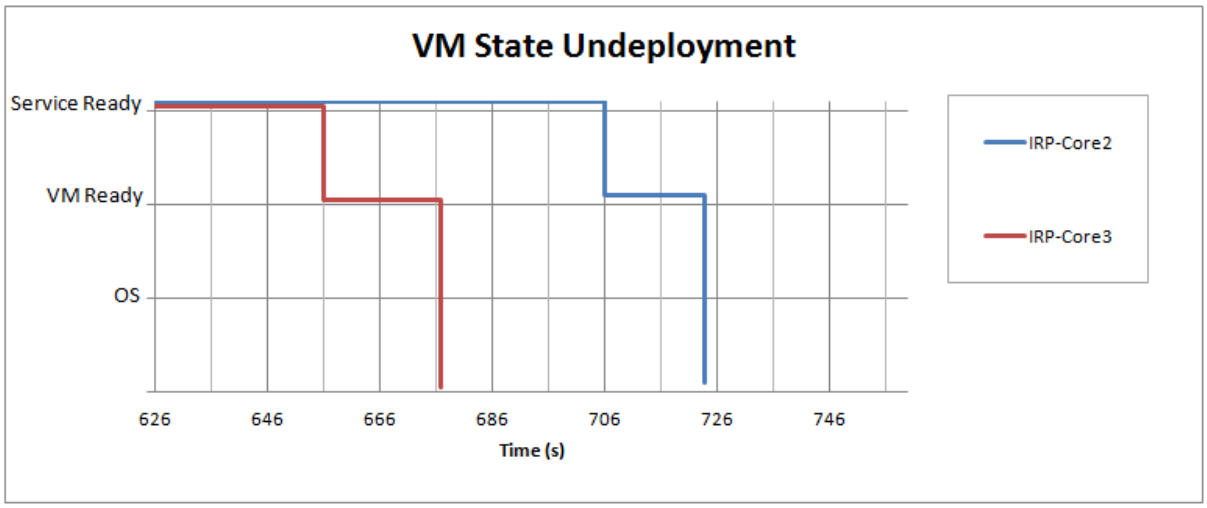

Figure 46 – ANDSFaaS: Scale IRP, Trigger Req, Increment 1, Profile 1 (scale in)

3.5.2.2.2.1.3 Scale out and in

The Figure 47 shows the combination of scaling out and in an ANDSFaaS instance on the IRP tier, as already shown above.

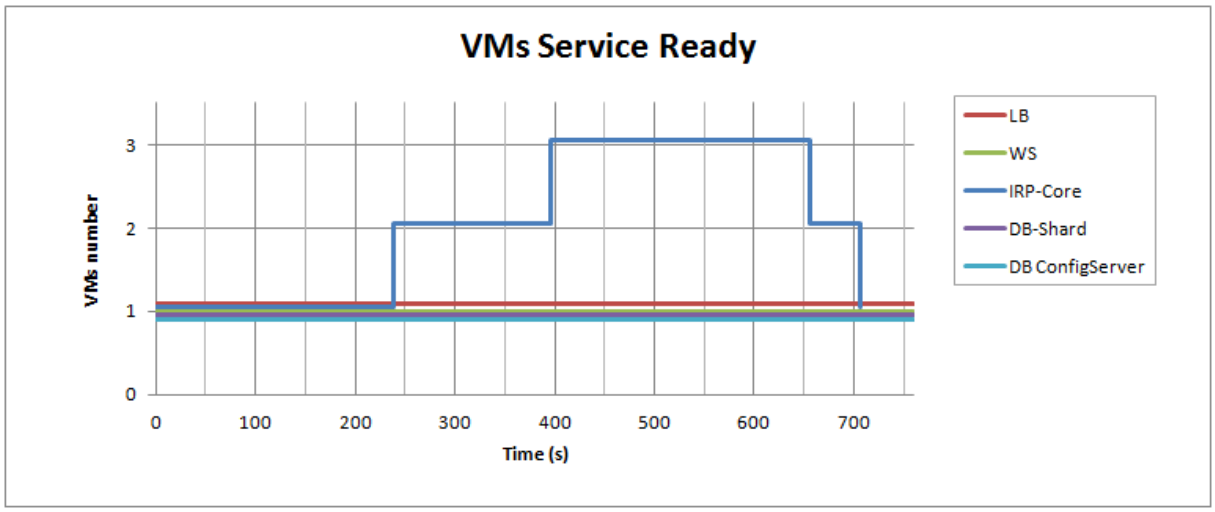

Figure 47 – ANDSFaaS: Scale IRP, Trigger Req, Increment 1, Profile 1 (scale out and in)

3.5.2.2.2.1.4 Number of Requests

The Figure 48 depicts the service behaviour during ANDSF scaling in and out operations on the IRP tier, showing the total number of requests and the number of requests per IRP component.

As a result, it can be seen that service loss is very low and the traffic is split among the IRP VMs quite balanced, improving the overall performance. The level of loss is similar when compared to the one observed in the WS tier, as well as the balancing capabilities.

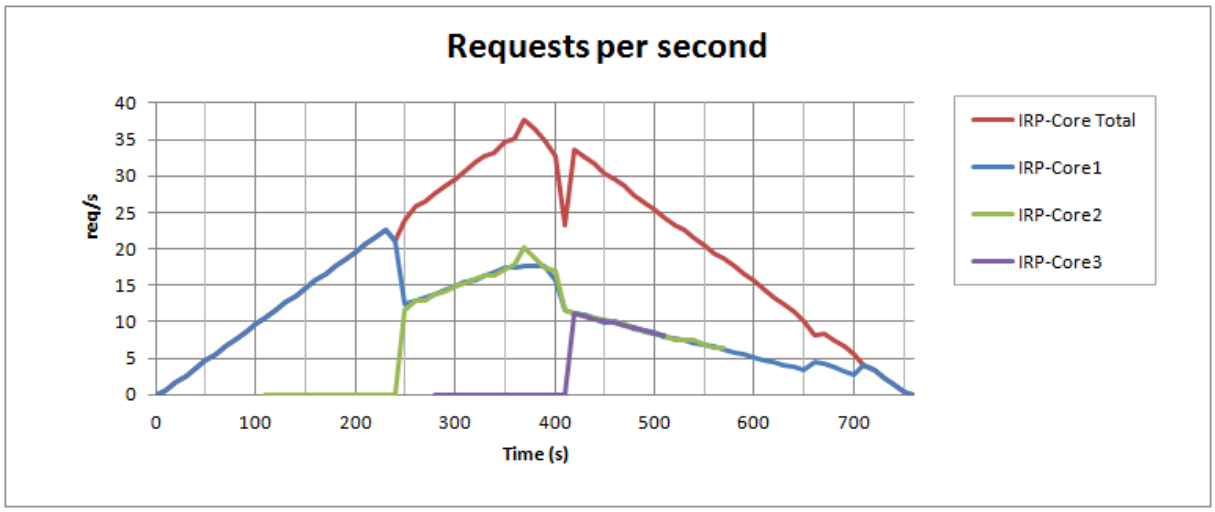

Figure 48 – ANDSFaaS: Scale IRP, Trigger Req, Increment 1, Profile 1 (number of requests)

3.5.2.2.2.1.5 Impacts on Physical Machine

The Figures below show the impact of the ANDSFaaS scaling operations on the physical machine resources. In particular, the Figure 49, Figure 50 and Figure 51 show the impacts on CPU, RAM and I/O utilization, respectively, along the deployment and release periods.

As a result, it can be seen that the most relevant impact on the physical machine relates to I/O utilization (but not that much as for deployment), while the CPU and RAM suffer minor increases. This level of resource consumption is similar to the one observed in the WS tier.

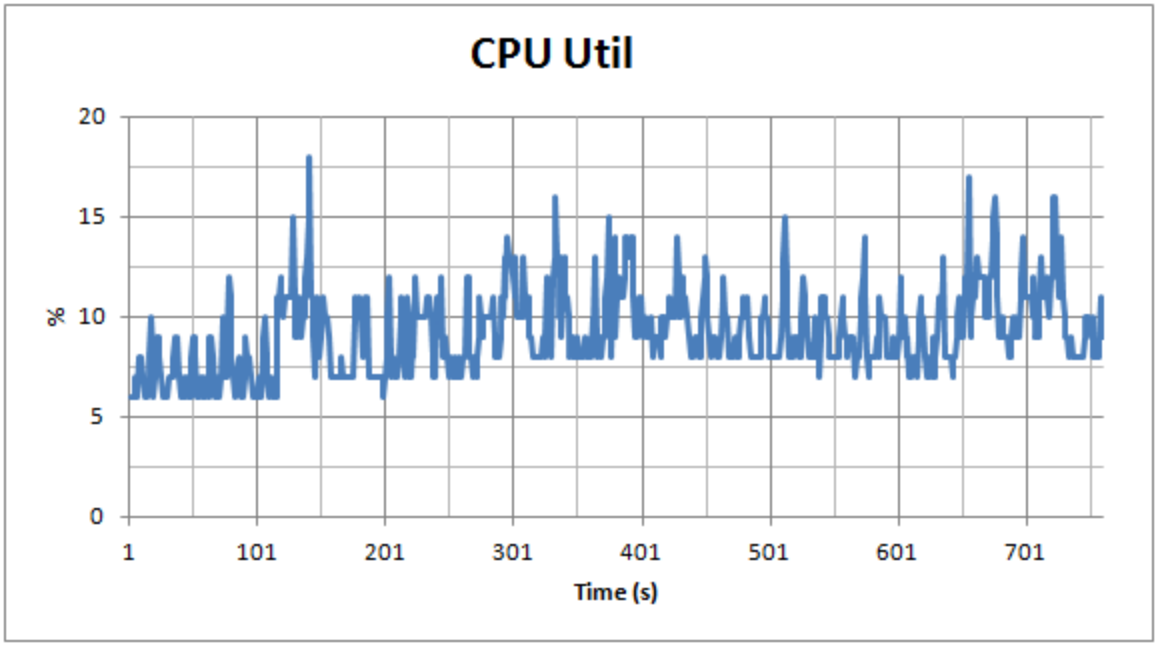

Figure 49 – ANDSFaaS: Scale IRP, Trigger Req, Increment 1, Profile 1 (PM CPU utilization)

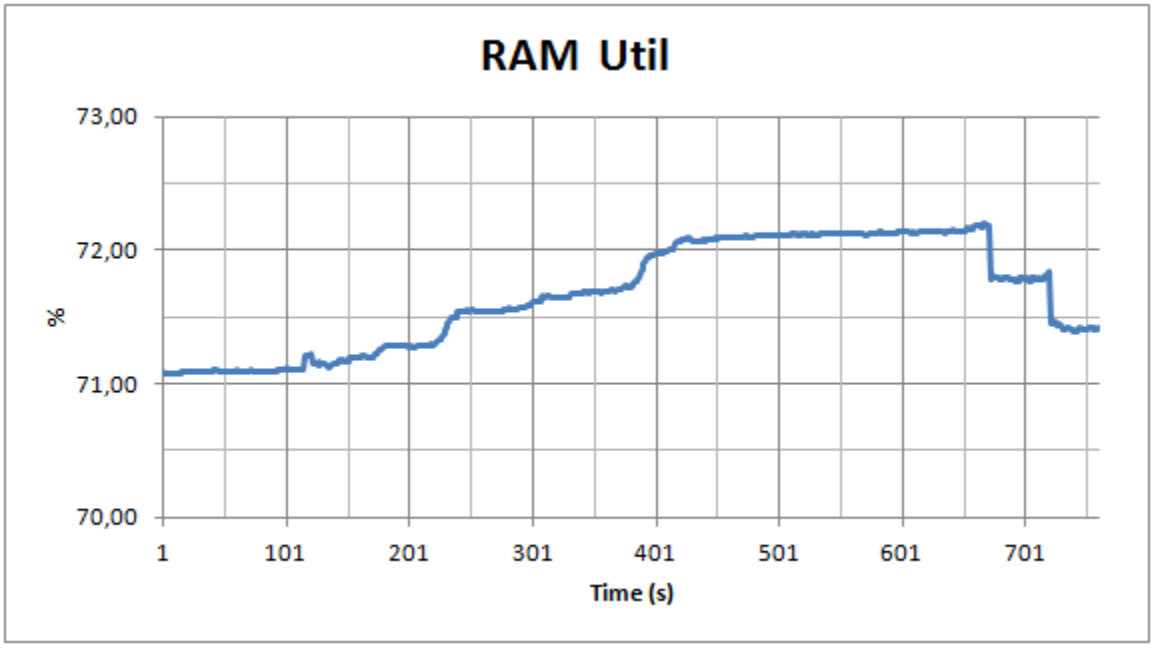

Figure 50 – ANDSFaaS: Scale IRP, Trigger Req, Increment 1, Profile 1 (PM RAM utilization)

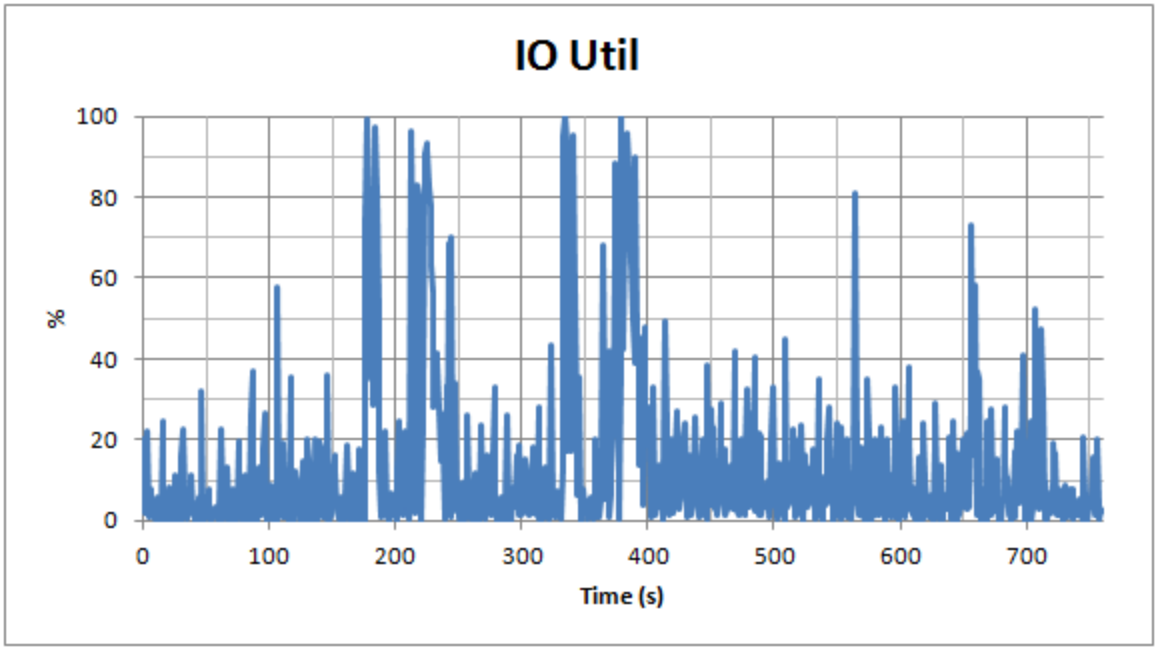

Figure 51 – ANDSFaaS: Scale IRP, Trigger Req, Increment 1, Profile 1 (PM I/O utilization)

3.5.2.2.2.2 Other configurations

The Annex B contains the detailed test results obtained by making some parameterization changes to the Base Configuration above.

In particular, the following three changes are evaluated:

- The traffic generation with Profile 2 (less aggressive slope);
- Number of components incremented/decrements per scaling event: +-2 VMs;
- The metric used as trigger is the CPU utilization (CPU);

3.5.2.2.3 DB Scaling

3.5.2.2.3.1 Base Configuration

This section considers the scaling tests with the following conditions, which is the configuration base for further comparison.

- **Scaling trigger: number of requests (Req);**
- **Scaling increments: +-1 VM;**
- **Traffic generation: Profile 1 (high increasing and decreasing rate).**

Note: For simplicity reasons, to avoid the deployment of a large number of VMs, the DB configuration used in these tests does not support redundancy; shards are not redundant, they only split DB entries.

3.5.2.2.3.1.1 Scale out

The Figure 52 depicts the time elapsed to scale out an ANDSFaaS instance on the DB tier, showing the Shard2 and Shard3 instances (Shard1 was already instantiated) and the 3 instantiation steps (OS, VM Ready and Service Ready).

As a result, it can be seen that each scaling out operation takes 110 seconds, 1 minute 50 seconds, which seems reasonable. This also means that it takes slightly less time to scale when compared to the WS and IRP tiers (150 and 125 seconds, respectively).

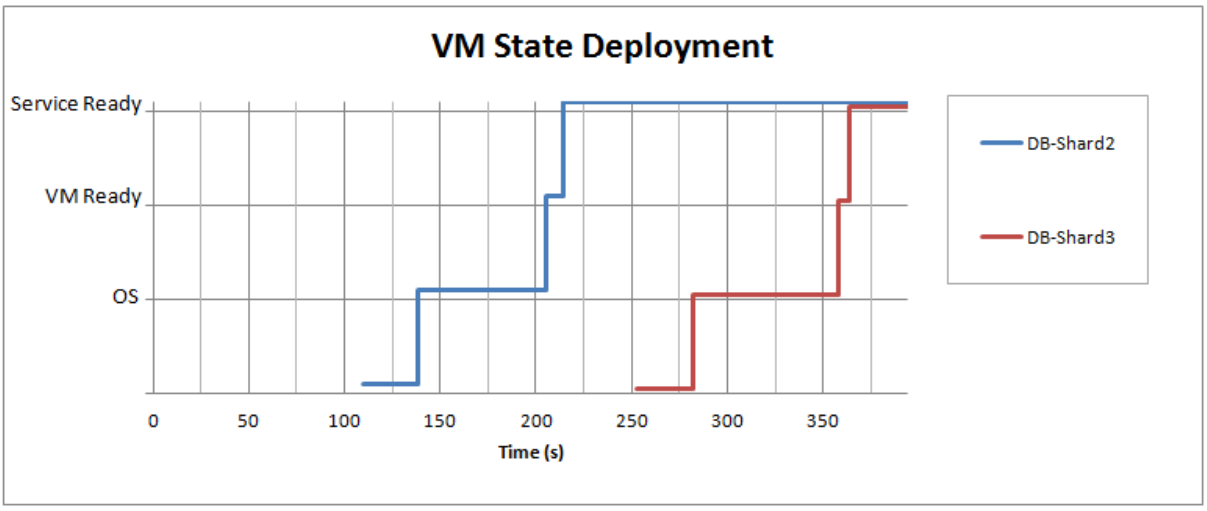

Figure 52 – ANDSFaaS: Scale DB, Trigger Req, Increment 1, Profile 1 (scale out)

3.5.2.2.3.1.2 Scale in

The Figure 53 depicts the time elapsed to scale in an ANDSFaaS instance on the DB tier, showing the Shard3 and Shard2 instances being released (Shard1 was already instantiated).

As a result, it can be seen that scaling in operations take 15 seconds which, as expected, is significantly lower compared to the scaling out. It is less than the time taken to scale in the WS and IRP tiers.

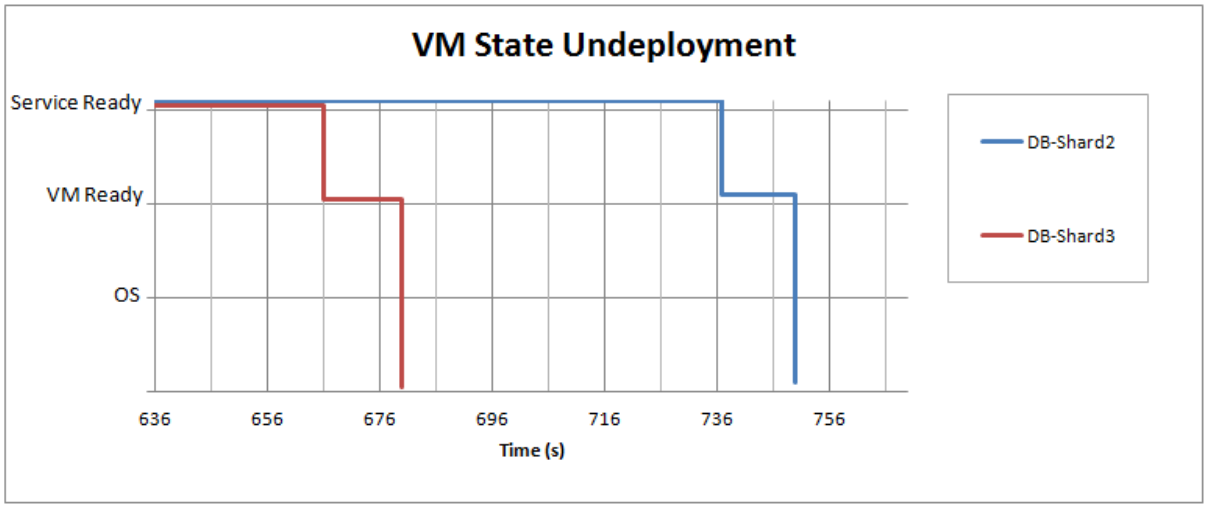

Figure 53 – ANDSFaaS: Scale DB, Trigger Req, Increment 1, Profile 1 (scale in)

3.5.2.2.3.1.3 Scale out and in

The Figure 54 shows the combination of scaling out and in an ANDSFaaS instance on the DB tier, as already shown above.

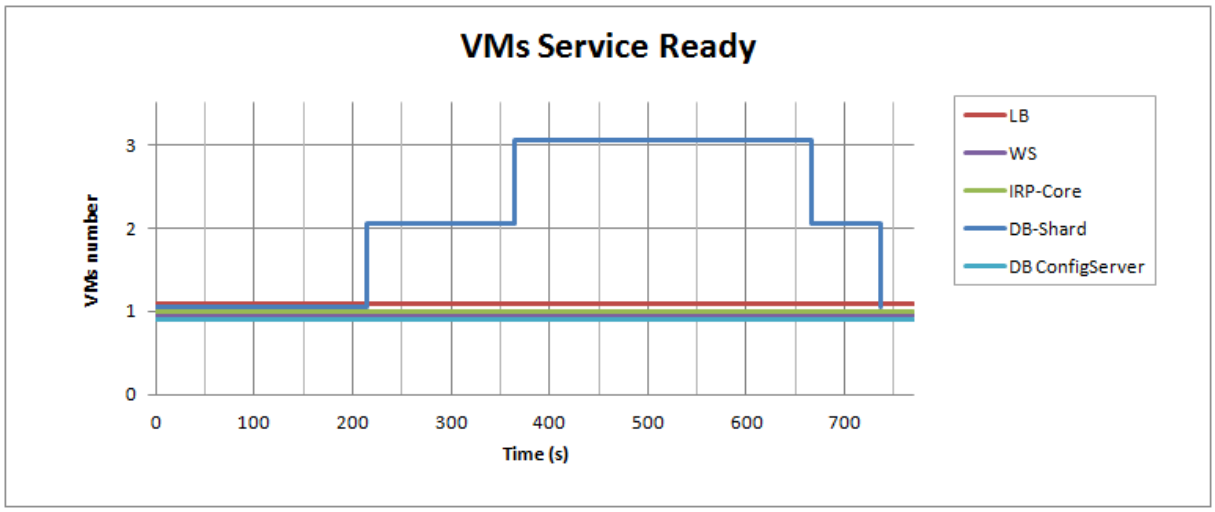

Figure 54 – ANDSFaaS: Scale DB, Trigger Req, Increment 1, Profile 1 (scale out and in)

3.5.2.2.3.1.4 Number of Requests

The Figure 55 depicts the service behaviour during ANDSF scaling in and out operations on the DB tier, showing the total number of requests and the number of requests per Shard component.

As a result, it can be seen that service loss is very low. However, the split of requests among the different Shards is not very balanced (as for WS and IRP tiers), which may have some inefficiencies on performance. Shards launched earlier tend to receive more requests than the others.

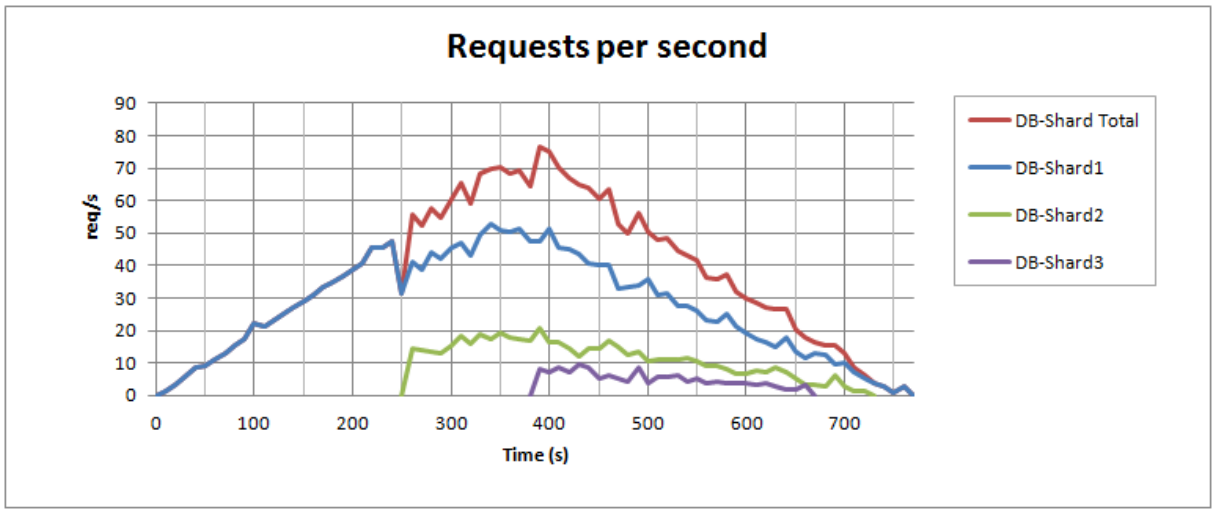

Figure 55 – ANDSFaaS: Scale DB, Trigger Req, Increment 1, Profile 1 (number of requests)

3.5.2.2.3.1.5 Impacts on Physical Machine

The Figures below show the impact of the ANDSFaaS scaling operations on the physical machine resources. In particular, the Figure 56, Figure 57 and Figure 58 show the impacts on CPU, RAM and I/O utilization, respectively, along the deployment and release periods.

As a result, it can be seen that the most relevant impact on the physical machine relates to I/O utilization (but not that much as for deployment), while the CPU and RAM suffer minor increases. This level of resource consumption is similar to the one observed in the WS and IRP tiers.

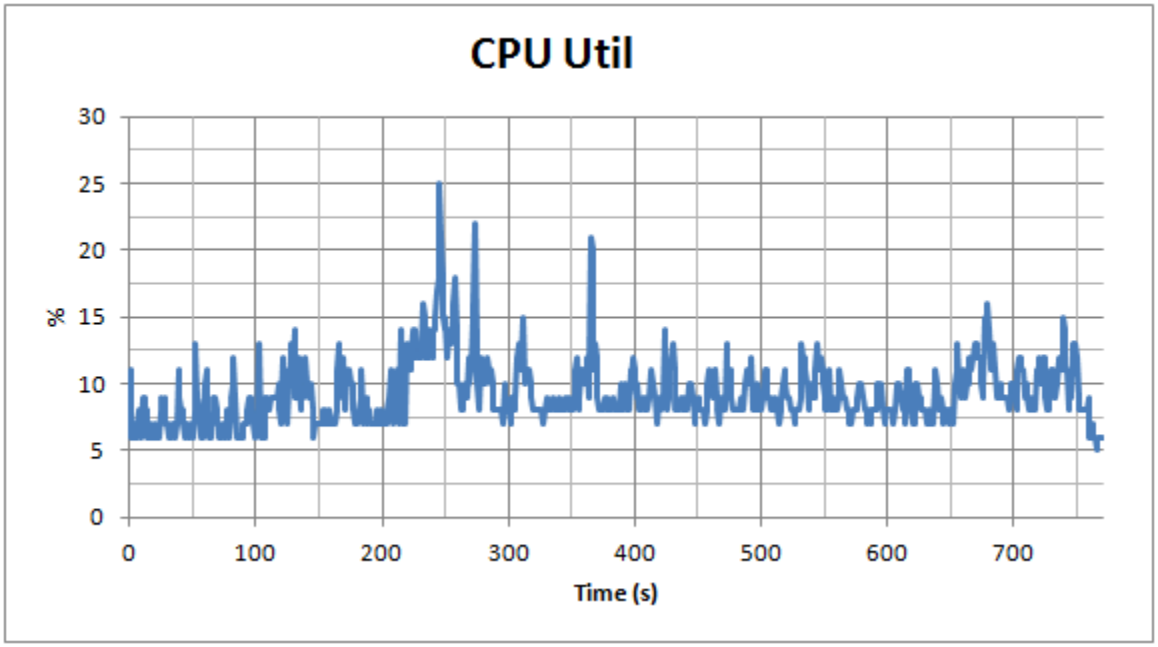

Figure 56 – ANDSFaaS: Scale DB, Trigger Req, Increment 1, Profile 1 (PM CPU utilization)

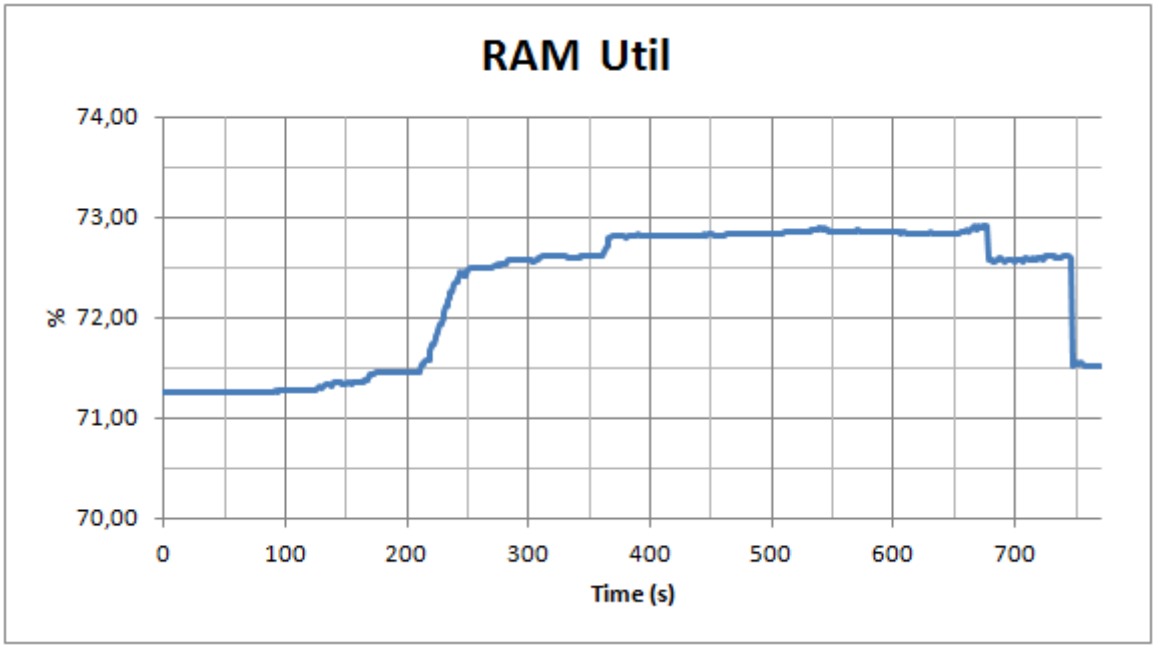

Figure 57 – ANDSFaaS: Scale DB, Trigger Req, Increment 1, Profile 1 (PM RAM utilization)

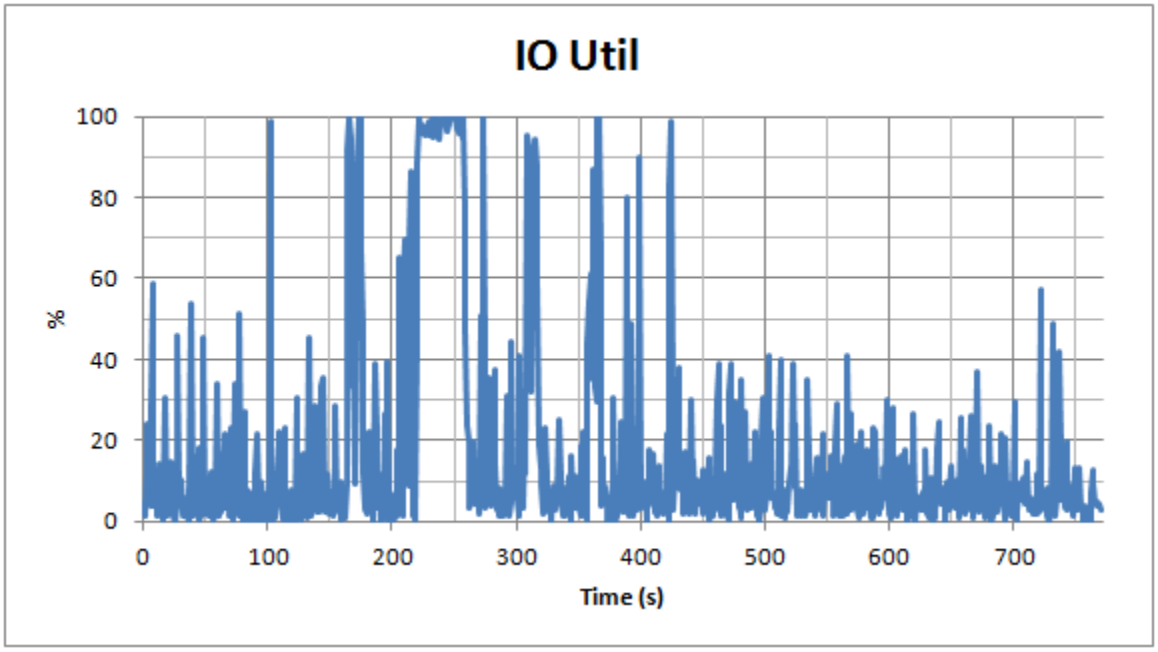

Figure 58 – ANDSFaaS: Scale DB, Trigger Req, Increment 1, Profile 1 (PM I/O utilization)

The section 6.3 contains the detailed test results obtained by making some parameterization changes to the Base Configuration above.

In particular, the following three changes are evaluated:

- The traffic generation with Profile 2 (less aggressive slope);
- Number of components incremented/decrements per scaling event: +-2 VMs;
- The metric used as trigger is the CPU utilization (CPU);

3.5.2.3 Conclusions

As overall conclusions, the scaling test results show that the scaling (out and in) operations work as expected for the 3 tiers, increasing and decreasing the resources allocated to the service, based on monitoring, in a reasonable period of time. It can also be observed that the impact of scaling on the physical machine is not significant, not constituting a major burden for the service.

The scaling behaviour for the 3 tiers is very similar, although on the DB tier some issues are slightly different due to database specific issues.

Scaling out and scaling in operations takes, in average, around 2-3 minutes and 20-30 seconds, respectively, with minor differences among different tiers. For all tiers, the loss during the operations is quite reduced, imposing very low (but not null) downtime periods. This is not significant assuming that scaling operations does not occur so often (e.g. every 10 minutes). However, for general purpose (e.g. for a few a day, for example, morning/afternoon/night profiles) it seems reasonable.

The use of different profiles (simulated rate of number of requests from clients) show that the service reacts differently to the rate of increase and decrease of requests from clients. Scaling out operations take place earlier when the slope curve of number of requests increase more aggressively. The same occurs with scaling in, releasing later resources when the number of requests decreases slowly.

The possibility to use different increment values (number of components/VMs added/released per scaling event) was also tested. It was observed that the time to scale out/in 2 simultaneous VMs is not significantly higher than scaling out/in only 1. For this reason, it was demonstrated that the use of higher increments can benefit when the service needs to be scaled very quickly (e.g. when the target deployment configuration is far away from the current deployment).

The impacts of scaling operation on the physical resources are, in general, not very relevant, especially when compared with the instantiation results, which are significantly higher. The I/O is the resource with a most significant increase, although not comparable to instantiation. CPU and RAM are only slightly affected. Scaling out operations are, as expected, the heaviest operations, while scaling in are shorter and less resource consuming, in particular for I/O utilization.

3.5.3 Test 3 – ANDSFaaS Reliability

This test intends to evaluate the reliability of the ANDSFaaS service. For this purpose, tests will simulate the failure of 1 component (VM), checking whether the service recovers and how much time it takes. The SO is not involved in this recovery; only the components itself have to react to failures.

3.5.3.1 Test Description

In order to test the service reliability, the service will be running normally after an initial deployment using multiple components (VMs) per tier, in order to assure load balancing, but especially in order to provide high availability, ensuring that the failure of one component can be solved by the remaining components of the tier. The initial deployment used is as follows.

Table 18 – Deployment Configuration

Configuration	VMs per Tier
Deployment 1	LB – 1 VM WS – 3 VMs IRP – 3 VMs DB – 4 VMs

During the tests, a constant request rate of 35 Req/s will be sent to the ANDSFaaS. That will map to the following number of requests (Req) per tier. (it maps to 35 Req/s for LB, WS and IRP and 70 to DB, since each UE request requires 2 accesses to the database).

Table 19 – Load Configuration

Tier	Constant rate
LB	35 Req/s
WS	35 Req/s
IRP	35 Req/s
DB	70 Req/s

The simulation of a component failure will be done by turning down its attachment to the network. The recovery is simulated by turning on again the network.

3.5.3.2 Test Results

The sections below show the results obtained for the different tier reliability.

3.5.3.2.1 WS Reliability

This section shows the reliability results for the WS tier.

3.5.3.2.1.1 WS Requests

The Figure 59 shows the behaviour of the WS tier when 1 of the 3 component that comprise this tier fails. In that case, the WS3 is turned down by purpose, making the WS service to rely only on the remaining 2 WS components. After some time (around 1 minute), the WS3 is turned on again.

As a result, it can be observed that the total WS service just lost some packets during around 5 seconds, and the total lost was less than 1/3 of the total amount of requests (around 27 in 35 Req/s). Looking into the WS component service, it can be noticed that initially the traffic is equally distributed by the 3 WS components, starts to be split equally by the remaining 2 WS components (the WS3 serves 0 Req/s), which is an excellent result.

After 1 minute, the WS3 is turned on and the WS service does not suffer any loss (only some delay until WS service is stabilized). After this point, the WS service starts to be load balanced again among the 3 WS components as before the failure.

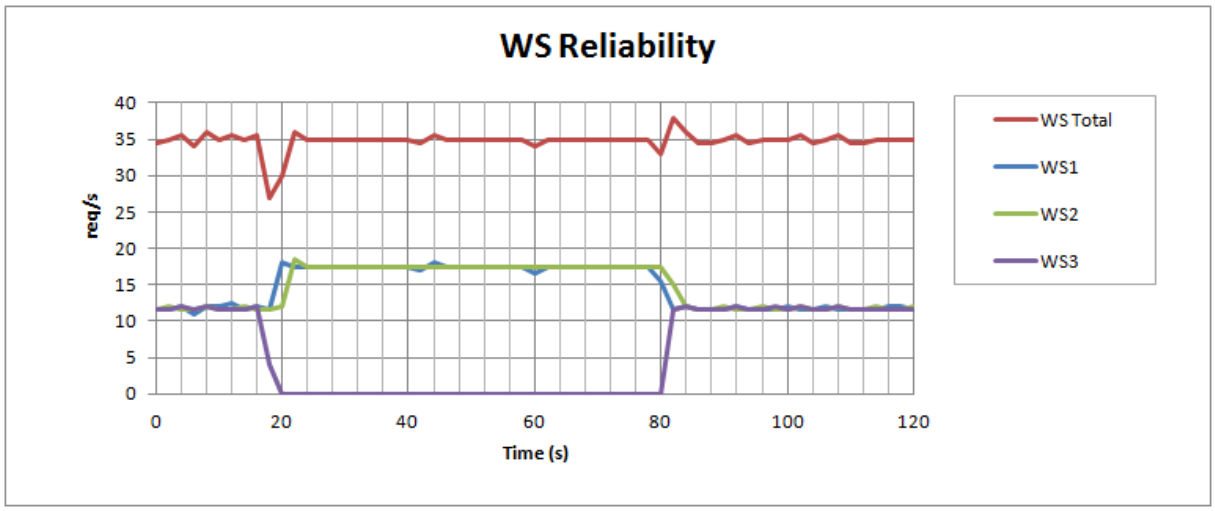

Figure 59 – ANDSFaaS: Reliability WS (WS Requests)

3.5.3.2.1.2 Service Requests

The Figure 60 shows the comparison between the service requests and the number of requests handled and responded by the WS tier overall. In this case, it can be seen that there is no significant difference, i.e. minor loss. It can be noticed only some delay peaks on the ANDSFaaS service.

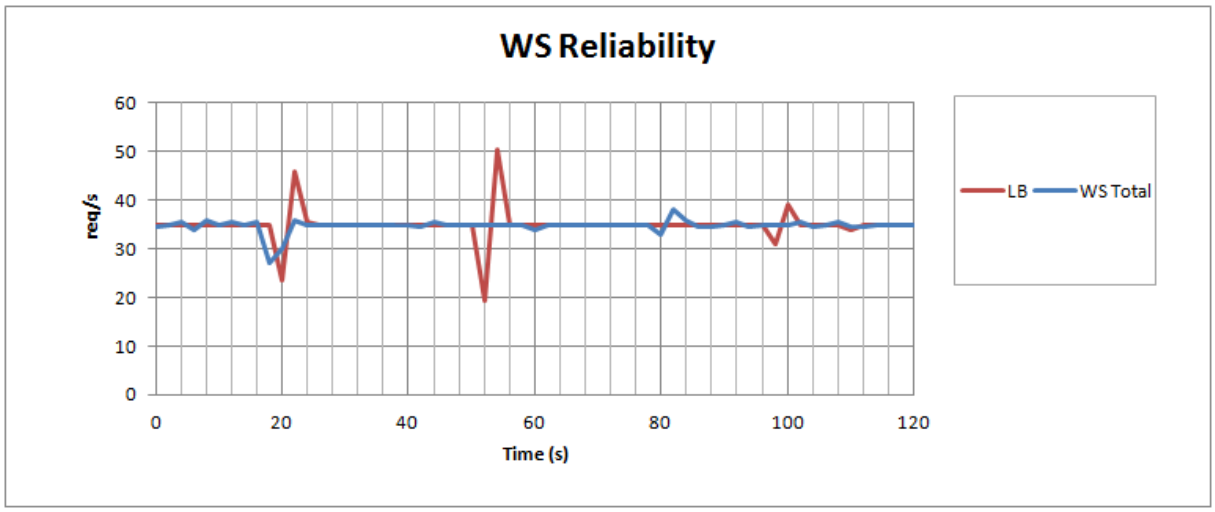

Figure 60 – ANDSFaaS: Reliability WS (Service Requests)

3.5.3.2.2 IRP Reliability

This section shows the reliability results for the IRP tier.

3.5.3.2.2.1 IRP Requests

The Figure 61 shows the behaviour of the IRP tier when 1 of the 3 component that comprise this tiers fails. In this case, the IRP3 is turned down by purpose, making the IRP service to rely only on the remaining 2 IRP components. After some time (around 1 minute), the IRP3 is turned on again.

As a result, it can be observed that the total WS service loses about 1/3 of requests during around 10 seconds. After these 10 seconds, the service recovers completely and starts splitting the traffic by the remaining 2 IRP components in a balanced way. This shows that the ANDSFaaS service is more sensible to failures in IRP than in WS components. IRP takes more time to detect and recover from failures.

After 1 minute, the WS3 is turned on, and the IRP service does not suffer any loss (although a significant delay is introduced until the WS service is stabilized). After this point, the IRP service starts to be balanced again among the 3 IRP components as before the failure.

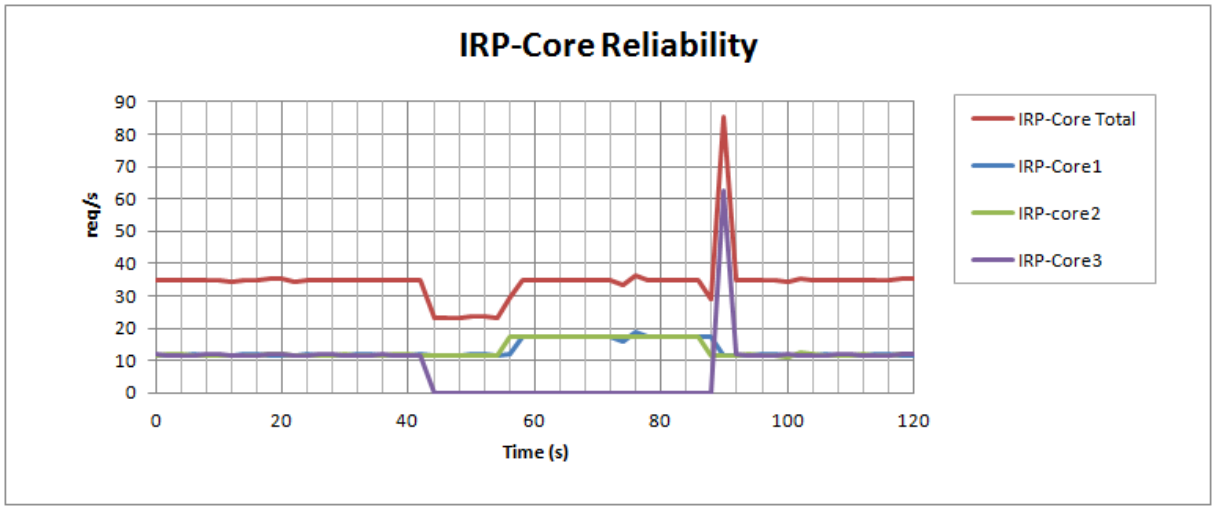

Figure 61 – ANDSFaaS: Reliability IRP (IRP Requests)

3.5.3.2.2.2 Service Requests

The Figure 62 shows the comparison between the service requests and the number of requests handled and responded by the IRP tier overall. It can be seen that during 10 seconds around 1/3 of requests are not responded.

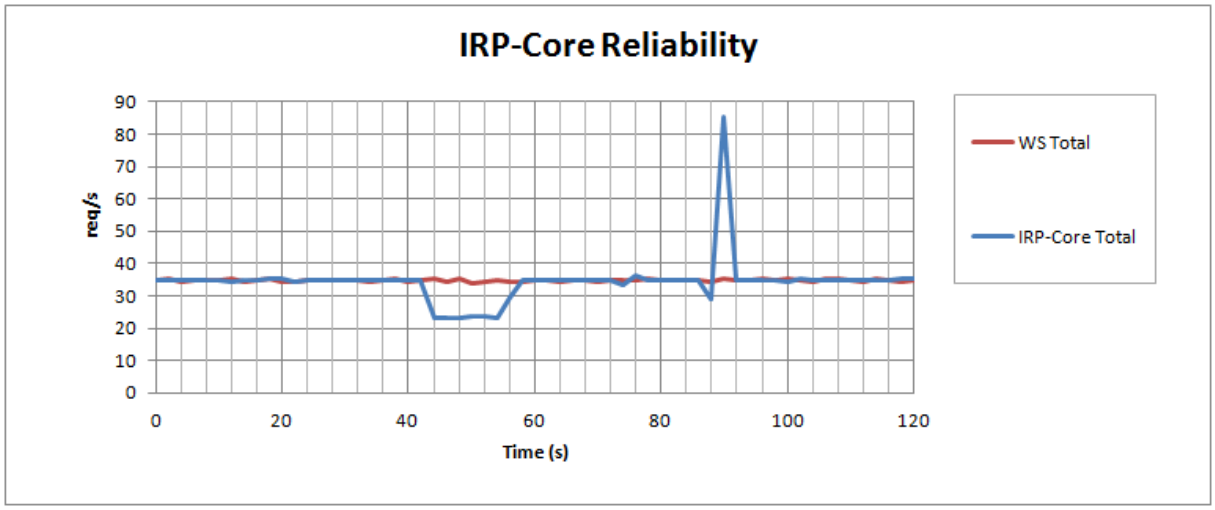

Figure 62 – ANDSFaaS: Reliability IRP (Service Requests)

3.5.3.2.3 DB Reliability

This section shows the reliability results for the DB tier.

3.5.3.2.3.1 DB Requests

The DB (database) is a special tier, since it comprises components with different roles. Here we have replica-sets, which are redundant repositories and can be active (primary) or backup (secondary). Additionally, we have the ConfigServer role, which is responsible for the overall management (and is not directly involved in the traffic path). The following Figures (Figure 63, Figure 64 and Figure 65) show the behaviour of the DB tier when each of those roles fail.

As a result, it can be observed that the failure of the primary replica-set (Figure 63) leads the DB service to a failure of 30 seconds. At that point, the secondary takes over and becomes primary, making the DB service to become up again. After 1 minute, when the failed component is turned back on again, the old primary becomes secondary, not resulting in any loss.

When a secondary fails (Figure 64), as expected, there is no impact on the service, and the same happens when the ConfigServer fails (Figure 65).

Note that 30 seconds seems a large period of time. However, it results from the recommended redundancy MongoDB configurations, for safety reasons. More aggressive configurations could make 2 replica-sets to become primary, and lead to integrity problems.

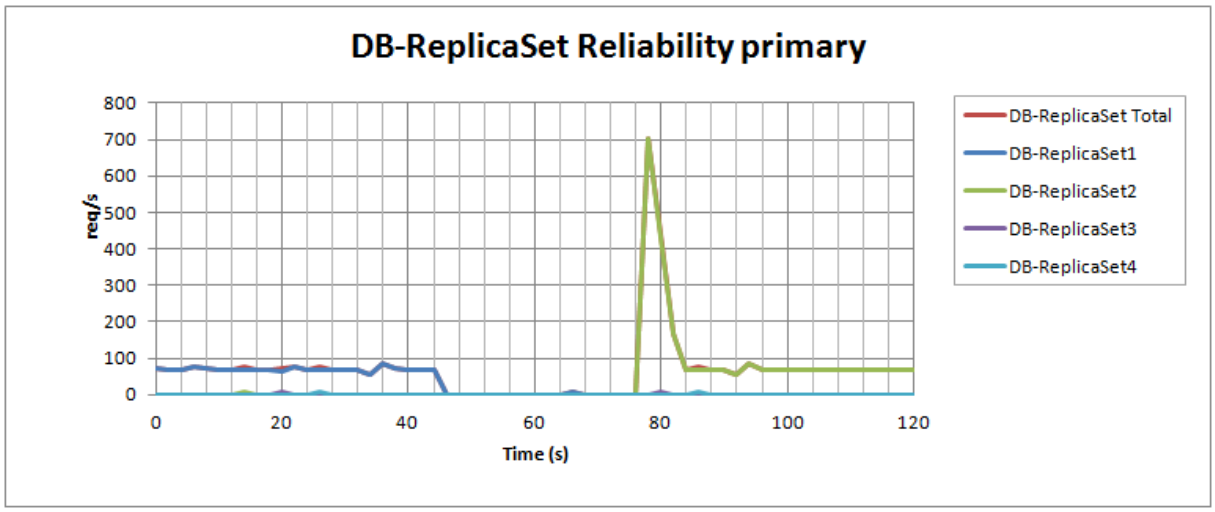

Figure 63 – ANDSFaaS: Reliability DB Primary (DB Requests)

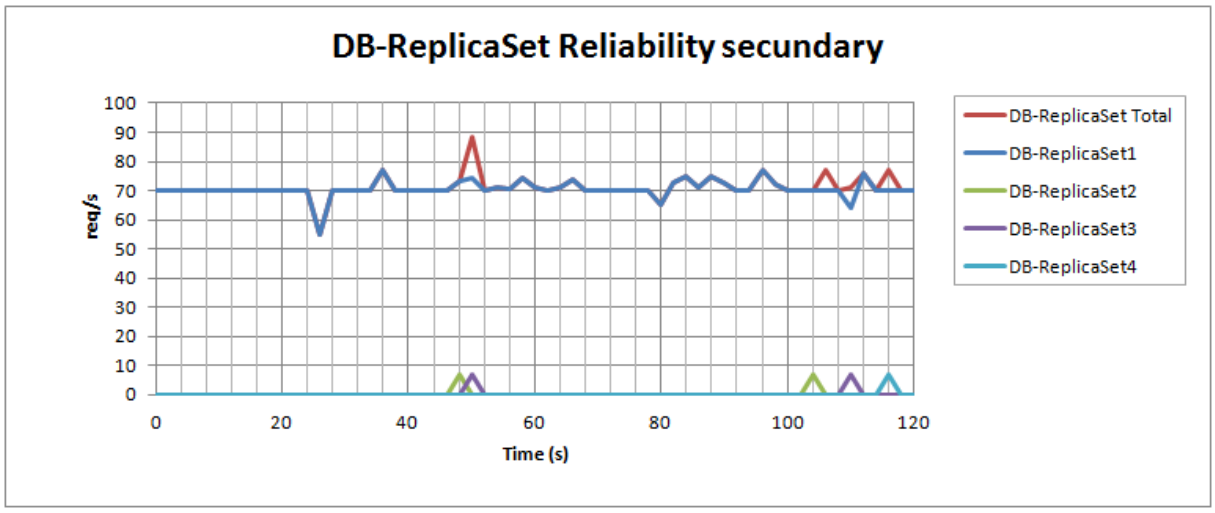

Figure 64 – ANDSFaaS: Reliability DB Secondary (DB Requests)

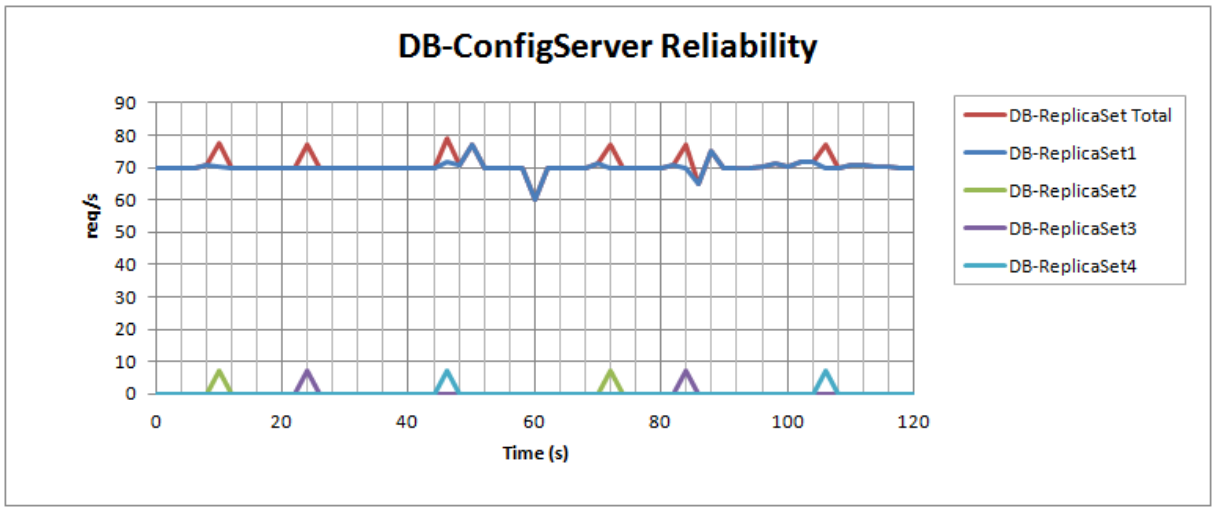

Figure 65 – ANDSFaaS: Reliability DB ConfigServer (DB Requests)

3.5.3.2.3.2 Service Requests

The following Figures (Figure 66 and Figure 67) show the comparison between the service requests and the number of requests handled and responded by the DB tier overall. It can be seen that when the primary fails the IRP tier, and therefore the ANDSFaaS service stops.

When the secondary fails, there are no impacts on the service. The same happens for the Config Server (not shown in a chart since it does not handle any traffic).

Note: Remember that DB requests are 2x Req since are required 2 DB accesses pr UE request.

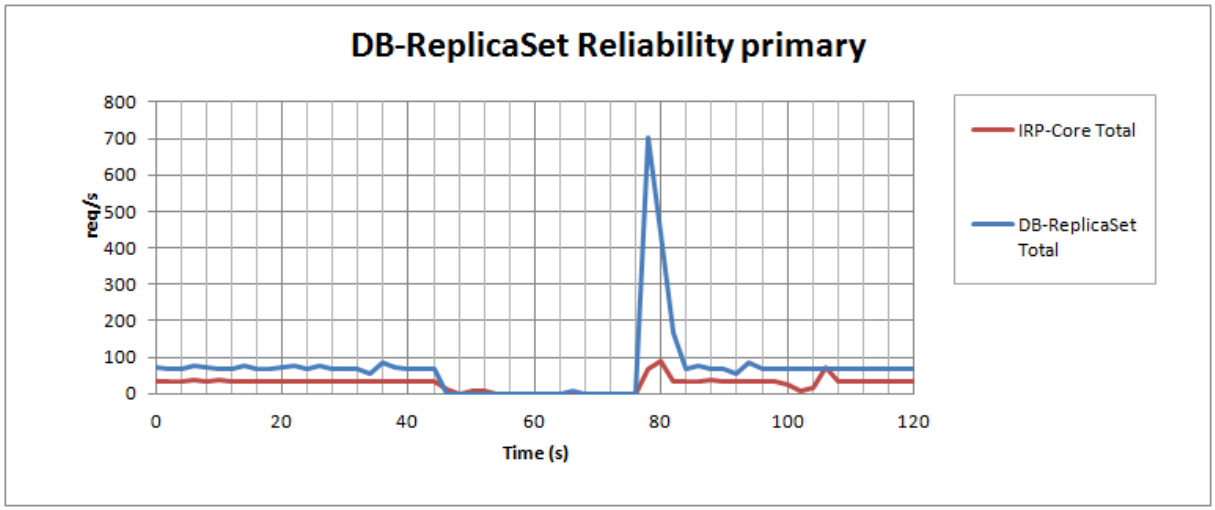

Figure 66 – ANDSFaaS: Reliability DB Primary (Service Requests)

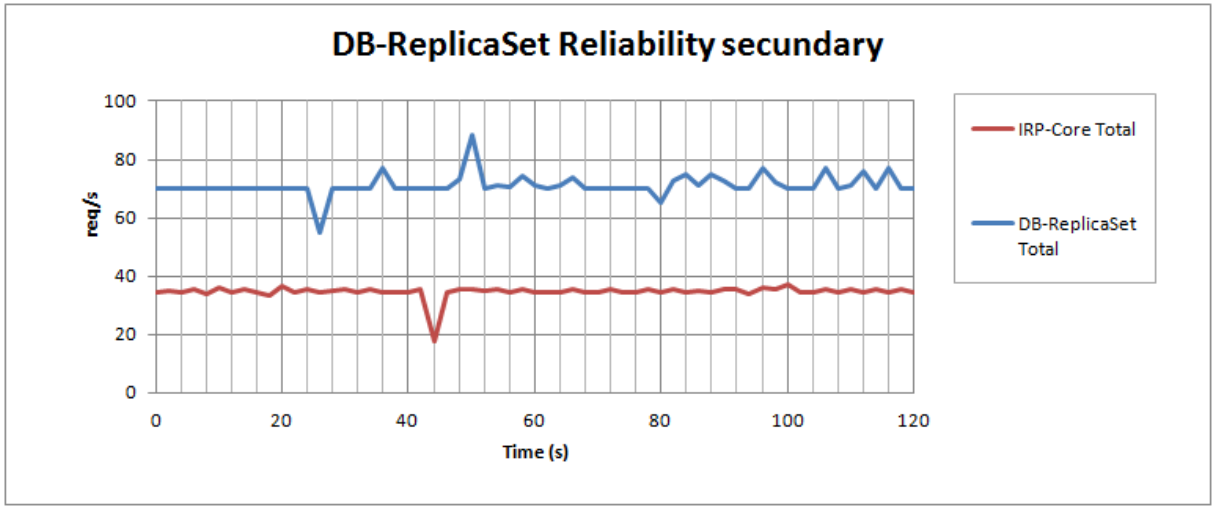

Figure 67 – ANDSFaaS: Reliability DB Secondary (Service Requests)

3.5.3.3 Conclusions

As overall conclusions, the reliability test results show that the reliability of the ANDSFaaS service is effective, being able to recover automatically. However, it cannot avoid the partial of complete loss of the service during some period of time. The tests already show that there are significant differences between failures occurred on components belonging to different tiers.

On the WS tier, it can be observed that the service is able to recover from a component failure. The service partially fails during around 5 seconds, losing basically the requests that are expected to be served by the failing component. After that time, the LB detects the failure and reacts to it, disregarding that component. When after some time the failed component recovers, it is used back again. On the component recovery, there is no loss observed.

On the IRP tier, it can be observed that the service is able to recover from a component failure. The service partially fails during around 10 seconds, losing basically the requests that are expected to be served by the failing component. After that time, the WS tier components detect the failure and react to it, disregarding that component. When after some time the failed component recovers, it is used back again. On the component recovery, there is no loss observed.

On the DB tier, it can be observed that the service is able to recover from a component failure. Depending on the role of the failing component, the results are quite different. When the failing component is a Config Server (shard management) or a secondary shard (backup), no impacts (loss) are observed. However, when the primary shard (active) fails, a complete loss occurs for a period around 30 seconds and no read/write operations are possible. The reason for this high value is because, for integrity reasons, the secondary shard needs to be sure that the primary shard is really down and he can take over safely. When after some time the failed component recovers, it is used back again. When the failed primary shard recovers it becomes secondary and is not active anymore.

3.6 Conclusions

As overall conclusions, the performed evaluation tests shown that the ANDSFaaS service has a very good properties for cloud environment, namely regarding on-demand, agility/flexibility and high availability. The ANDSF was evaluated in the following aspects: Deployment, Scaling and Reliability.

Regarding **Deployment**, it can be observed that the number of nodes used for an initial instantiation has impacts on the required time to deploy an ANDSFaaS service-ready instance. Instantiations with more nodes take more time; however, this time seems linear.

Typical instantiation times range from 5 to 10 minutes, which is a reasonable time, considering the used physical machine. The service disposal times ranged from 1 to 2 minutes.

Deployment operations impose significant impacts on the physical machines resources, especially on service instantiation. The I/O utilization is the most impacted resource, while CPU and RAM also noticed a slight impact.

Regarding **Scaling**, it can be observed that these operations work properly for the 3 scalable tiers, increasing and decreasing the resources allocated to the service, based on monitoring, in a reasonable period of time. The behaviour for the different tiers is very similar, although for the DB tier is slightly different. Scaling of multiple tiers cannot take place simultaneously for safety reasons.

The scaling out and scaling in operations take, in average, around 2-3 minutes and 20-30 seconds, respectively, with minor differences among the different tiers. For all tiers, the loss during the operations is quite reduced, imposing very low (but not null) downtime periods. These seem reasonable times for general purposes scaling.

The combinations of multiple configuration parameters show that scaling based on different metrics are possible (e.g. number of requests or CPU utilization), providing similar behaviour. This, together with the service load, defines when the scaling operations are triggered. It is also possible to configure

the number of components to add/remove per scaling event. Using this feature, each time a scaling is triggered N components (VMs) are created or released. This can help to adapt the service more quickly to a given service target configuration.

It can also be observed that the impact of scaling on the physical machine is not significant, not representing a major burden for the physical machine. The I/O is the more affected resource; however, quite less when compared to deployment operations.

Regarding **Reliability** issues, it can be observed that the ANDSFaaS service is able to automatically recover from component failures of 1 component. This is true for the 3 tiers; however, the behaviour is different from tier to tier. Some tiers take more time to recover and suffer more losses than others. When the failed component recovers there is no loss verified.

On the WS tier, it can be observed that the service partially fails during around 5 seconds, losing basically the requests that are expected to be served by the failing component. After that time, the LB detects the failure and reacts to it, disregarding that component. When the failed component recovers, it is used back again.

On the IRP tier, it can be observed that the service partially fails during around 10 seconds, losing basically the requests that are expected to be served by the failing component. After that time, the WS tier components detect the failure and react to it, disregarding that component and when the failed component recovers, it is used back again.

On the DB tier, it can be observed that the behaviour depends on the role of the failed component. When the failing component is a Config Server (shard management) or a secondary shard (backup), no impacts (loss) are observed. However, when the primary shard (active) fails, a complete loss occurs for a period around 30 seconds and no read/write operations are possible. When the failed component recovers, it becomes a secondary shard not being used as active anymore.

4 MOBaaS Evaluation

4.1 Updates from D4.4

4.1.1 MOBaaS

This section explains the innovations in MOBaaS that have been produced after the previous deliverables [1][2][8] . The following innovations have been addressed and are described in this deliverable:

- **The mobility prediction algorithm.** A single user mobility prediction algorithm has been presented in the previous deliverables. In this deliverable, we have extended our previous work and included multiple user and group user prediction algorithms, which are required to integrate MOBaaS with ICNaaS and EPCaaS.
- **Interface between MOBaaS and other MCN services.** In previous deliverable D4.4, the interface between MOBaaS and other MCN services has been designed as a customized HTTP web service. This has been refined into a standard web service using the existing flask framework.
- **MOBaaS SM/SO.** Open Cloud Computing Interface (OCCI) has been adopted for managing external communications towards SM, as well as, inter-communication between SM and SO, which are now consistent with the ecosystem of the project and easy to use.

4.1.2 Mobility prediction

Mobility prediction provided by MOBaaS [1] can be used in two following ways.

- **Single user and multiple user prediction:** MOBaaS runs an HTTP server that constantly waits for consumer requests. In the single user prediction mode, estimation is based on the user's current location (cell ID) at a given time (time and day). In the multiple user case, when the consumer asks for a given time and day in the request message, the service makes prediction on cells where given users may be located in the future. Moreover, MOBaaS can provide details about the movement of users (the cell IDs that the users were moving).

ICNaaS could use multi user mobility prediction to decide on the content relocation. Based on the network topology, the content may be relocated to CCNx routers/repositories, which are placed on the eNodeBs and/or the S/P-GWs (Serving/PDN-Gateway). When a user moves from one cell (source eNodeB) to another cell (target eNodeB), which are served by the same S/P-GWs, the content may be relocated between the CCNx routers/repositories implemented on eNodeBs. If the source and target eNodeB are not served by the same S/P-GWs, the content may be moved between the CCNx routers/repositories which are placed on target S/PGW.

- **Group user prediction:** In group user prediction a trigger-based approach as a notification mechanism for the generated information is used. The group user prediction aims to provide a probability distribution of the number of users in a certain cell at a given moment (e.g., $t+\delta$, $t+2\delta$, ...), knowing the probability distribution of location(s) of users at time (t). Equation 1 defines the probability that (M) users may visit cell (C_i) at time ($t+\delta$). Parameters of this equation are listed in Table 20.

Table 20 - Group user prediction parameters

Parameter Name	Parameter Definition
$C = \{C_1, C_2, .., C_i\}, \lvert C\rvert = I$	Set of cells
$U = \{U_1, U_2, .., U_j\}, \lvert U\rvert = J$	Set of users
$N_{C_i}(t)$	Number of users in cell C_i at time *(t)* (*e.g.,* m)
$N_{C_i}(t+\delta)$	Number of users in cell C_i at time *(t+δ)* (*e.g.,* M)
n_1	Number of users that may move to C_i at time *(t+δ)*
n_2	Number of users that may move from C_i at time *(t+δ)*
$\triangle m = M - m = n_1 - n_2$	The difference between number of users in cell C_i at time *(t)* and *(t+δ)*
$F_{C_{i'}(t)}$	Subset of all users out of C_i at time *(t)*
$F_{C_i(t)}$	Subset of all users in C_i at time *(t)*
P_{j_1} , U_{j_1} ∈J	$P\{U_{j_1}\ is\ in\ C_{i'}\ at\ time\ (t)\} \times P\{U_{j_1}\ moves\ to\ C_i\ at\ time\ (t+\delta)\}$
P_{j_2} , U_{j_2} ∈m	$P\{U_{j_2}\ is\ in\ C_i\ at\ time\ (t)\} \times P\{U_{j_2}\ moves\ from\ C_i\ at\ time\ (t+\delta)\}$

Equation 1

$$P\{N_{C_i}(t+\delta) = M\} = \sum_m P\{N_{C_i}(t+\delta) = M \mid N_{C_i}(t) = m\} \times P\{N_{C_i}(t) = m\}$$
$$= \sum_m \langle \sum_{n_1,n_2,n_1-n_2=\Delta m} P\{N_{in,C_i}(t+\delta) = n_1\} \times P\{N_{out,C_i}(t+\delta) = n_n\}\rangle$$
$$\times P\{N_{C_i}(t) = m\}$$

In Equation 1, $P\{N_{in,Ci}(t+\delta)\}$ and $P\{N_{out,Ci}(t+\delta)\}$, describe the probability that n_1 users that may move into cell C_i and n_2 users that may move out from C_i at time (t+δ), respectively. These probabilities can be calculated using Equation 2.

Equation 2

$$P\{N_{in,C_i}(t+\delta) = n_1\} = \sum_{A_1 \in F_{C_{i'}(t)}} \prod_{j_1 \in A_1} P_{j_1} \prod_{j'_1 \in A_1^c} (1 - P_{j'_1})$$

$$P\{N_{out,C_i}(t+\delta) = n_2\} = \sum_{A_2 \in F_{C_i(t)}} \prod_{j_2 \in A_2} P_{j_2} \prod_{j'_2 \in A_2^c} (1 - P_{j'_2})$$

The group user prediction information could be further integrated with the network link bandwidth estimation and be utilized by EPCaaS to optimize the placement of components in the cloud or to dynamically adapt components while taking into account number of users and traffic load.

4.2 Testing Environment

MOBaaS has been evaluated through two different cloud infrastructures having the following organization.

4.2.1 Testbed of University of Bern (OpenStack IaaS)

This testbed is geographically hosted at the Institute of Computer Science and Applied Mathematics, University of Bern, Switzerland. It is running OpenStack Juno. We have committed the following resources to test MOBaaS: 100 instances, 100 VCPU, 50 GB RAM, 50 Public IPs and 1 TB of storage. The default MOBaaS instance has been configured to use 2 VCPU, 40 GB of storage, and 4 GB of RAM.

4.2.2 Testbed of ZHAW (OpenStack IaaS)

The testbed is geographically located at Zurich University of Applied Science in Winterthur Switzerland. It is running OpenStack Juno also, and resources available for MOBaaS are: 10 instances, 20 VCPU, 20 GB RAM, 8 Public IPs and 1 TB of storage. The default MOBaaS instance has been configured to use the same resources, i.e. 2 VCPU, 40 GB of storage, and 4 GB of RAM.

4.3 Evaluation Description

4.3.1 Different evaluation scenarios

Three types of measurements have been performed to evaluate the performance of MOBaaS with respect to service itself, MOBaaS integration with ICNaaS, and MOBaaS integration with EPCaaS.

4.3.1.1 MOBaaS service itself

This evaluation focuses on the service management performance including such parameters as MOBaaS instantiation and disposal delays.

4.3.1.2 Integration evaluation with ICNaaS

This evaluation focuses on the performance of MOBaaS when integrated with ICNaaS. Evaluation metrics include the prediction latency, i.e. an amount of time that MOBaaS requires to compute the prediction after a request is sent.

The integration between MOBaaS and ICNaaS is based on the request-answer approach. In this approach, a web server running at the MOBaaS side listens for requests from ICNaaS. Whenever ICNaaS needs a prediction, a request encapsulated in the JSON format is sent to MOBaaS. The request including the next time for prediction (e.g, δ=20), current time (e.g, 23:35) and date (e.g, Monday) and the IP address for receiving answers generated by MOBaaS. Figure 68 shows an example request message sent by ICNaaS.

```
'{"user_id":"0","future_time_delta":"20","current_time":"23:35","current_date":"Monday","reply_url":"http://130.92.70.142:8080"}'
```

Figure 68 - Example of request sent by ICNaaS to MOBaaS.

Given this information, MOBaaS makes the mobility prediction on the cell, where a given user will be located at the future time (δ) and returns the results in the JSON message. This prediction is repeated for every (δ) and the results are returned to ICNaaS immediately after finishing the prediction

calculation, Figure 69, shows a part of the output sent by MOBaaS triggered by the ICNaaS request shown in Figure 68.

{"multiple_user_predictions": [{"next_time": "23:55", "user_id": 5972, "current_cell_id": 670, "predictions": [{"cell_id": "670", "probability": "0.5000"}, {"cell_id": "3729", "probability": "0.5000"}]}, {"next_time": "23:55", "user_id": 5962, "current_cell_id": 5815, "predictions": [{"cell_id": "5815", "probability": "0.7524"}]}, {"next_time": "23:55", "user_id": 5993, "current_cell_id": 3121, "predictions": [{"cell_id": "2964", "probability": "0.1606"}, {"cell_id": "3121", "probability": "0.4733"}]}, {"next_time": "23:55", "user_id": 5977, "current_cell_id": 6013, "predictions": [{"cell_id": "6013", "probability": "1.0000"}]}, {"next_time": "23:55", "user_id": 5986, "current_cell_id": 1192, "predictions": [{"cell_id": "1192", "probability": "0.7497"}]}, {"next_time": "23:55", "user_id": 5978, "current_cell_id": 25580, "predictions": [{"cell_id": "25580", "probability": "1.0000"}]}, {"next_time": "23:55", "user_id": 5992, "current_cell_id": 5389, "predictions": [{"cell_id": "5389", "probability": "1.0000"}]}, {"next_time": "23:55", "user_id": 5963, "current_cell_id": 3729, "predictions": [{"cell_id": "3729", "probability": "0.6806"}, {"cell_id": "3738", "probability": "0.3194"}]}, ……..]

Figure 69 - Example of MOBaaS answer sent ICNaaS based on its request.

Figure 70, shows the request/answer messages exchanged between ICNaaS and MOBaaS. Upon getting request message by MOBaaS, mobility prediction on the cell, where a given user will be located at the future time (δ) is calculated and the results are returned to ICNaaS. Prediction procedure is repeated for every (δ) and the results are sent by answer messages after finishing the calculation.

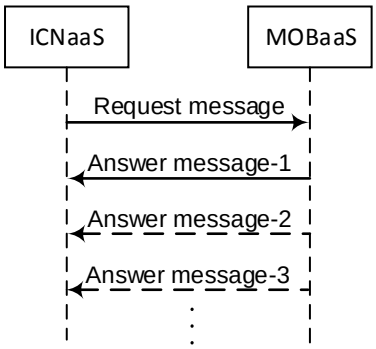

Figure 70 – Request/answer messages diagram between ICNaaS and MOBaaS

4.3.1.3 Integration evaluation with EPCaaS

This evaluation focuses on performance of MOBaaS when integrated with EPCaaS. We evaluated the time latency that MOBaaS is required to notify EPCaaS.

The integration between MOBaaS and EPCaaS is performed through MaaS. It is based on notification (trigger) approach. In this approach a web server running at the MOBaaS side can get prediction requests from EPCaaS, including two parameters: (i) maximum number of users (N) in each cell as a threshold (ii) the prediction interval (δ). In the current version of EPCaaS, there is no way to send N and δ in request messages and they are hard-encoded in MOBaaS. Upon receiving a request message (or starting manually in the current version), MOBaaS gets the current time and day from system and starts to predict the number of users in each cell after δ interval. If the number of users per cell exceeds the threshold, a trigger message is generated by MOBaaS and results are pushed into MaaS. Notification messages stored in MaaS contain the list of cells and the number of users in each after every (δ) interval. Figure 71, shows an example output of MOBaaS sent to MaaS. The threshold is set to 5 users, while each field in the notification message contains the time, cell ID, and number of user in the cell, respectively.

15:48 2677.0 7.0	15:48 562.0 10.0	15:48 5814.0 11.0	15:48 2964.0 27.0	
16:10 3018.0 6.0	16:10 2677.0 7.0	16:10 5814.0 11.0	16:10 562.0 12.0	6:10 2964.0 26.0
16:38 2677.0 7.0	16:38 2843.0 8.0	16:38 5814.0 10.0	16:38 562.0 10.0	16:38 2964.0 28.0
17:02 2843.0 7.0	17:02 562.0 7.0	17:02 2677.0 8.0	17:02 5814.0 10.0	17:02 2964.0 28.0

Figure 71 - Example of MOBaaS messages stored in zabbix server, in this example N=5.

Afterwards this information could be read by EPCaaS's SO for scaling upon decision making procedure. Figure 72, shows message sequence diagram exchanged between EPCaaS and MOBaaS.

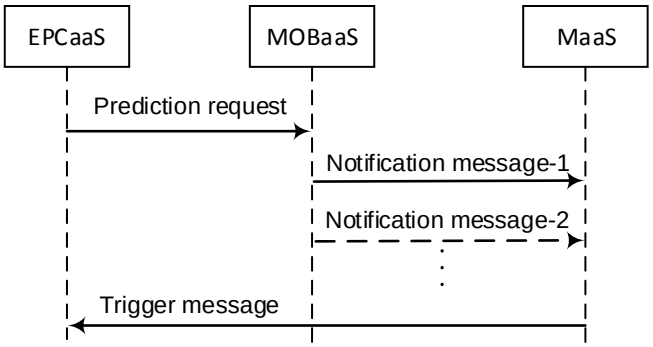

Figure 72 – Messages sequence diagram between EPCaaS and MOBaaS

In order to have a better view of prediction results, MOBaaS creates a graph for each run (prediction for each δ time), see Figure 73, providing the number of users in each cell for a specific date and time

Figure 73 - Number of users in each cell in a specific time and date

4.3.2 Mobility prediction

The spatial granularity of this algorithm is at the cell level. It means that for a user, the algorithm computes possible future cell(s) (locations) with respected probabilities. The temporal granularity of the algorithm depends on the application requirements and could be tuned at the minute scale. In order to evaluate the performance of the algorithm, we used the mobility data trace provided by Nokia for

academic research [9]. We selected mobility traces data for 100 users ranging over 2-6 months. For each user, we divided the available data into two parts, the learning data set (L), and the testing data set (T). L spans 70% of the user's data trace. It is used to derive the Marko Chain (M.C.) states and to calculate the transition probability matrix [1]. T contains the rest 30% of the data trace, which is used to test and evaluate the algorithm. For instance if the length of a data trace is 2.5 months (it includes trace data for 10 consecutive Mondays), we use the data trace of first seven Mondays the learning phase and the rest for testing.

The number of valid states (with a time step and cell ID) in the derived M.C. for each user depends on the quality of data trace for each day. In order to evaluate the accuracy of the algorithm, we randomly selected 10% of states for each user, out of M.C. states derived for each particular weekday from L. Afterwards, for each of the selected states, we predict future possible cell(s) in next (δ) minutes (e.g, δ =20 min). The states were chosen as random test points (timestamps, cell and ID) during the evaluation process. We check the probability of possible transitions for the same selected states in the data set T as well. Afterward, the Mean Absolute Error (MAE) for the possible transitions of the corresponding test points (chosen from the learning and testing data sets) is computed to get the prediction accuracy for each user in a particular weekday.

Equation 3

$$MAE = \frac{1}{M}\sum_{i=1}^{M}|P_{iL} - P_{iT}|\,,\ \ Accuracy = (1 - MAE) \times 100$$

In Equation 3, (M) represents the number of all possible transitions in the selected states, (PiL) and (PiT) define the calculated probability for each possible transition in the test points chosen from data set (L), and the checked probability of possible transitions for the same selected states in the data set (T), respectively.

4.3.3 Bandwidth prediction

The bandwidth prediction algorithm is based on the link dimensioning approach described in [10] and [2]. Its accuracy has been already proven in [10] and is therefore not assessed here in detail, but is, however, affected by multiple parameters. These parameters also affect the run time of the algorithm, which is evaluated within the context of MCN, and more specifically, within the aforementioned testbed at the University of Bern.

The input set contains an hour of network traffic (represented as flows) consisting of approximately 130.000 flow records. This network traffic has peaks of around 500Mbps. The link dimensioning algorithm has been subjected to 10 runs for each parameter set (described below), measuring the running time of the algorithm. If the standard deviation is too high after 10 runs, more runs are executed to provide additional data points.

The parameters for the algorithm are:

- ε: allowed error margin, ranging from 0 to 1, e.g. ε = 0.01 means a 1% error margin
- T: timescale on which the algorithm operates, in seconds

Smaller timescales result in more accurate predictions, at the cost of more computational resources spent on the algorithm execution. A lower error margin has similar effects.

The measurements are performed with the following parameters:

- ε ← {0.01, 0.1}
- T ← {1, 0.500, 0.250, 0.010}

This gives 2*4=8 sets of parameters. After all the runs for a parameter-set are performed, the mean and standard deviation of the running times are calculated, in milliseconds.

4.4 Evaluation Results

This section includes the performance evaluation of MOBaaS. The evaluation has two parts. The first part is about the service itself, which includes the evaluations of service instantiation/disposal time latency and service response time. The section part includes the evaluation of service accuracy, which means the accuracy evaluation of the prediction algorithms.

4.4.1 MOBaaS (performance evaluation)

This subsection shows the performance of MOBaaS service itself, such as time latency for service instantiation/disposal, time latency of single/multiple user prediction. Below, we show the results of different evaluation scenarios.

- **Time latency for service instantiation/disposal**

  Since MOBaaS does not depend on other MCN services, the instantiation time is short. To instantiate it, only certain functions of MOBaaS are needed, such as feeding the database with all the required data to issue a prediction. We therefore evaluated the MOBaaS instantiation time and the service disposal time on the UBERN testbed being resp. around 38 s and 10 s. In the tables below, we summarize the latency results received from UBERN and ZHAW testbed runs.

UBERN testbed	**Service Instantiation**	**Service Disposal**
Time Latency	31 seconds	11 seconds

ZHAW testbed	**Service Instantiation**	**Service Disposal**
Time Latency	42 seconds	18 seconds

- **Time latency for single user mobility prediction**

  We have performed various tests with different values of Delta (prediction interval), users, and days of a week (such as Monday, Tuesday,... Sunday). Here, we present the results with the delta value of 30 minutes.

User ID	**Current**	**Current**	**Current**	**Predicted**	**Execution Time**

	Day	Time	Cell	Cell/Probability	
5980	Friday	04:29	12963	3134, 0.33 12963, 0.67	9.53674316406e-07 seconds
6005	Wednesday	20:57	742	663, 1.0	1.19209289551e-06 seconds
5974	Tuesday	20:36	1535	1695,0.6 2050, 0.05 1535, 0.15	9.53674316406e-07 seconds
5963	Monday	22:51	3729	3738, 0.25 3729, 0.75	9.53674316406e-07 seconds

- **Time latency for multi-user mobility prediction**

  Due to the fact that the mobility prediction algorithm described in previous deliverables [D4.3 and D4.4] takes into account the day of a week (i.e. Monday, Tuesday, ..., Sunday), we have performed different tests to see the prediction latency with respect to different weekdays. The prediction results heavily depend on the size of the database, while the algorithm requires lots of information on the user mobility to provide an accurate prediction. In the following, we show the impact of the number of user (test carried out for 30 and 100 users) on the system performance.

30 Users	**Monday**	**Tuesday**	**Wednesday**	**Thursday**	**Friday**	**Saturday**	**Sunday**
Time Latency	68.2 second	83.5 second	81.32 second	84.4 second	93.7 second	92.6 second	80.6 second

100 Users	**Monday**	**Tuesday**	**Wednesday**	**Thursday**	**Friday**	**Saturday**	**Sunday**
Time Latency	287.2 second	273.5 second	285.2 second	298.4 second	363.7 second	341.75 second	276.6 second

As we can observe from the results, different weekdays have different latency results. This is due to the fact that the internal database has various resolutions of the location information for different users. Some users have more detailed location information recorded, which leads to longer prediction processing times. Other users have less-detailed information, which impacts the prediction accuracy, but results in a shorter prediction latency.

- **Time latency for group-user mobility prediction**

  The group-user mobility prediction will take longer, because the algorithm needs to monitor all the location information with respect to all the users concerned. This requires a significant amount of computing. We have performed 10 individually independent experiments. To save space, in the following table, we show the results of two tests. The table combines time slots

with certain cell IDs for which the number of users exceeds the threshold value of 6 (every cell id suspected to have more than 6 users attached). MOBaaS will send the predicted results to MaaS.

Time Slots	Cell ID	Predicted number of attached user
10:39	5815	6
10:39	3017	6
10:39	2677	6
10:39	2843	6
10:39	3112	6
10:39	562	6
10:39	5814	9
10:39	2964	25
Time latency for this prediction	1307.72740197 Second	

Time Slots	Cell ID	Predicted number of attached user
11:01	564	6
11:01	3018	6
11:01	2843	6
11:01	2963	7
11:01	562	8
11:01	2677	8
11:01	5814	9
11:01	2964	24
Time latency for this prediction	1255.18462992 Second	

4.4.2 Mobility prediction (accuracy evaluation)

As an example, Figure 74 shows the prediction accuracy calculated per weekday for a few users.

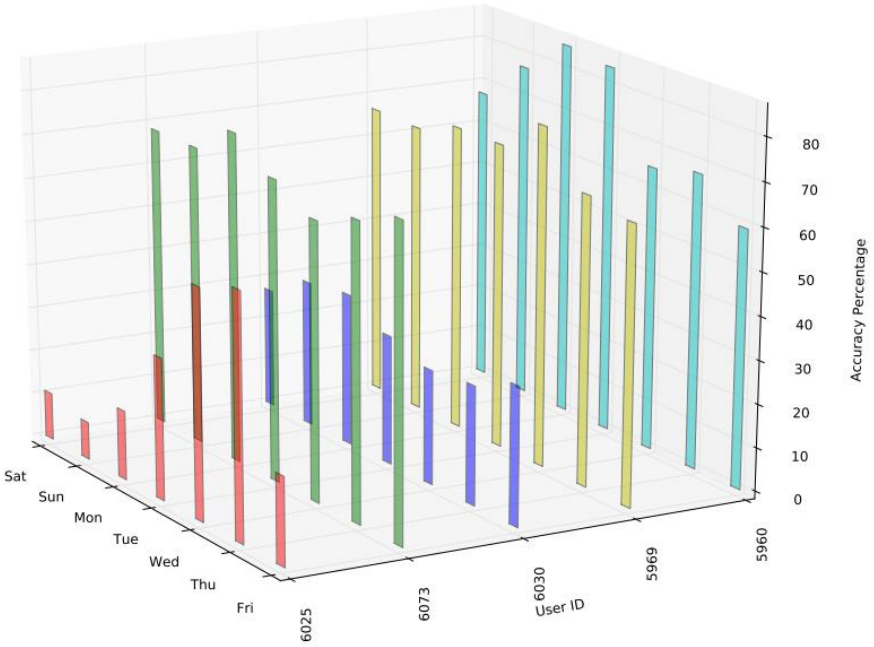

Figure 74 - Accuracy of algorithm for some users per day

Prediction accuracy is effectively influenced by quality of the data trace used to derive the transition probability matrix. As an example Figure 75 illustrates data trace for two users, leading to low (for user ID 6025) and high (for user ID 5960) prediction accuracies, see Figure 76.

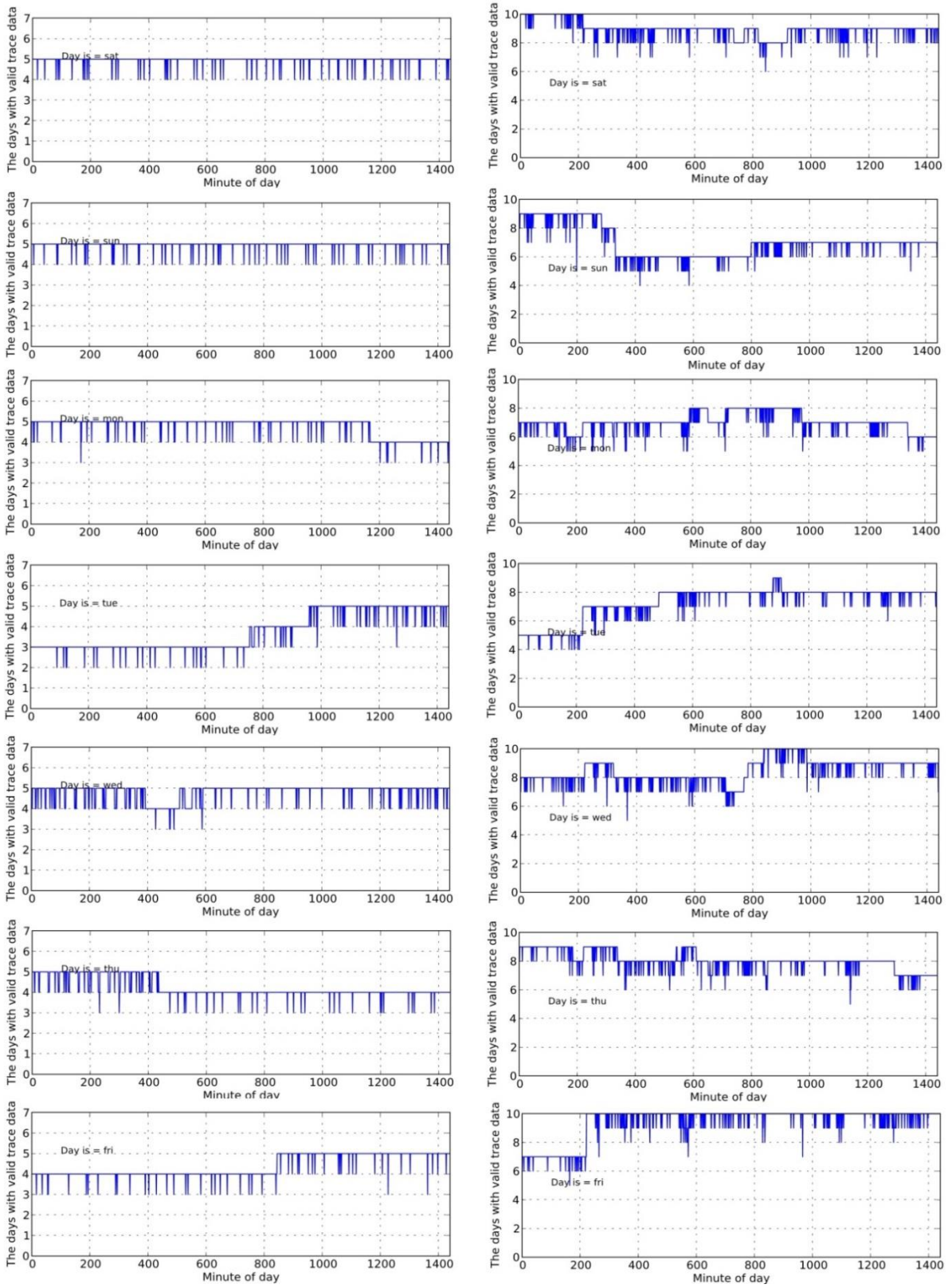

Figure 75 - Quality of data trace for two different users

Figure 76, displays overall accuracy for 100 users. For each user, it represents the average accuracy calculated for each single day of a week.

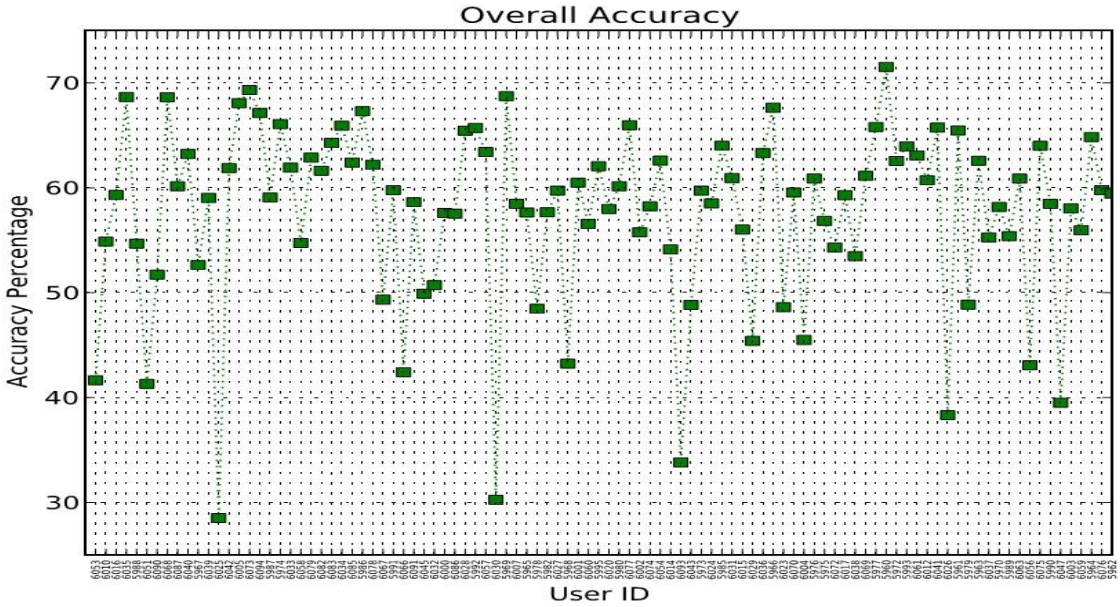

Figure 76 - The overall accuracy of prediction for 100 users

4.4.3 Bandwidth prediction (algorithm accuracy evaluation)

Table 21 shows the results of 8 individual measurements performed. For every set of parameters, the initial 10 iterations of the algorithm resulted in a standard deviation of <0.005 ms. Therefore, the yielded mean values in the table are deemed accurate.

Table 21 – Running times of bandwidth prediction algorithm

ε	0.1	0.1	0.1	0.1	0.01	0.01	0.01	0.01
T (s)	1.000	0.500	0.250	0.01	1	0.500	0.250	0.010
Iterations	10	10	10	10	10	10	10	10
Mean (ms)	0.01301	0.01117	0.01356	0.01294	0.01282	0.01327	0.01362	0.01360
Std (ms)	0.00357	0.00050	0.00327	0.00353	0.00228	0.00453	0.00431	0.00246

In all these cases, the execution time is at the sub-millisecond scale and therefore acceptable. The similarity in the mean points out that there is actually no real bottleneck. Even in the case of { ε = 0.01, T = 0.010 }, the execution time is fast, although the accuracy is not trustworthy, as previously described in this document and depicted in Figure 78. In reality, the time needed to propagate the requests and results from and to MOBaaS will be higher than the time required by the algorithm. This illustrates that the algorithm is fast enough to be used within the MCN scenario.

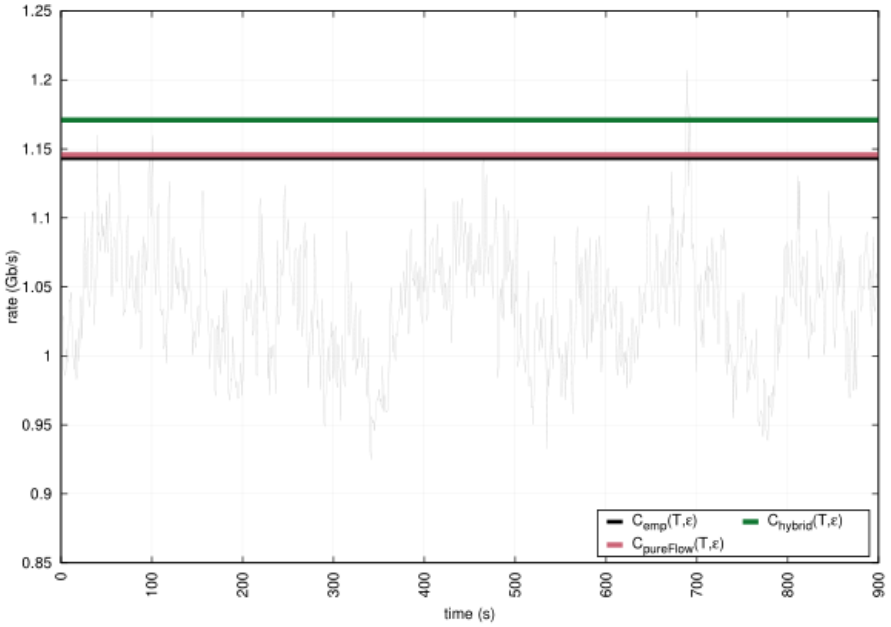

Figure 77 - Accuracy using T=1 (based on [10])

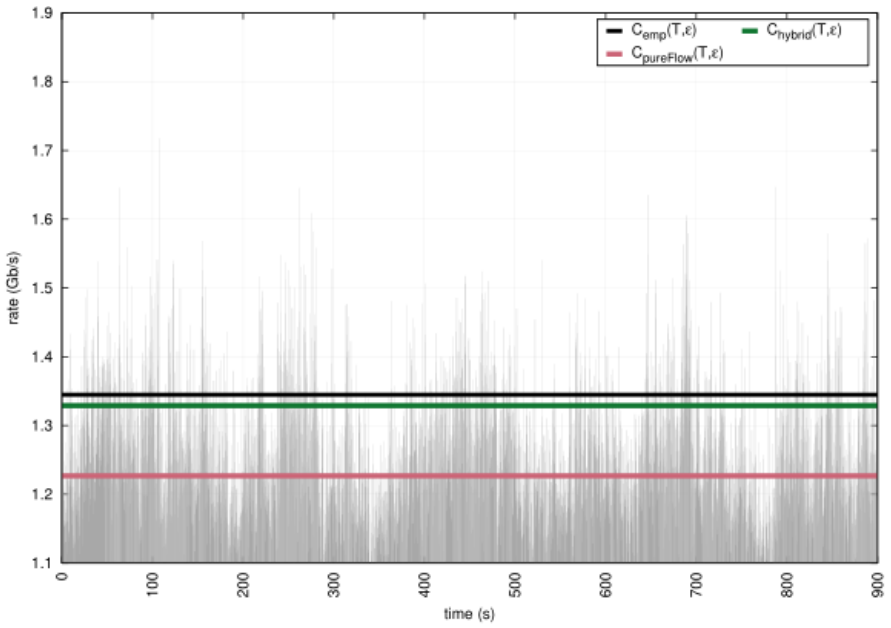

Figure 78 - Accuracy using T=0.010 (based on [10])

5 Run time fine-grained resource aware network service management

The MCN system architecture requires a Service Orchestrator (SO) in order to control the lifecycle management (LCM) decisions for the various services. Based on some policy criteria, the SO will trigger the necessary management actions. Usually the policy criteria can be based on a single observable metric. For example, in the case of the SO for EPCaaS, the decision to scale in/out is based on the number of users attaching to the system. However, in this section we provide the details of a novel fine-granular resource-aware network service management (NSM) method [11] that enables the SO to make optimum runtime network service management (NSM) or LCM decisions (such as migration, scaling, cloning etc.) by taking a correlated view of the utilization of the underlying resources by the SICs. A simple approach of making LCM decisions is based on monitoring utilization of a specific Resource Unit (RU) (such as CPU, I/O, memory etc.), and then triggering a pre-set management action whenever a specific threshold is crossed. In contrast, the adopted method takes an inclusive view of the utilization of one or more RU with reference to a reference resource unit (RRU). The motivation is to enhance in general the capabilities of the MCN SO in making optimum resource-aware management decisions. The conceptual overview of the aforementioned method is given below.

5.1 Method overview

The concept of "affinity" is central to this method, whereby the term "affinity" refers to the correlation between different RUs. The affinity value, or "affinity score", is a vector quantity that indicates the level or degree of dependence of one or more RUs on a RRU. This method derives and communicates information depicting the correlation, or affinity, between different RUs with reference to a specified key RU under different workload conditions. This derived vector is referred to as Reference Resource Affinity Score (RRAS).

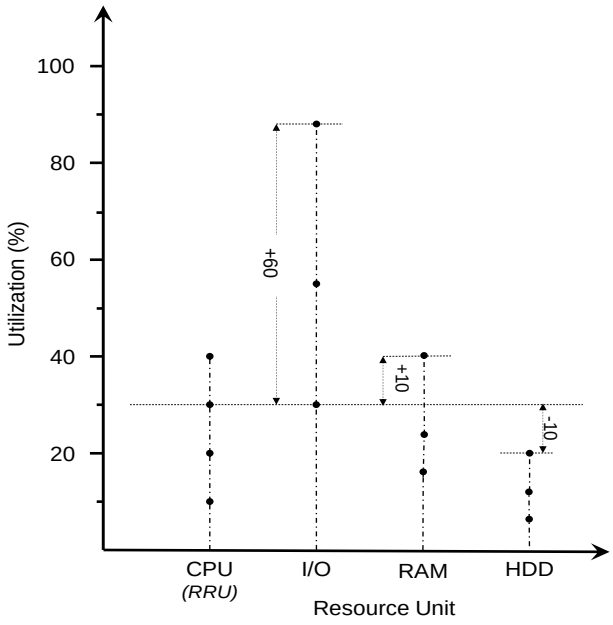

Figure 79 - Resource Utilization for a specific SIC with RRAS with reference to CPU.

RRAS provides an insight on how and by how much the utilization of one reference RU will impact the utilization of other RUs. RRAS thus expresses the correlation, or the level of dependence, of an

individual RU on the reference RU in terms of utilization. For example, the RRAS value of an I/O resource with reference to CPU indicates the degree of its utilization dependence on the CPU utilization. A high RRAS value would indicate a strong affinity, whereas a small value will indicate weaker affinity or dependence. The RRAS value computation, which can be carried out within the respective server or by the Cloud Controller (CC), is done for all SICs in a server and are presented as RRAS reports. A RRAS report, that is stored in a repository within the CC, presents a fine-granular information that can be incorporated by an internal resource management entity (i.e., a hypervisor) or by an external resource management entity, such as the CC and/or the SO, to make informed decisions in terms of optimum resource management during run-time under different workload conditions at the server level or infrastructure level respectively.

As an example, Figure 79 shows the average percent utilization of each respective RU recorded over a specific time period $T = t_n - t_0$, where t_0 and t_n denote the start and end times of a monitoring epoch for a SIC, respectively. The values are monitored and recorded for a pre-defined utilization interval of the RRU. In this case, the utilization of the RUs are monitored for every 5% change in CPU utilization, and the values are averaged over the duration T. Based on these individual scalar values, the CC-agent within the server (as part of hypervisor) derives the RRAS report for each VM hosting the SIC, an example of which is shown in Table 22. Please note that an implementation choice may also allow for the sending of the utilization values to the CC where the RRAS report can also be derived. Table 22 provides the RRAS report corresponding to the example utilization values of a specific SIC shown in Figure 79. Based on the absolute average utilization values, the CC computes the RRAS for each RU with reference to each other RU by taking the difference between the average utilization values of the RUs with the reference RU, and considers this difference as the RRAS value (see Table 22). In Table 22, the RU of each row is a reference with respect to which the RRAS for other RUs are computed.

Table 22 - RRAS report snapshot for a specific SIC.

Reference Resource Unit	***Absolute Average Utilization (%)***	***Reference Resource Affinity Score***			
		CPU	*I/O*	*RAM*	*HDD*
CPU	30	-	+60	+10	-10
I/O	90	-60	-	-50	-70
RAM	40	-10	+50	-	-20
HDD	20	+10	+70	+20	-

Being vector quantities, the RRAS values show how much the utilization of an RRU is impacting the other RUs. A positive RRAS value of a particular RU indicates that the utilization of the RRU will result in an increased utilization of the specific RU well above the level of utilization of the reference RU. On the other hand, a negative value indicates that the utilization of the reference RU will result in a decreased utilization of the particular RU well below the level of utilization of the reference RU. This will help enable CC to precisely determine the influence of a RRU on the other RUs. For instance, with CPU as a RRU, it is observed from Figure 79 that 30% utilization of CPU will correspond to 60% of the utilization of the I/O RU, 10% utilization of the memory RU and -10% utilization of the storage RU. The RRAS values are derived by taking the absolute difference between the percentage utilization of the respective RUs and the RRU, as shown in Figure 80. For example, the first row of Table 22 indicates the RRAS of different RUs with reference to the CPU as a RRU. Thus, from the table above, it can be interpreted that the I/O module has the "strongest" affinity with the CPU with a RRAS value of +60, while the storage has the least affinity, which is -10. In other words, the I/O module will experience a higher degree of utilization than the storage with respect to CPU utilization. This could be indicative of a SIC that may perform packet forwarding and routing.

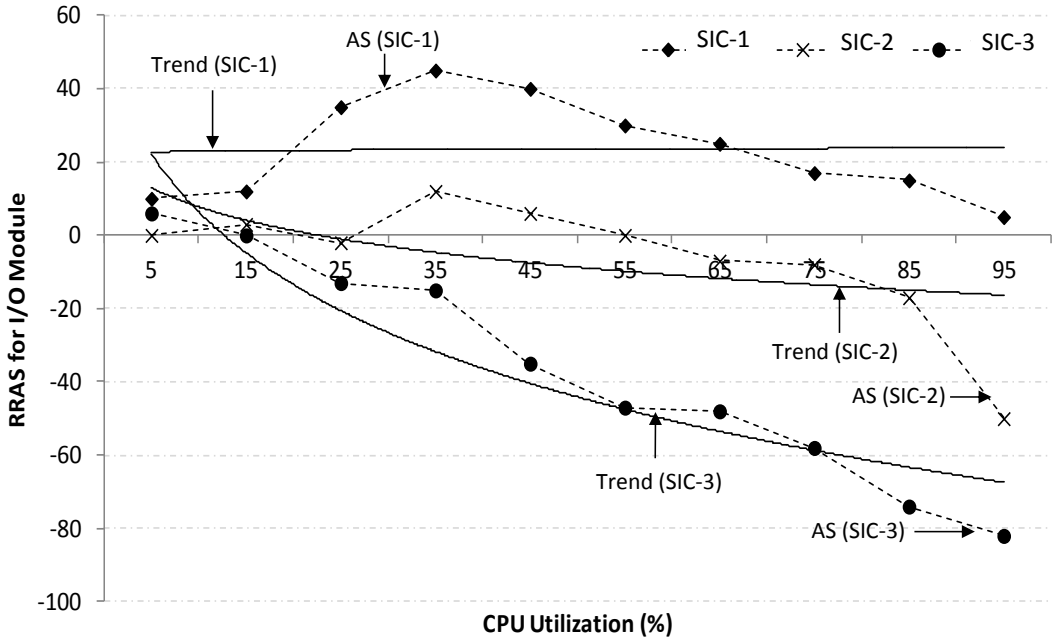

Figure 80 - Affinity Signature for I/O module RU with reference to CPU RU based on RRAS reports

The CC maintains the past RRAS reports for a specific RU or a set of RUs. The period of history can range from minutes to hours or even days, depending on the policy. Such historical/past record of the RRAS report enables the CC to derive "Affinity Signature (AS)" of RU(s) with respect to a RRU. AS provides the CC the information about the long term affinity of a RU with a RRU for a SIC. Time series models can be used to improve the accuracy of AS. An AS is a plot of the successive RRAS values against the specific utilization values of a RRU. The information provided by the RRAS reports and AS can be manipulated by deriving statistics such as affinity trend as seen in Figure 80. The figure shows the AS of the I/O modules for three SICs with reference to the processing RU (i.e., CPU). The dotted lines indicate AS of the respective SICs while the solid lines indicate the logarithmic trend of the affinity of I/O with CPU. As evident from the figure, the I/O module of SIC-1 shows a consistently strong long-term affinity with CPU. This is also indicated by the almost constant trend on the positive axis. This shows that the utilization of the I/O module is highly dependent on the CPU utilization; as high CPU utilization will incur high utilization of the I/O module. On the other hand, the long-term affinity of SIC-2 and SIC-3 is not very strong, as indicated by the decaying trend of the AS in the negative axis. This shows that the utilization of the I/O resources is not significantly impacted with CPU utilization. This could imply SIC-2 and SIC-3 as non-I/O intensive while SIC-1 being I/O intensive.

The AS of different RUs can be combined to determine the short-term and long-term demands and behaviour of a SIC. The information thus derived from the AS can be utilized by the Service Orchestrator Decision (SOD) for making short-term and long-term informed and optimized decisions in performing network service management operations such as, but not limited to deployment/instantiation, migration, cloning, scaling (in/out; up/down), and run-time dynamic resource provisioning.

To further refine the RRAS reports and the AS graph, the time of day (ToD) can be also considered to indicate the times at which the affinity is the strongest or weakest. This can help the system in optimum management of resources at different ToDs. As a use-case, consider a routing VNF the AS of which indicates a strong affinity of the I/O RU and weak affinity of the memory RU during specific ToDs when the routing VNF is involved in routing user plane (U-plane) data traffic. Considering the fact that for M2M applications the control-plane (C-plane) traffic exceeds the user-plane (U-plane) traffic, thus

imparting a higher demand on memory RU (for maintaining states) than on I/O RU, the system may then decide to reroute the M2M traffic over this routing VNF to maximize the utilization of available resources. Additionally, the AS information can also be used for power savings, i.e., VNFs indicating weaker affinity to allotted resources can be consolidated on fewer servers, thereby preserving resources and energy.

As another use-case, the RRAS reports and the AS graph can be used by VMM in making local management decisions, such as the (re)allocation of underutilized resources (i.e., resource black-holes) from one VM to another within the same PM. It should be noted that a demo prototype based on this method is being developed the details of which will be presented in D6.4.

6 Conclusions and Further Work

This deliverable has presented the basic evaluation of the core network services including the orchestration, the virtualized EPC, the virtualized ANDSF and the mobility and bandwidth prediction. For each different fundamental experiments were described, including the experimentation itself, the achieved measurements and their assessment.

From the perspective of the specific services, it was observed that there are no abnormal delays or performance requirements, deriving a rather high level of trust in the developed technology.

The runtime procedures as seen by the EPCaaS orchestrator was assessed. Especially the scaling in and the scaling out of the virtualised Switch component enabling to determine when is the most opportune to start dynamically such a component within an OpenStack environment.

An initial set of measurements of the virtualized EPC functionality was provided concentrating mainly on the control plane operations. The measurements establish the delay of the specific operations, giving a rough estimate of the compute dimensioning required for the specific components as well as the critical functionality. In the assessed N:2 architecture, only the delay of the CTRL matters, as both the switch and the HSS have almost instant response (having in the given situations only a request-response role). For the data path measurements, the introduced delay by the virtualization was inconclusive mainly because the used benchmarking tool was also virtualized, thus not accounting for the delay of transferring the data traffic to a virtualized environment.

A comprehensive set of measurements was executed for the ANDSFaaS service, in order to validate and evaluate the behaviour of a 1:N implementation architecture. The testing plan included scenarios considering, the deployment, scaling and reliability aspects; providing measurements about the quality of service delivery, namely regarding availability, balancing and impact on physical resources.

Regarding ANDSFaaS deployment, different options were tested with deployment times ranging between 5 and 10 minutes, resulting in low response times for ANDSFaaS on-demand requests from customers. It was observed that, as expected, as the number of nodes increases the deployment time also increases, in a linear way.

Regarding ANDSFaaS scaling, all tiers are able to scale triggered by monitoring data, using any metrics, in particular, the CPU utilization or the number of requests per second. It was observed that scaling works properly on the 3 tiers, with minor downtimes associated. Scaling out actions take around 2 minutes to complete, while scaling in take a few tens of seconds. This permits acceptable response times for service adaptation to fast load service variation.

Regarding ANDSFaaS reliability, it was observed that the ANDSFaaS service is able to automatically recover from component failures within acceptable times. Nevertheless, for the case of DB, and for integrity reasons, it takes significantly more time to recover than for the others.

Although the algorithms for bandwidth and mobility prediction are simple, through the proposed results its opportunity to be introduced in the current and future virtualized networks is high as they provide an insight into the future needs of the specific subscriber. Based only on the derivation of the requirements form the subscriber mobility, the aggregated requirements are further assessed enabling to determine the capacity needs for the specific parts of the end-to-end system.

Additionally, the resource aware network service provides an initial view on the functional placement within a data center of the different network components.

The further steps, which are part of the experimentation and evaluation task, are related to the evaluation of all the other implemented features within the EPCaaS system, including the scaling-in and scaling-out of the control plane components, the evaluation and the optimization of the data path components as well as the comparison between physical and virtualized systems.

A very careful look will be given to the reliability of the end-to-end system especially to assess the state sharing mechanism with a more advanced solution compared to the ceph based one proposed in D4.3 and D4.4.

A next step is the evaluation within the end-to-end prototype of the introduced delay by the specific EPC functionality as well as by its integration with third party functionality e.g. the RAN, the IMS and the DSS service.

References

[1] Mobile Cloud Networking, "D4.4".

[2] Mobile Cloud Networking, "D4.3".

[3] Mobile Cloud Networking, "D2.5".

[4] "EPCaaS git repository," [Online]. Available: https://git.mobile-cloud-networking.eu/m18-integrated-demo/d-epc-sm.git.

[5] TE-524262, White Paper, *NEC Virtualized Evolved Packet Core - vEPC,* http://www.nec.com/en/global/solutions/nsp/nfv/doc/vEPC_WP.pdf, 2014.

[6] Z. Yousaf, P. Loureiro, F. Zdarsky, T. Taleb and M. Liebsch, "Cost Analysis of Initial Deployment Strategies for a Virtualized Mobile Core Network Functions," *IEEE Communication Magazine,* vol. (submitted), 2015.

[7] MongoDB, "Sharding manual," [Online]. Available: http://docs.mongodb.org/manual/sharding/.

[8] Apache, "Apache JMeter," [Online]. Available: http://jmeter.apache.org/.

[9] Idiap, "Mobile Data Challenge (MDC) Dataset," December 2014. [Online]. Available: https://www.idiap.ch/dataset/mdc/.

[10] R. Schmidt, Measurement-based Link-Dimensioning for the Future Internet, 2014.

[11] Y. Zarrar and T. Taleb, "Fine Granular Resource-Aware Virtual Network Function Management for 5G Carrier Cloud," *IEEE Network Magazine,* 2015.

[12] IEEE, "IEEE Guide for Developing System Requirements Specifications," 1998.

[13] R. Yrjo and D. Rushil, "Cloud Computing in Mobile Networks – Case MVNO," Berlin, 2011.

[14] R. E. Freeman, Strategic Management: A Stakeholder Approach, Pitman, Boston, Toronto, 1984.

[15] R. Copeland, "Modelling multi-MNO business for MVNOs in their evolution to LTE, VoLTE & advanced policy," Berlin, 2011.

[16] R. Buyyaa, C. S. Yeoa, S. Venugopala, J. Broberga and I. Brandic, "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility," 2009.

[17] R. K. Mitchell, B. R. Agle and D. J. Wood , "Toward a Theory of Stakeholder Identification and Salience: Defining the Principle of Who and What Really Counts," *The Academy of Management Review,* vol. 2, no. 4, 1997.

[18] S. Robertson and J. Robertson, Mastering the Requirements Process: Getting Requirements Right, 3rd edition ed., Addison-Wesley Longman, 2012.

[19] S. Robertson and J. Robertson, Mastering the Requirements Process: Getting Requirements Right, 2nd ed., Addison Wesley Longman, 1996.

[20] Wireless Inteligence, "Analysis: The MVNO model, global footprint and outlook," [Online]. Available: https://wirelessintelligence.com/analysis/2012/05/the-mvno-model-global-footprint-and-outlook/. [Accessed 11 03 2013].

[21] B. Ayvazian, "Market Opportunities for B2C Fourth-Generation MVNOs," Heavy Reading, 2012.

[22] Mobile Cloud Networking, "D4.2".

[23] OpenStack, "Software Configuration," 23 06 2015. [Online]. Available: http://docs.openstack.org/user-guide/hot-guide/hot_software_deployment.html.

[24] Mobile Cloud Networking, "D4.1".

A Third Party and Open Source Software used

The ANDSF software is to be used with the following 3rd party components

1. nginx – HTTP load balancer
   - http://nginx.org
   - http://nginx.org/LICENSE
2. tomcat - Webserver
   - http://tomcat.apache.org
   - http://www.apache.org/licenses/LICENSE-2.0
3. MongoDB – Object-oriented Database
   - http://www.mongodb.org
   - Lhttp://www.gnu.org/licenses/agpl-3.0.html

The OpenEPC software is built with the following 3rd party libraries:

1. RSA Data Security, Inc. MD5 Message-Digest Algorithm
2. libdrizzle - Drizzle Client & Protocol Library
   - https://launchpad.net/libdrizzle
   - http://www.opensource.org/licenses/bsd-license.php
3. libgcrypt - General purpose cryptographic library
   - http://directory.fsf.org/project/libgcrypt/
   - http://directory.fsf.org/license/LGPL/
4. libxml2 - The XML C parser and toolkit of Gnome
   - http://xmlsoft.org/
   - http://www.opensource.org/licenses/mit-license.html MIT License
5. Sofia-SIP - Open Source SIP User-Agent library
   - http://sofia-sip.sourceforge.net/
   - http://www.gnu.org/copyleft/lesser.html
6. GStreamer - Library for constructing graphs of media-handling components
   - http://gstreamer.freedesktop.org/
   - http://opensource.org/licenses/lgpl-2.1.php
7. UDNS - DNS Resolver Library
   - http://www.corpit.ru/mjt/udns.html
   - http://www.gnu.org/licenses/licenses.html#LGPL
8. OpenSSL - library used for cryptographic algorithms
   - http://www.openssl.org/
   - http://www.openssl.org/source/license.html

9. libmicrohttpd - a small C-library for running a HTTP server
   - http://www.gnu.org/software/libmicrohttpd/
   - http://www.gnu.org/licenses/lgpl.html
10. libcurl - free and easy-to-use client-side URL transfer library
    - http://curl.haxx.se/libcurl/
    - http://curl.haxx.se/docs/copyright.html
11. libwbxml - parse, encode and handle WBXML documents
    - https://libwbxml.opensync.org/
    - http://www.gnu.org/licenses/lgpl.html
12. asn1c - The ASN.1 to C compiler (with FOKUS modifications)
    - http://lionet.info/asn1c/blog/
    - http://sourceforge.net/directory/license:bsd/
13. libsctp - Linux Kernel Stream Control Transmission Protocol Tools
    - http://lksctp.sourceforge.net/
    - http://www.gnu.org/licenses/lgpl-2.1.html
14. libgps - C service library for communicating with the GPS daemon
    - http://www.catb.org/gpsd/
    - http://git.savannah.gnu.org/cgit/gpsd.git/tree/COPYING
15. libpcsc-lite - Middleware to access a smart card using SCard API (PC/SC).
    - http://pcsclite.alioth.debian.org/
    - http://pcsclite.alioth.debian.org/pcsclite.html

The OpenEPC software is to be used with the following 3rd party components, which are not covered by this license:

1. MySQL - Open Source database software
   - http://www.mysql.com
   - http://www.gnu.org/licenses/old-licenses/gpl-2.0.html
2. ISC-DHCP - Open Source DHCP implementation
   - http://www.isc.org/software/dhcp
   - http://www.opensource.org/licenses/isc-license.txt
3. Apache2 - Open Source HTTP Server
   - http://httpd.apache.org/
   - http://www.apache.org/licenses/
4. PHP5 - General-purpose scripting language suited for Web development
   - http://php.net/
   - http://www.php.net/license/
5. Squid - Cache proxy for Web supporting HTTP, HTTPS, FTP and more
   - http://www.squid-cache.org/
   - http://opensource.org/licenses/gpl-2.0.php
6. Tcpdump - a powerful command-line packet analyzer; and libpcap, a portable C/C++ library for network traffic capture
   - http://www.tcpdump.org/

- http://www.tcpdump.org/license.html

7. Wvdial - an easier way to connect to the internet
   - http://freecode.com/projects/wvdial
   - http://freecode.com/tags/gnu-lesser-general-public-license-lgpl

OpenEPC implements and uses proprietary algorithms for which a separate license has been specifically acquired by Fraunhofer FOKUS. The following items fall under the terms of their specific licensors and might need an additional license based on particular use purposes or code re-usability.

1. 3GPP Confidentiality and Integrity Algorithms UEA2 and UIA2 (a.k.a. SNOW3G)
   - http://www.gsma.com/technicalprojects/fraud-security/security-algorithms

     "The GSMA , having cooperated in the development of the 3GPP Confidentiality and Integrity Algorithms UEA2 and UIA2 ("The UEA2 & UIA2 Algorithm"), has been granted distribution rights to the Algorithms that have been developed through the collaborative efforts of the 3GPP Organisational Partners.

     The UEA2 and UIA2 Algorithm specifications are available below and may be used only for the development and operation of equipment conforming to the UEA2 & UIA2 Algorithm or standards based on it. Every Beneficiary intending to implement and/or use the UEA2 & UIA2 Algorithm must sign a Restricted Usage Undertaking with a Custodian and demonstrate that they satisfy the approval criteria specified in the Restricted Usage Undertaking."

The EPC_Config software (EPC Side) is built with the following 3rd party libraries:

1. libdrizzle - Drizzle Client & Protocol Library
   - https://launchpad.net/libdrizzle
   - http://www.opensource.org/licenses/bsd-license.php
2. libxml2 - The XML C parser and toolkit of Gnome
   - http://xmlsoft.org/
   - http://www.opensource.org/licenses/mit-license.html MIT License
3. UDNS - DNS Resolver Library
   - http://www.corpit.ru/mjt/udns.html
   - http://www.gnu.org/licenses/licenses.html#LGPL
4. OpenSSL - library used for cryptographic algorithms
   - http://www.openssl.org/
   - http://www.openssl.org/source/license.html
5. libmicrohttpd - a small C-library for running a HTTP server
   - http://www.gnu.org/software/libmicrohttpd/
   - http://www.gnu.org/licenses/lgpl.html
6. libcurl - free and easy-to-use client-side URL transfer library
   - http://curl.haxx.se/libcurl/
   - http://curl.haxx.se/docs/copyright.html
7. libwbxml - parse, encode and handle WBXML documents
   - https://libwbxml.opensync.org/
   - http://www.gnu.org/licenses/lgpl.html

The EPC_Config software (EPC Side) is to be used with the following 3rd party components, which are not covered by this license:

1. OpenStack
   - http://www.openstack.org/
   - http://www.apache.org/licenses/LICENSE-2.0.txt
2. zabbix:
   - http://www.zabbix.com/
   - http://www.gnu.org/licenses/old-licenses/gpl-2.0.html
3. bottlepy:
   - http://bottlepy.org/docs/dev/
   - MIT License
4. moniker, dnsaas:
   - https://github.com/stackforge/designate
   - https://github.com/stackforge/designate/blob/master/LICENSE

B ANDSFaaS Scaling (Additional Results)

This Appendix includes some scaling scenarios that were tested and not presented in the document body. This includes the 3 tier scaling: WS, IRP and DB; with the following parameterizations:

- The traffic generation with Profile 2 (less aggressive slope);
- Number of components incremented/decrements per scaling event: +-2 VMs;
- The metric used as trigger is the CPU utilization (CPU);

6.1 WS Scaling

6.1.1 Comparison with Profile 2

This section intends to compare the results of the Base Configuration (section 3.5.2.2.1.1), when the traffic profile changes to a less aggressive increase (and decrease) in the number of requests.

This section considers scaling tests with the following parameterization..

- Scaling trigger: number of requests (Req);
- Scaling increments: +-1 VM;
- Traffic generation: Profile 2 (low increasing and decreasing rate).

6.1.1.1 Scale out

The Figure 81 depicts the time elapsed to scale out an ANDSFaaS instance on the WS tier, showing the WS2 and WS3 instances (WS1 was already instantiated) and the 3 instantiation steps (OS, VM Ready and Service Ready).

As a result, it can be seen that the scaling out decision is taken later, considering that the threshold on the number of requests takes more time to be reached when compared with the section 3.5.2.2.1.1. For example, here the first scaling out occurs around the 170s, while with Profile 1 it is around the 120s. A similar effect occurs for the second scaling out operation.

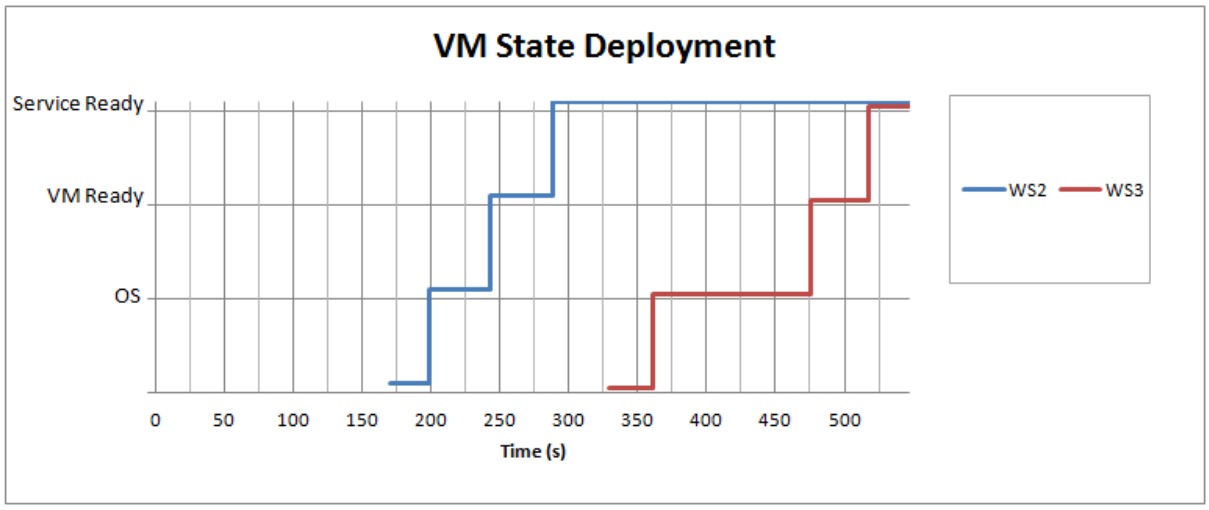

Figure 81 – ANDSFaaS: Scale WS, Trigger Req, Increment 1, Profile 2 (scale out)

6.1.1.2 Scale in

The Figure 82 depicts the time elapsed to scale in an ANDSFaaS instance on the WS tier, showing the WS3 and WS2 instances (WS1 was already instantiated).

As a result, it can be seen that the scaling in decision is taken later, considering that the threshold on the number of requests takes more time to be reached when compared with the section 3.5.2.2.1.1. For example, here the first scaling in occurs around 930s, while with Profile 1 it is around 630s. The same occurs for the second scaling in operation.

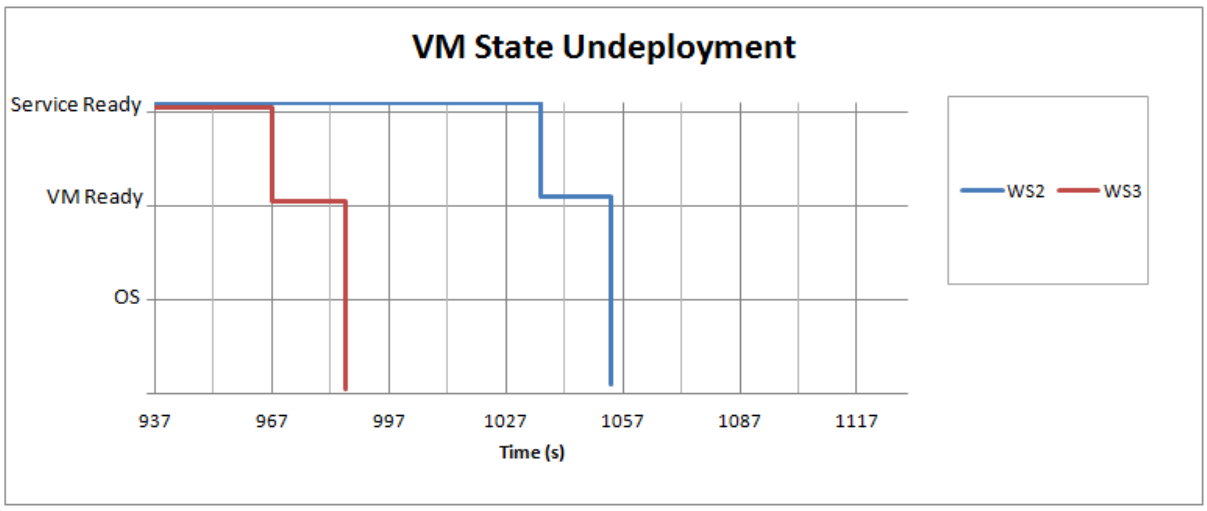

Figure 82 – ANDSFaaS: Scale WS, Trigger Req, Increment 1, Profile 2 (scale in)

6.1.1.3 Scale out and in

The Figure 83 shows the combination of scaling out and in an ANDSFaaS instance on the WS tier, as already shown above.

Compared to the usage of Profile 1 (high increasing rate), and as expected, the scaling operations are triggered later.

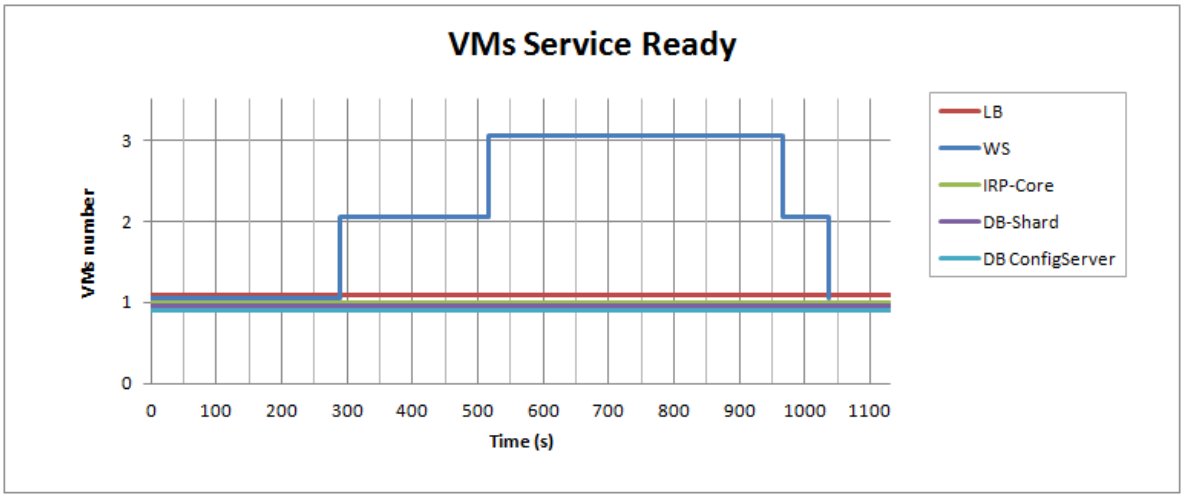

Figure 83 – ANDSFaaS: Scale WS, Trigger Req, Increment 1, Profile 2 (scale out and in)

6.1.1.4 Number of Requests

The Figure 84 depicts the service behaviour during ANDSF scaling in and out operations on the WS tier, showing the total number of requests and per WS VM.

As a result, it can be seen that service loss is similar as for the case of Profile 1.

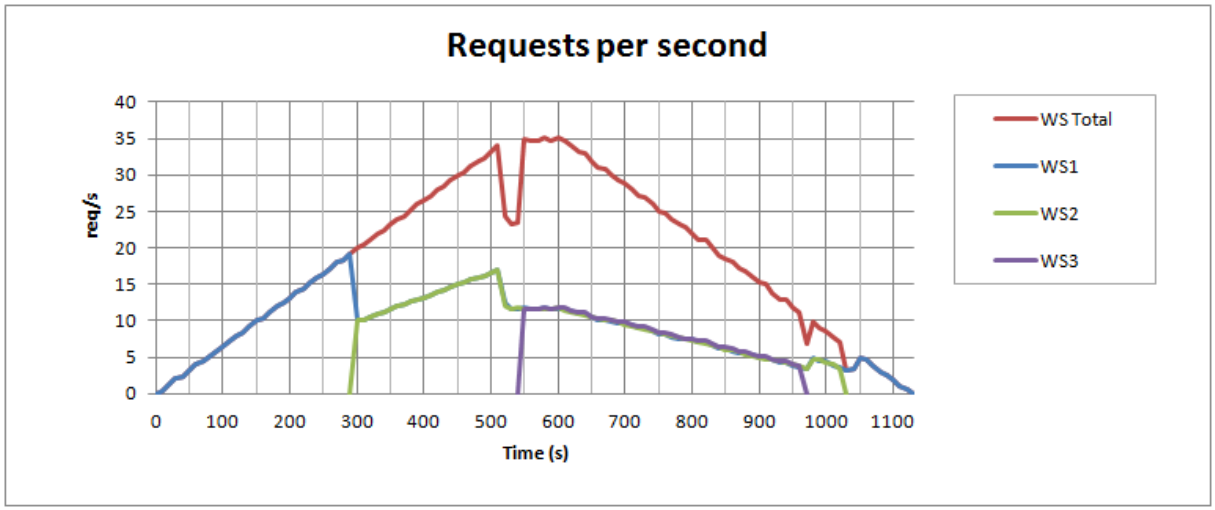

Figure 84 – ANDSFaaS: Scale WS, Trigger Req, Increment 1, Profile 2 (number of requests)

6.1.1.5 Impacts on Physical Machine

The Figures below show the impact of the ANDSFaaS scaling operations on the physical machine resources. In particular, the Figure 85, Figure 86 and Figure 87 show the impacts on CPU, RAM and I/O utilization, respectively, along the deployment and release periods.

As a result, it can be seen that the most relevant impact on the physical machine relates to I/O utilization, although quite less significant than for the deployment operations. The CPU and RAM are slightly affected. It can also be seen that scale-in operations require less resources than scale out.

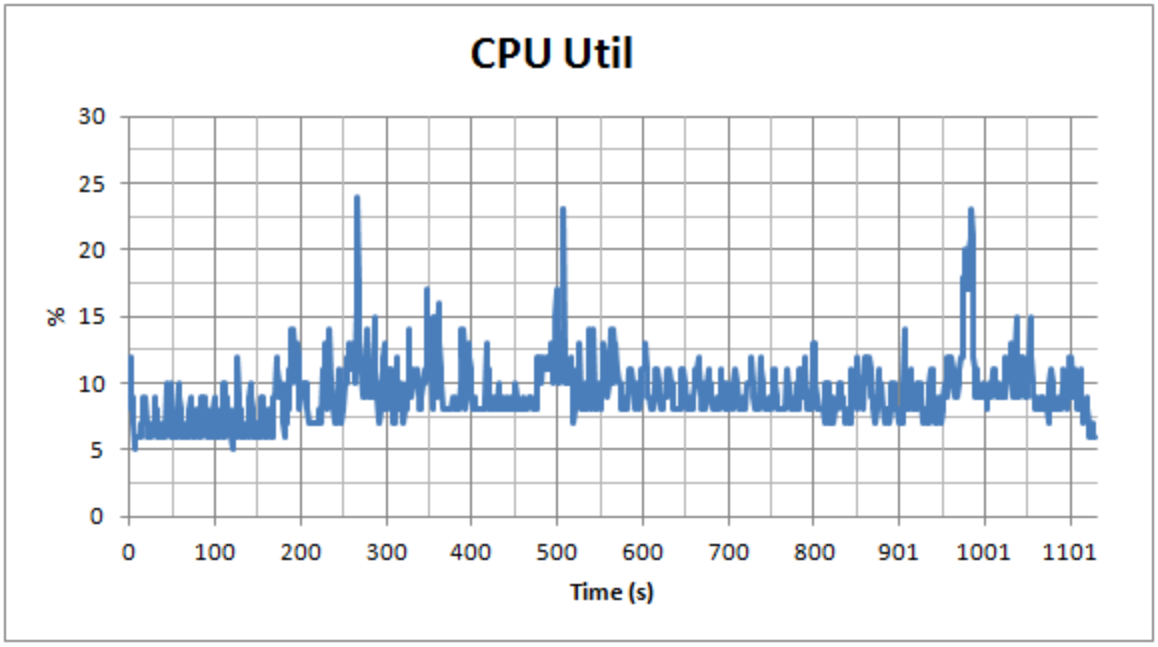

Figure 85 – ANDSFaaS: Scale WS, Trigger Req, Increment 1, Profile 2 (PM CPU utilization)

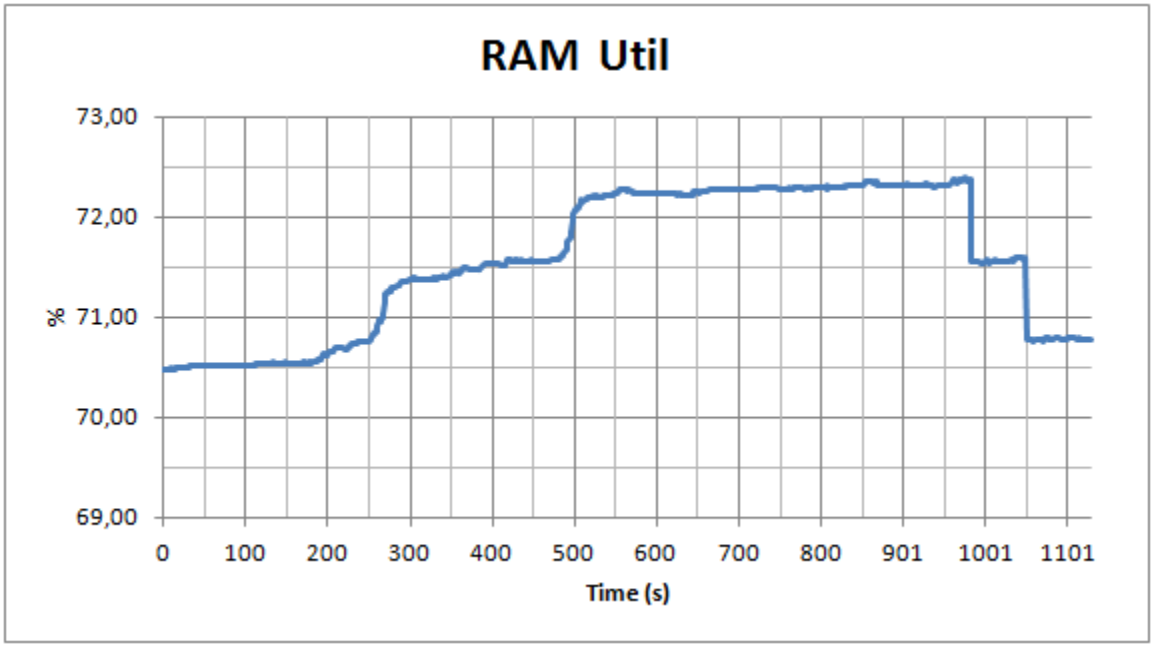

Figure 86 – ANDSFaaS: Scale WS, Trigger Req, Increment 1, Profile 2 (PM RAM utilization)

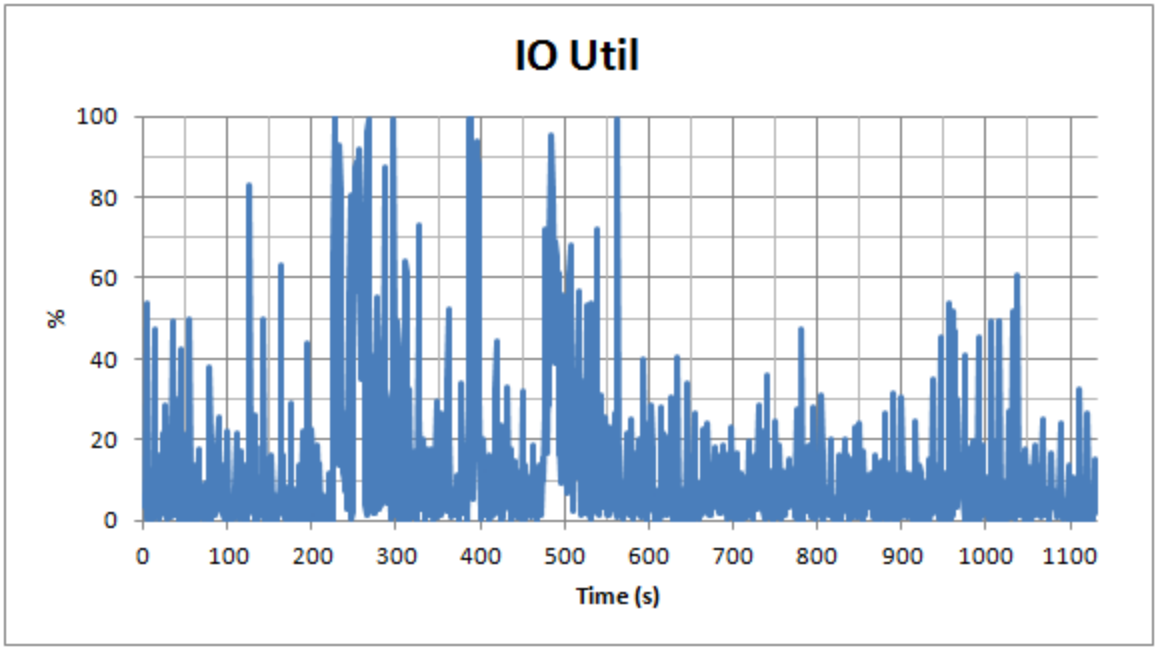

Figure 87 – ANDSFaaS: Scale WS, Trigger Req, Increment 1, Profile 2 (PM I/O utilization)

6.1.2 Comparison with Increment 2

This section intends to compare the results of the Base Configuration (section 3.5.2.2.1.1), when the increment of components (VMs) changes to a more aggressive increase (and decrease), i.e. each time a scaling out or in are performed, 2 VMs are added or released, respectively.

This section considers scaling tests with the following parameterization.

- Scaling trigger: number of requests (Req);
- Traffic generation: Profile 1 (high increasing and decreasing rate).
- Scaling increments: +-2 VM;

6.1.2.1 Scale out

The Figure 88 depicts the time elapsed to scale out an ANDSFaaS instance on the WS tier, showing the WS2 and WS3 instances (WS1 was already instantiated) and the 3 instantiation steps (OS, VM Ready and Service Ready).

As a result, it can be seen that the scaling out of 2 simultaneous components (VMs) take 160 seconds, 2 minutes 40 seconds. This is around the same time it takes to scale a single machine (150s). That means that this should be a good solution when a quick scaling is required.

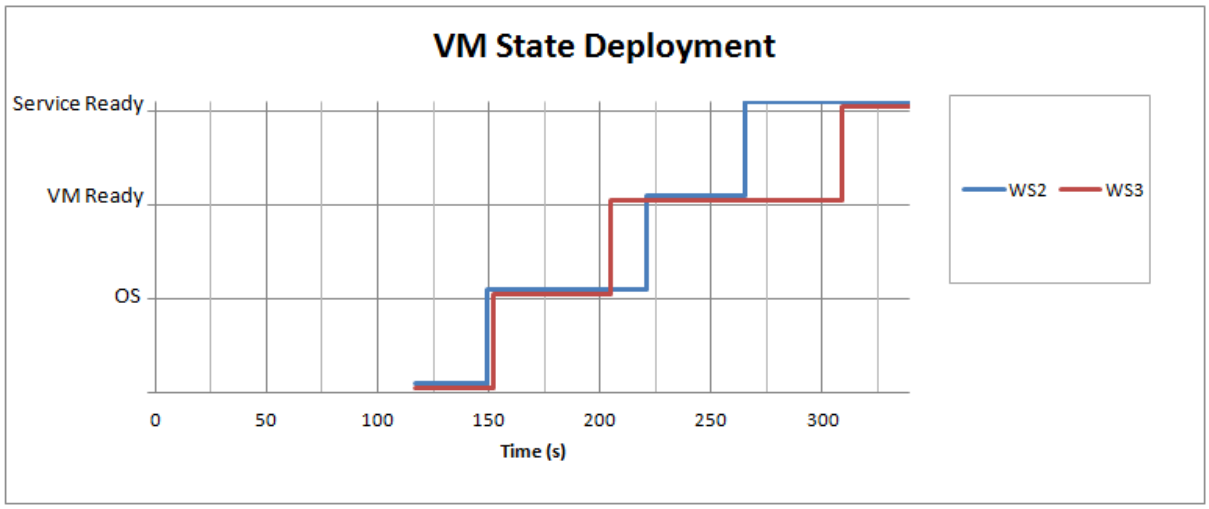

Figure 88 – ANDSFaaS: Scale WS, Trigger Req, Profile 1, Increment 2 (scale out)

6.1.2.2 Scale in

The Figure 89 depicts the time elapsed to scale in an ANDSFaaS instance on the WS tier, showing the WS3 and WS2 instances (WS1 was already instantiated).

As a result, it can be seen that the scaling out of 2 simultaneous components take 30 seconds, which is around the same time of scaling in a single component (VM). This should be used when fast scaling in operations are required.

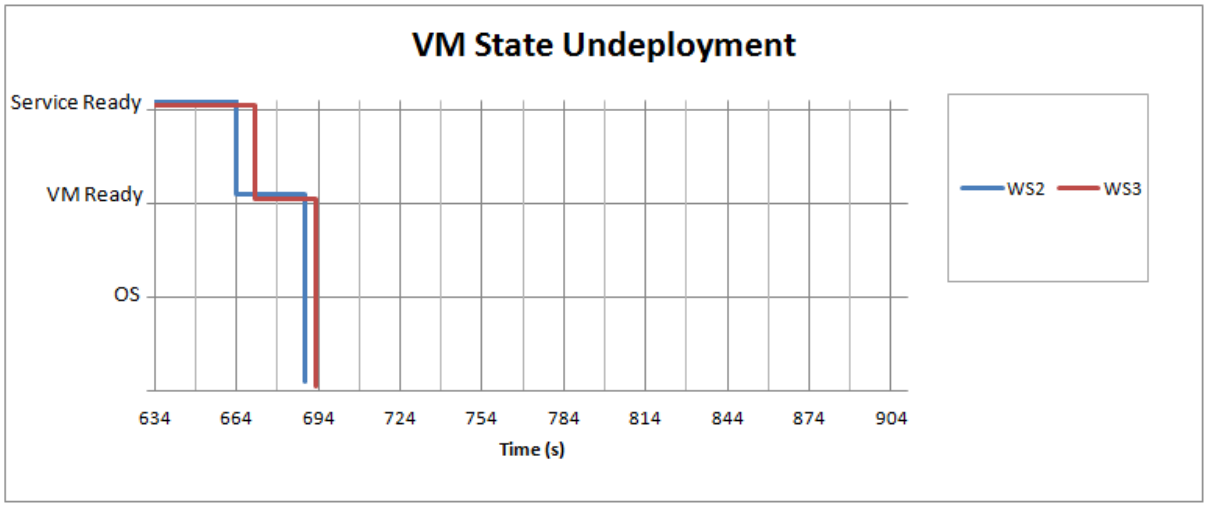

Figure 89 – ANDSFaaS: Scale WS, Trigger Req, Profile 1, Increment 2 (scale in)

6.1.2.3 Scale out and in

The Figure 90 shows the combination of scaling out and in of an ANDSFaaS instance on the WS tier, as already shown above.

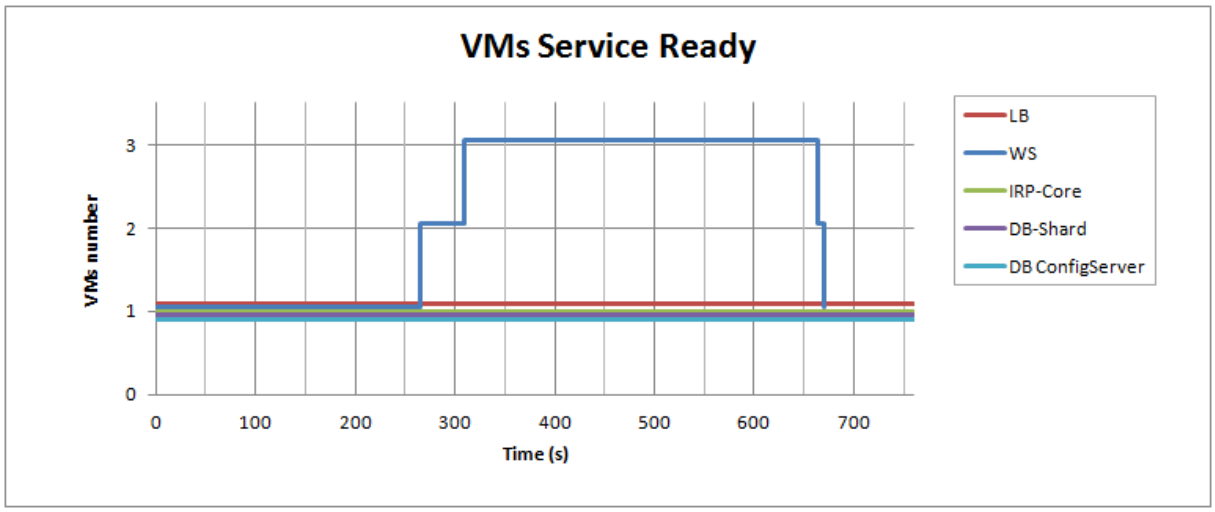

Figure 90 – ANDSFaaS: Scale WS, Trigger Req, Profile 1, Increment 2 (scale out and in)

6.1.2.4 Number of Requests

The Figure 91 depicts the service behaviour during ANDSF scaling in and out operations on the WS tier, showing the total number of requests and per WS component (VM).

As a result, it can be seen that service loss is very low and the traffic split among the WS components is quite balanced, improving the overall performance.

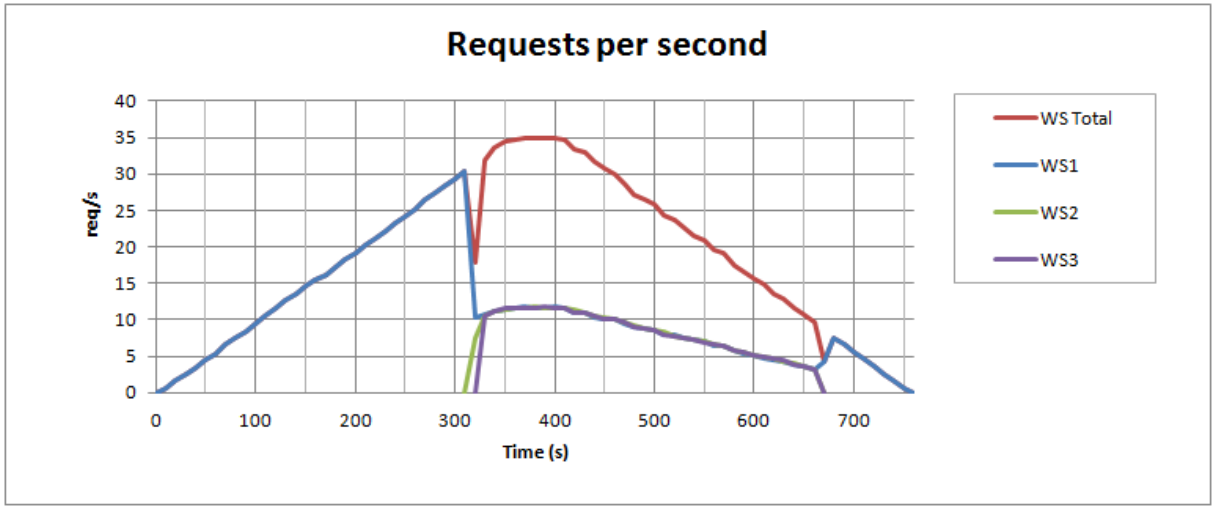

Figure 91 – ANDSFaaS: Scale WS, Trigger Req, Profile 1, Increment 2 (number of requests)

6.1.2.5 Impacts on Physical Machine

The Figures below show the impact of the ANDSFaaS scaling operations on the physical machine resources. In particular, the Figure 92, Figure 93 and Figure 94 show the impacts on CPU, RAM and I/O utilization, respectively, along the deployment and release periods.

As a result, it can be seen that the most relevant impact on the physical machine relates to I/O utilization, although quite less significant than for the deployment operations. The CPU and RAM are slightly affected. It can also be seen that scale-in operations require less resources than scale out.

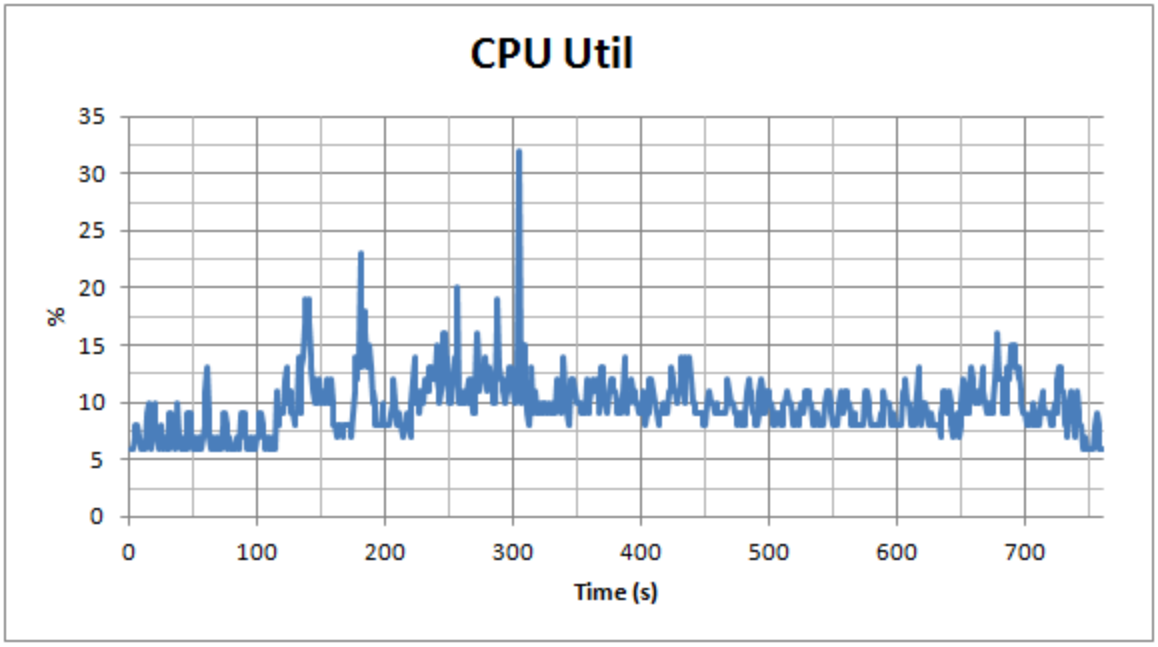

Figure 92 – ANDSFaaS: Scale WS, Trigger Req, Profile 1, Increment 2 (PM CPU utilization)

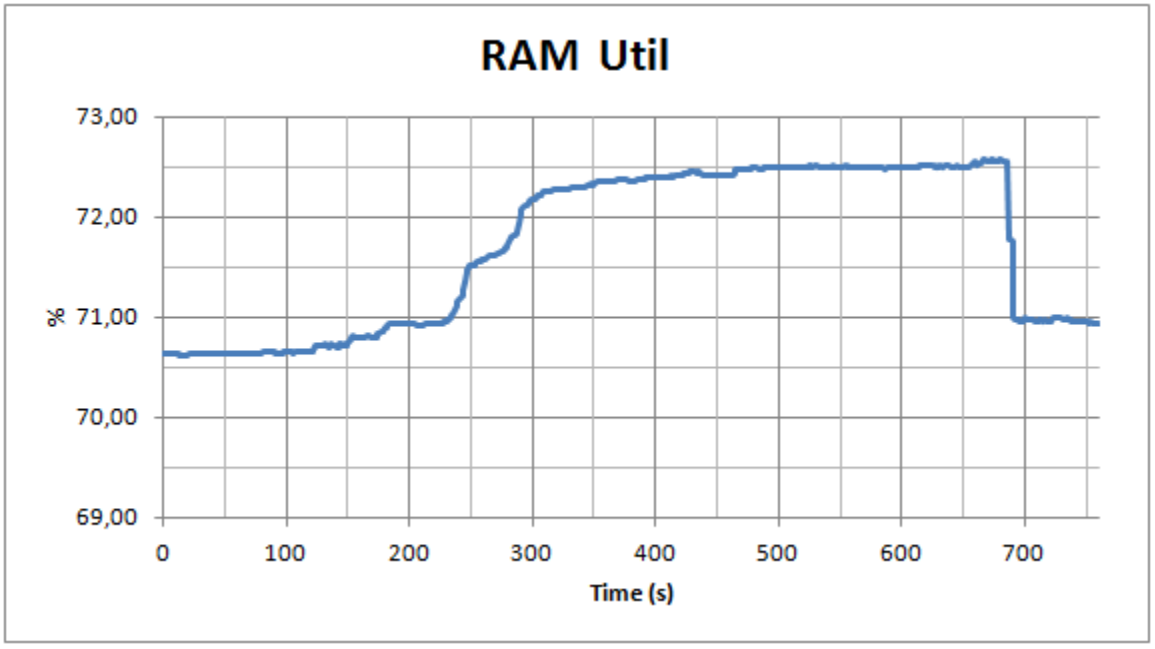

Figure 93 – ANDSFaaS: Scale WS, Trigger Req, Profile 1, Increment 2 (PM RAM utilization)

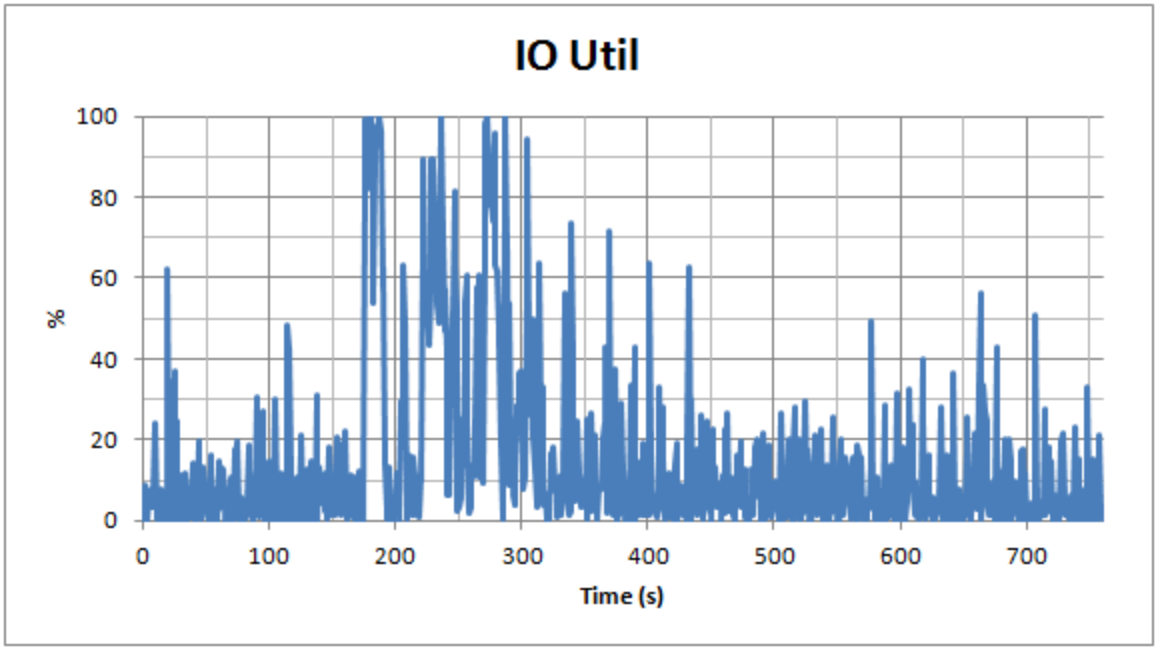

Figure 94 – ANDSFaaS: Scale WS, Trigger Req, Profile 1, Increment 2 (PM I/O utilization)

6.1.3 Comparison with Trigger CPU

This section intends to compare the results of the Base Configuration (section 3.5.2.2.1.1), when the monitoring event that triggers scaling operations is the CPU (instead the Number of Requests).

This section considers scaling tests with the following parameterization..

- Traffic generation: Profile 1 (medium increasing and decreasing rate).
- Scaling increments: +-1 VM;
- Scaling trigger: CPU utilization (CPU);

6.1.3.1 Scale out

The Figure 95 depicts the time elapsed to scale out an ANDSFaaS instance on the WS tier, showing the WS2 and WS3 instances (WS1 was already instantiated) and the 3 instantiation steps (OS, VM Ready and Service Ready).

As a result, it can be seen that scaling out based on CPU threshold is also possible, although is harder to control (for testing purposes). It is also important to note that average CPU values must be used in order to avoid high variations and correspondent unstable scaling policies.

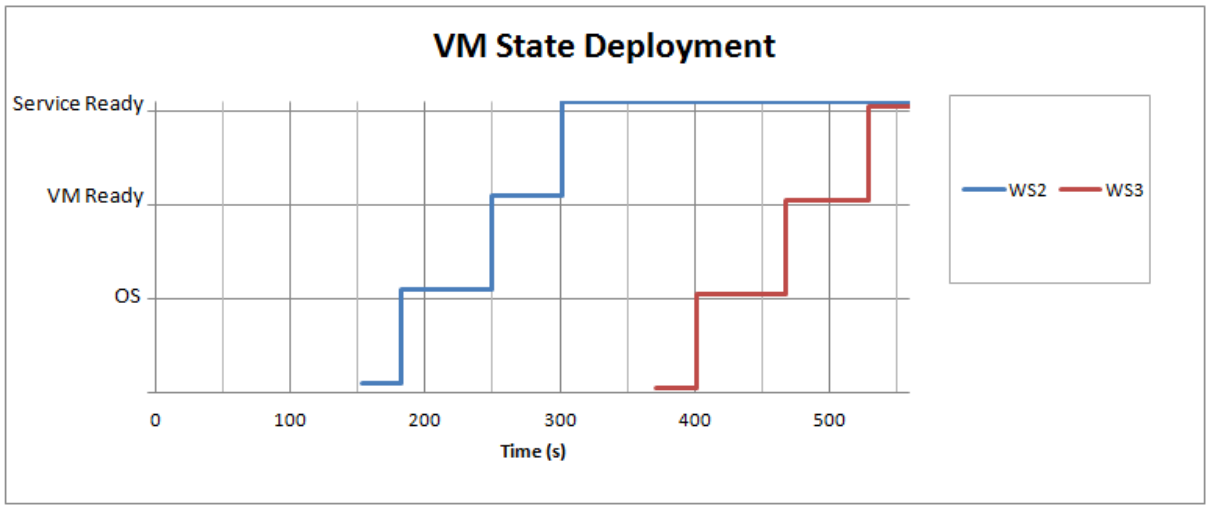

Figure 95 – ANDSFaaS: Scale WS, Increment 1, Profile 1, Trigger CPU (scale out)

6.1.3.2 Scale in

The Figure 96 depicts the time elapsed to scale in an ANDSFaaS instance on the WS tier, showing the WS3 and WS2 instances (WS1 was already instantiated).

As a result, it can be seen that scaling out operations are also possible based on CPU.

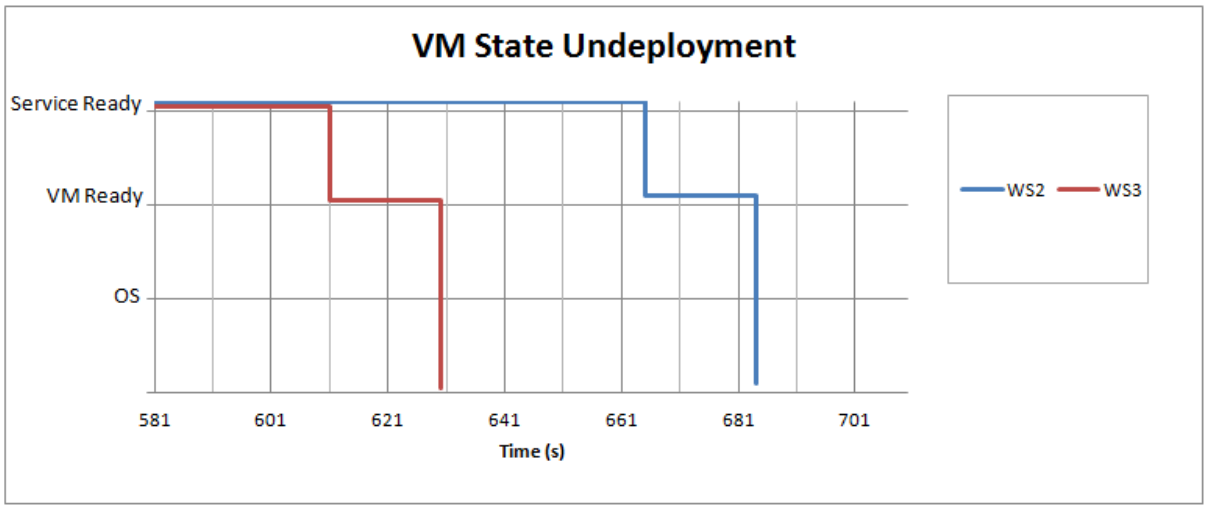

Figure 96 – ANDSFaaS: Scale WS, Increment 1, Profile 1, Trigger CPU (scale in)

6.1.3.3 Scale out and in

The Figure 97 shows the combination of scaling out and in an ANDSFaaS instance on the WS tier, as already shown above.

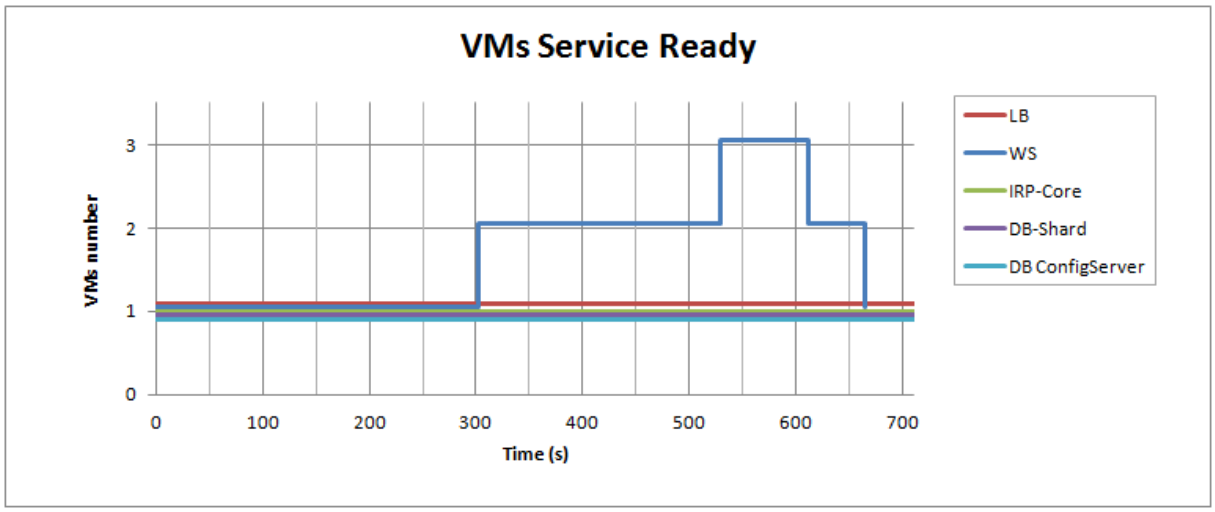

Figure 97 – ANDSFaaS: Scale WS, Increment 1, Profile 1, Trigger CPU (scale out and in)

6.1.3.4 CPU Utilization

The Figure 98 depicts the service behaviour during ANDSF scaling in and out operations on the WS tier, showing the total CPU utilization and per WS component (VM).

As a result, it can be seen that the CPU utilization thresholds are applied and the CPU load is quite balanced among the multiple components. However, it takes more time to balance than using the number of requests per second (Req).

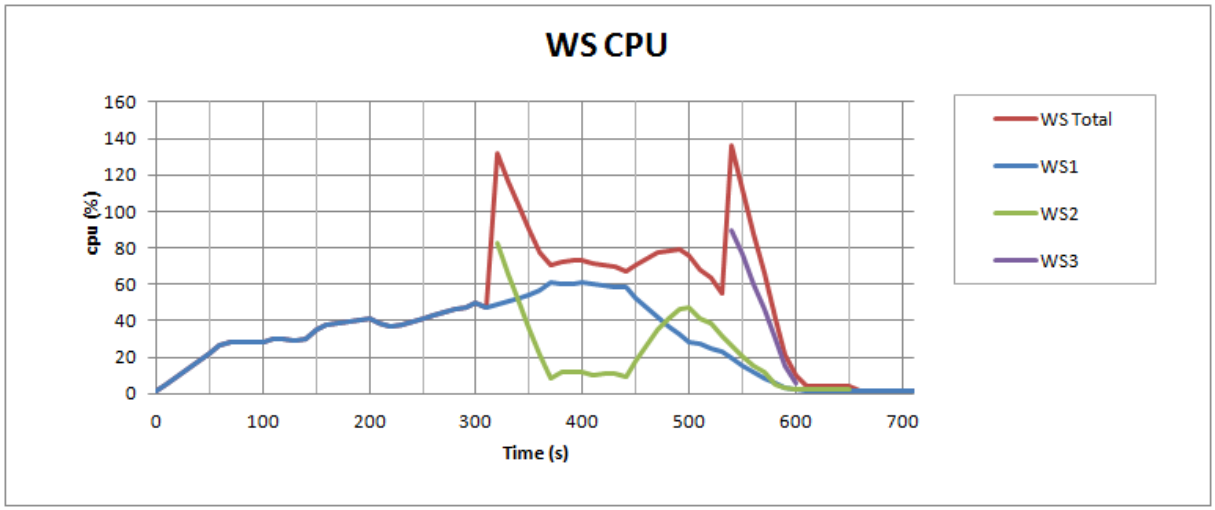

Figure 98 – ANDSFaaS: Scale WS, Increment 1, Profile 1, Trigger CPU (number of requests)

6.1.3.5 Impacts on Physical Machine

The Figures below show the impact of the ANDSFaaS scaling operations on the physical machine resources. In particular, the Figure 99, Figure 100 and Figure 101 show the impacts on CPU, RAM and I/O utilization along the deployment period.

As a result, it can be seen that the results are similar to the triggering based on the number of requests.

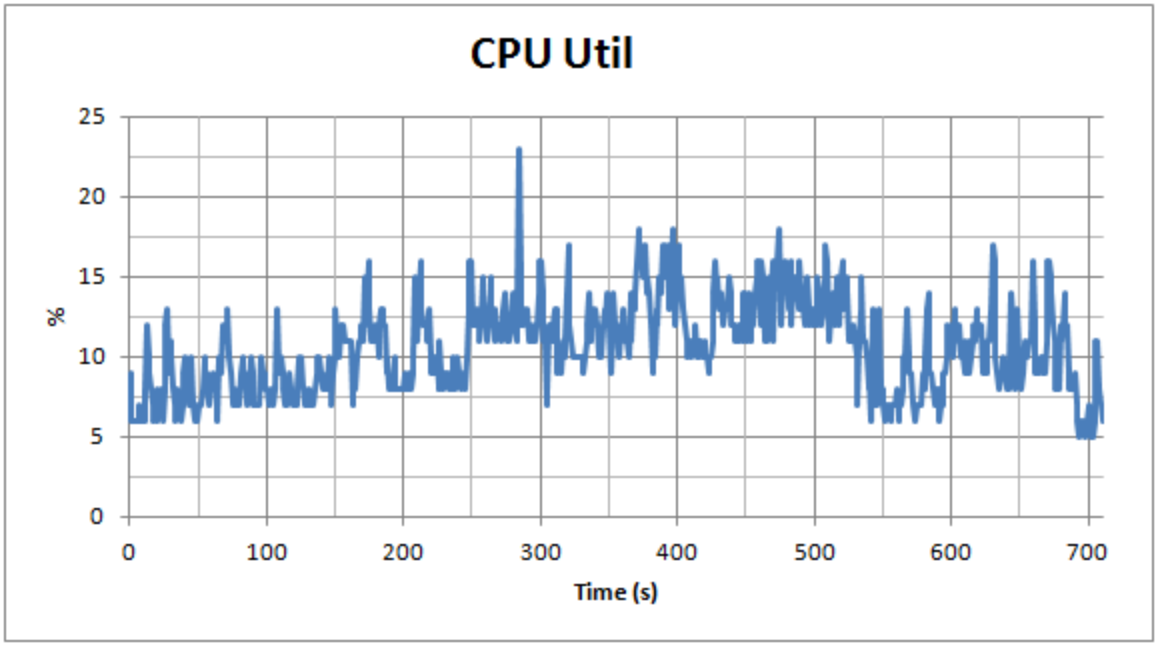

Figure 99 – ANDSFaaS: Scale WS, Profile 1, Increment 1, Trigger CPU (PM CPU utilization)

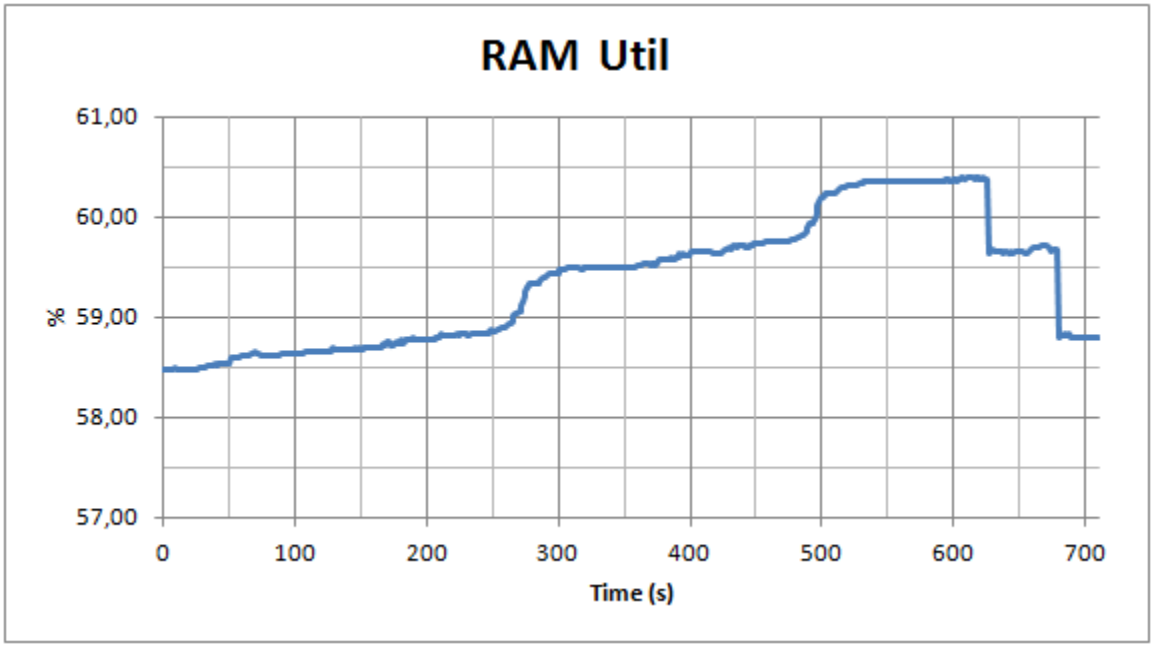

Figure 100 – ANDSFaaS: Scale WS, Profile 1, Increment 1, Trigger CPU (PM RAM utilization)

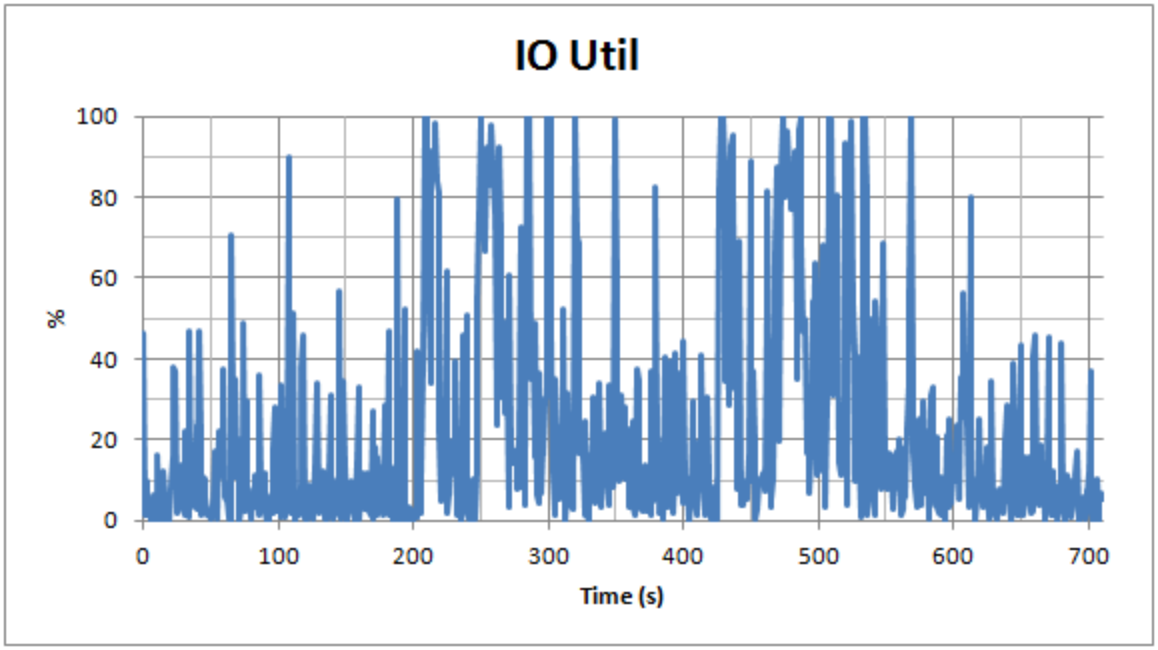

Figure 101 – ANDSFaaS: Scale WS, Profile 1, Increment 1, Trigger CPU (PM I/O utilization)

6.2 IRP Scaling

6.2.1 Comparison with Profile 2

This section intends to compare the results of the Base Configuration (section 3.5.2.2.1.1), when the traffic profile changes to a less aggressive increase (and decrease) in the number of requests.

This section considers scaling tests with the following parameterization.

- Scaling trigger: number of requests (Req);
- Scaling increments: +-1 VM;
- Traffic generation: Profile 2 (low increasing and decreasing rate).

6.2.1.1 Scale out

The Figure 102 depicts the time elapsed to scale out an ANDSFaaS instance on the IRP tier, showing the IRP2 and IRP3 instances (IRP1 was already instantiated) and the 3 instantiation steps (OS, VM Ready and Service Ready).

As a result, it can be seen that the scaling out decision is taken later, considering that the threshold on the number of requests takes more time to be reached when compared with the Base Configuration (section 3.5.2.2.2.1). For example, here the first scaling out occurs around the 160s, while with Profile 1 it is around the 125s. A similar effect occurs for the second scaling out operation.

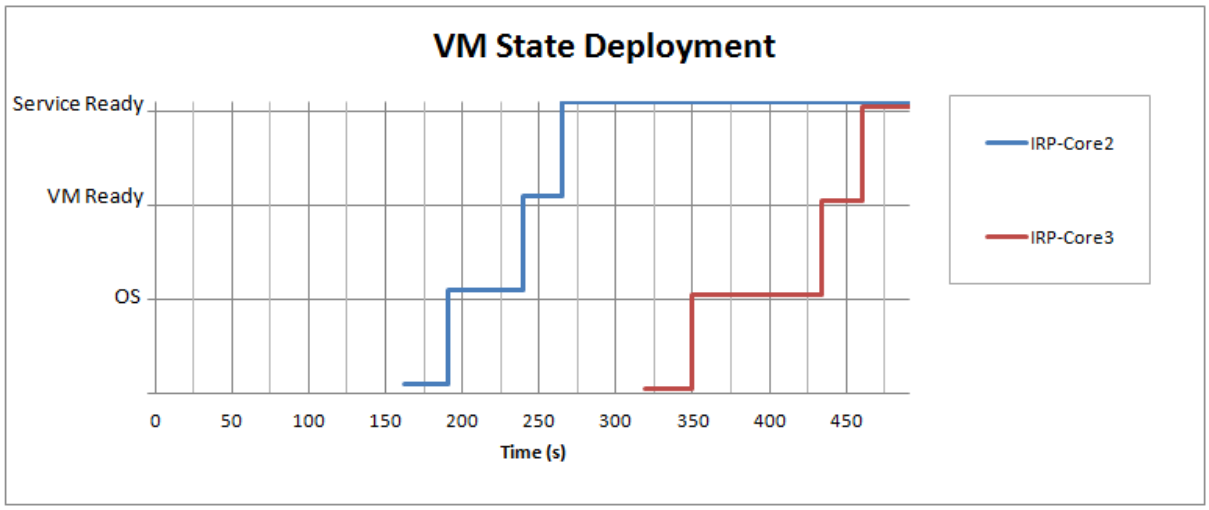

Figure 102 – ANDSFaaS: Scale IRP, Trigger Req, Increment 1, Profile 2 (scale out)

6.2.1.2 Scale in

The Figure 103 depicts the time elapsed to scale in an ANDSFaaS instance on the IRP tier, showing the IRP3 and IRP2 instances (IRP1 was already instantiated).

As a result, it can be seen that scaling in decision is taken later, considering that the threshold on the number of requests takes more time to be reached when compared with the Base Configuration (section 3.5.2.2.2.1. For example, here the first scaling in occurs around 965s, while with Profile 1 it is around 655s. The same occurs for the second scaling in operation.

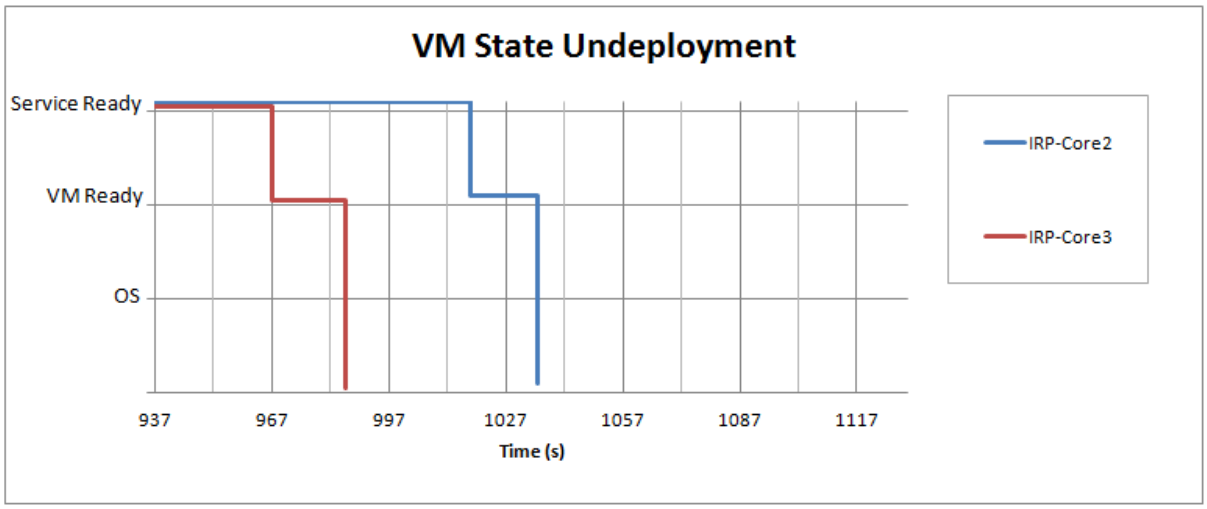

Figure 103 – ANDSFaaS: Scale IRP, Trigger Req, Increment 1, Profile 2 (scale in)

6.2.1.3 Scale out and in

The Figure 104 shows the combination of scaling out and in an ANDSFaaS instance on the IRP tier, as already shown above.

Compared to the usage of Profile 1 (high increasing rate), and as expected, the scaling operations are triggered later.

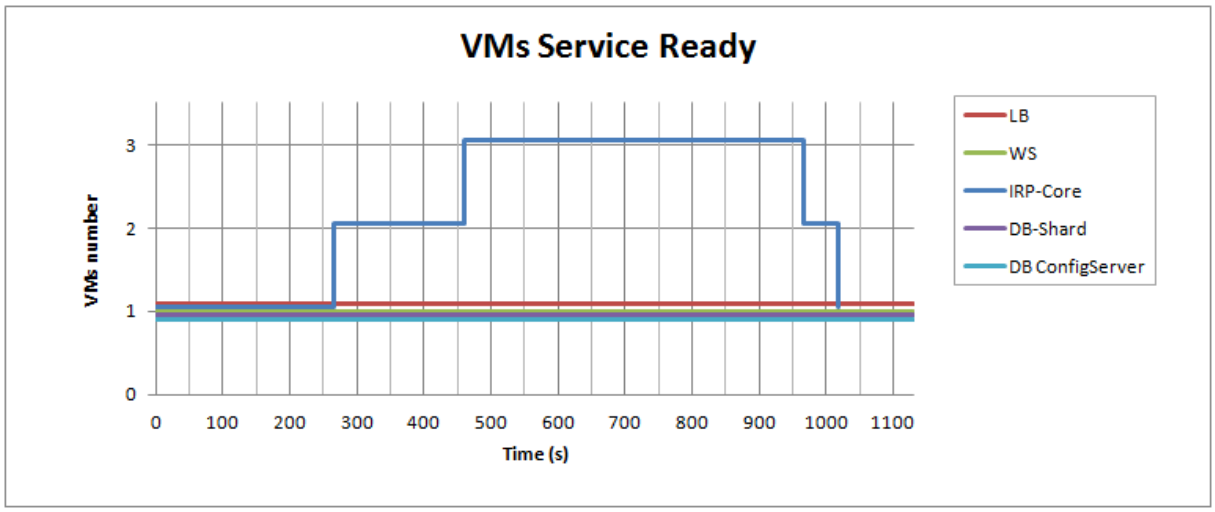

Figure 104 – ANDSFaaS: Scale IRP, Trigger Req, Increment 1, Profile 2 (scale out and in)

6.2.1.4 Number of Requests

The Figure 105 depicts the service behaviour during ANDSF scaling in and out operations on the IRP tier, showing the total number of requests and per IRP VM.

As a result, it can be seen that service loss is similar as for the case of Profile 1.

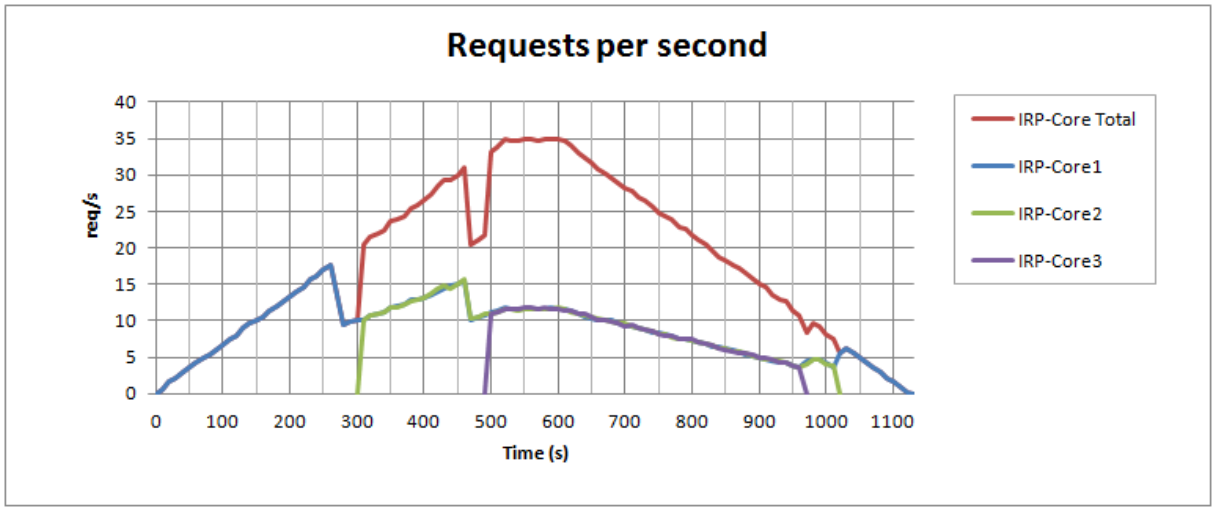

Figure 105 – ANDSFaaS: Scale IRP, Trigger Req, Increment 1, Profile 2 (number of requests)

6.2.1.5 Impacts on Physical Machine

The Figures below show the impact of the ANDSFaaS scaling operations on the physical machine resources. In particular, the Figure 106, Figure 107 and Figure 108 show the impacts on CPU, RAM and I/O utilization, respectively, along the deployment and release periods.

As a result, it can be seen that the most relevant impact on the physical machine relates to I/O utilization, although quite less significant than for the deployment operations. The CPU and RAM are slightly affected. It can also be seen that scale-in operations require less resources than scale out.

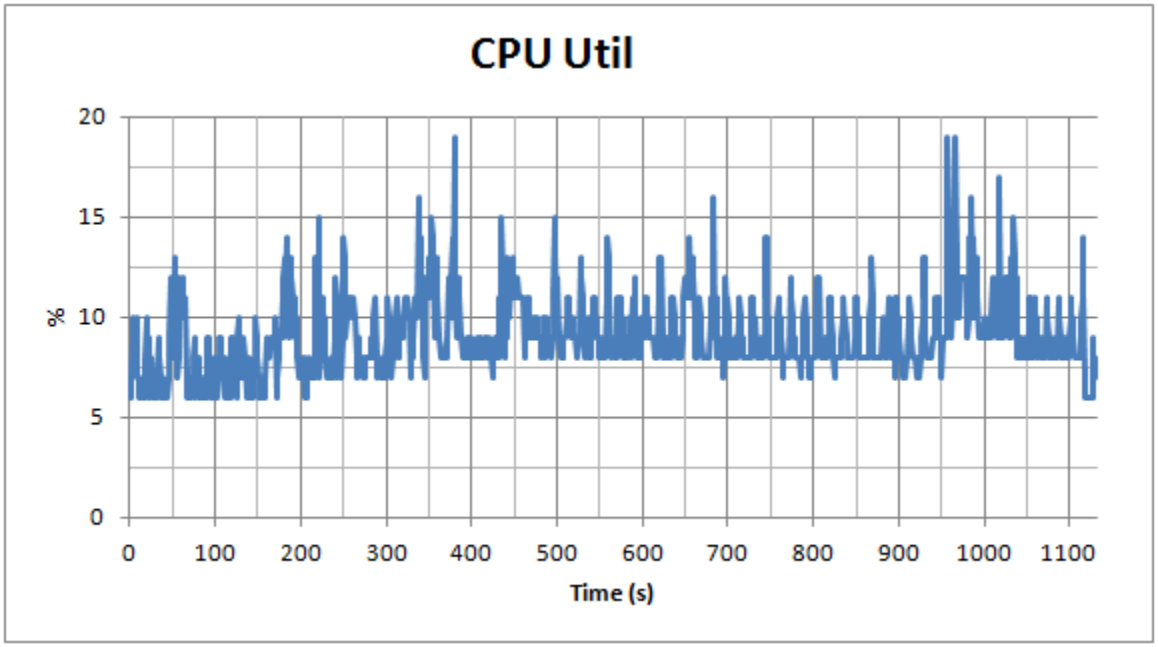

Figure 106 – ANDSFaaS: Scale IRP, Trigger Req, Increment 1, Profile 2 (PM CPU utilization)

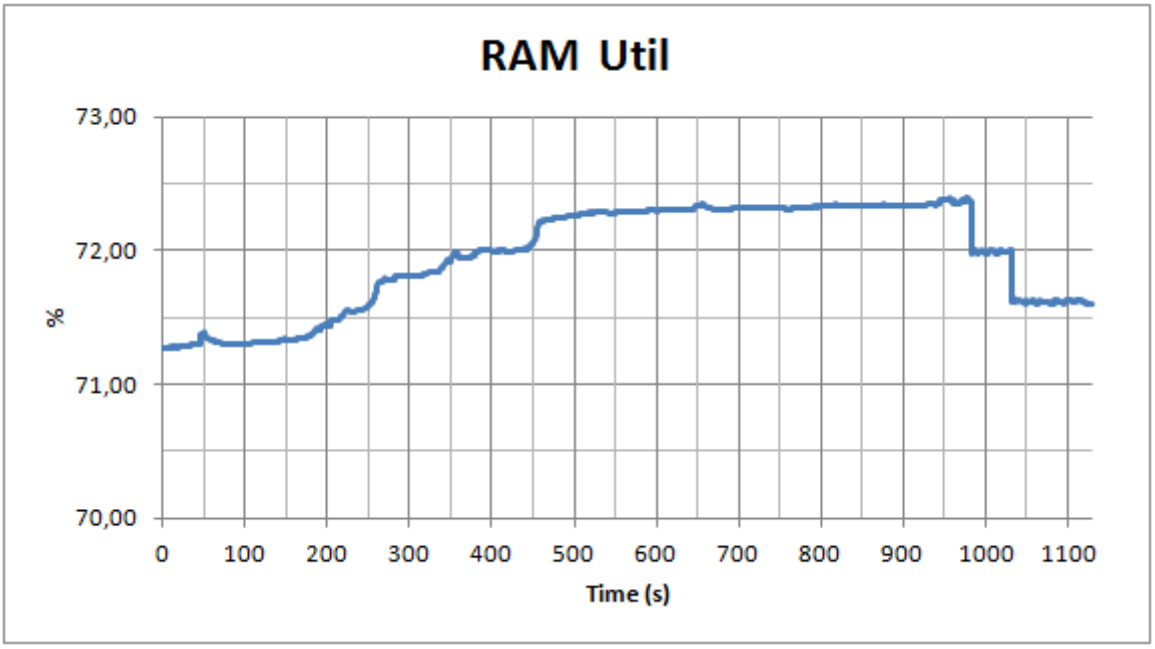

Figure 107 – ANDSFaaS: Scale IRP, Trigger Req, Increment 1, Profile 2 (PM RAM utilization)

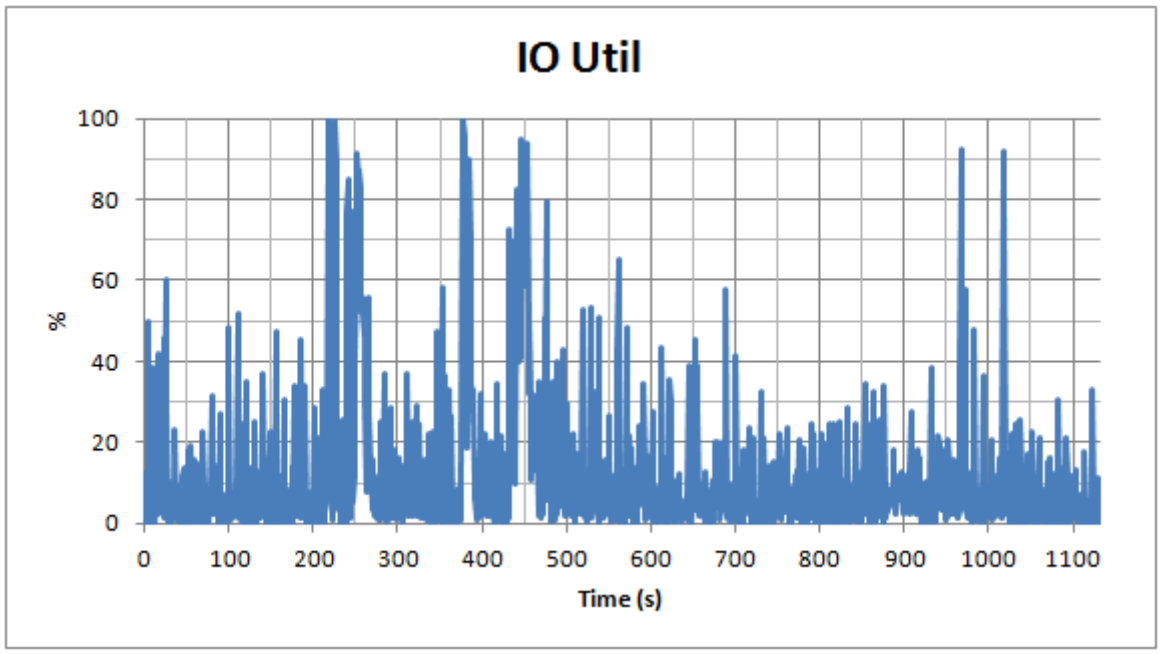

Figure 108 – ANDSFaaS: Scale IRP, Trigger Req, Increment 1, Profile 2 (PM I/O utilization)

6.2.2 Comparison with Increment 2

This section intends to compare the results of the Base Configuration (section 3.5.2.2.2.1), when the increment of components (VMs) changes to a more aggressive increase (and decrease), i.e. each time a scaling out or in are performed, 2 VMs are added or released, respectively.

This section considers scaling tests with the following parameterization.

- Scaling trigger: number of requests (Req);
- Traffic generation: Profile 1 (high increasing and decreasing rate).
- Scaling increments: +-2 VM;

6.2.2.1 Scale out

The Figure 109 depicts the time elapsed to scale out an ANDSFaaS instance on the IRP tier, showing the IRP2 and IRP3 instances (IRP1 was already instantiated) and the 3 instantiation steps (OS, VM Ready and Service Ready).

As a result, it can be seen that the scaling out of 2 simultaneous components (VMs) take 180 seconds, 3 minutes. This is around the same time it takes to scale a single machine (160s). That means that this should be the solution when a quick scaling is required.

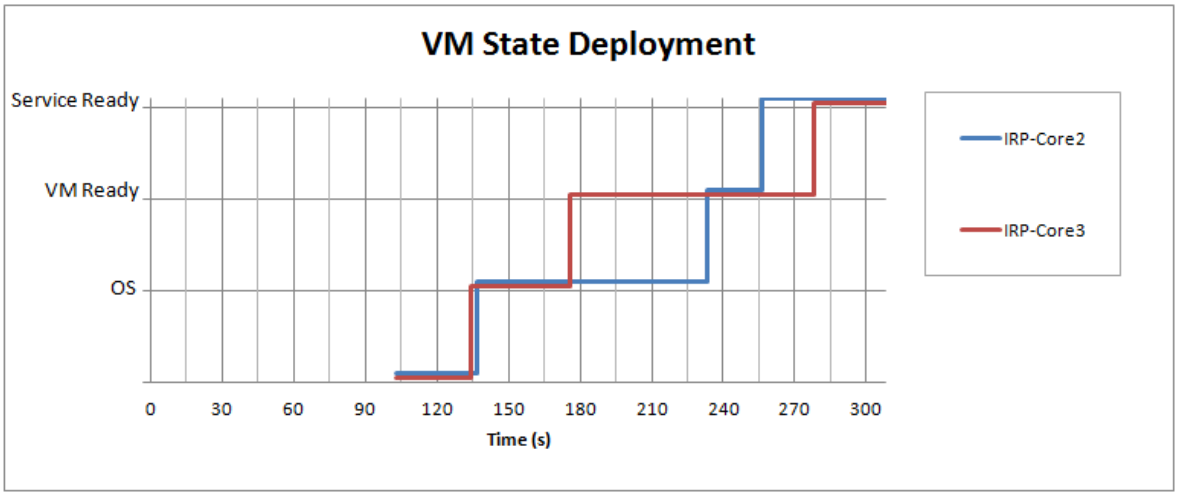

Figure 109 – ANDSFaaS: Scale IRP, Trigger Req, Profile 1, Increment 2 (scale out)

6.2.2.2 Scale in

The Figure 110 depicts the time elapsed to scale in an ANDSFaaS instance on the IRP tier, showing the IRP3 and IRP2 instances (IRP1 was already instantiated).

As a result, it can be seen that scaling in operations take 30 seconds, which is 50% more time than scaling in a single component (20s). This should be used when fast scaling in operations are required.

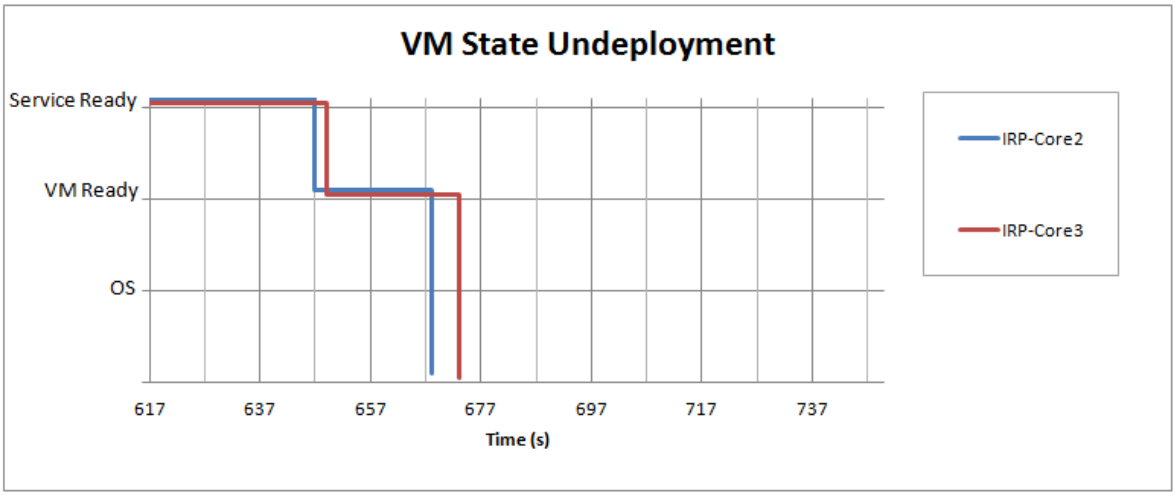

Figure 110 – ANDSFaaS: Scale IRP, Trigger Req, Profile 1, Increment 2 (scale in)

6.2.2.3 Scale out and in

The Figure 111 shows the combination of scaling out and in of an ANDSFaaS instance on the IRP tier, as already shown above.

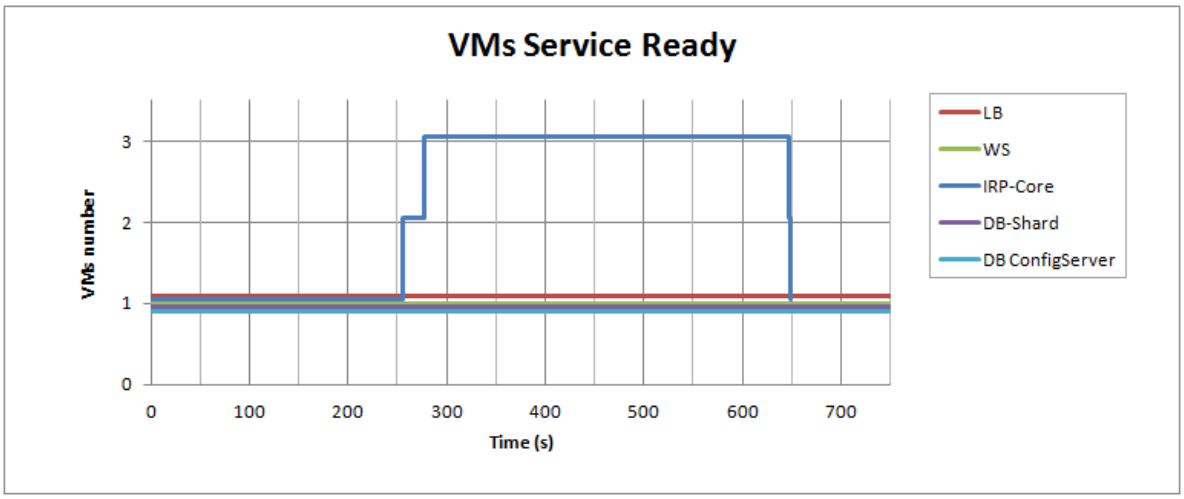

Figure 111 – ANDSFaaS: Scale IRP, Trigger Req, Profile 1, Increment 2 (scale out and in)

6.2.2.4 Number of Requests

The Figure 112 depicts the service behaviour during ANDSF scaling in and out operations on the IRP tier, showing the total number of requests and per IRP component (VM).

As a result, it can be seen that service loss is very low and the traffic split among the IRP components is quite balanced, improving the overall performance.

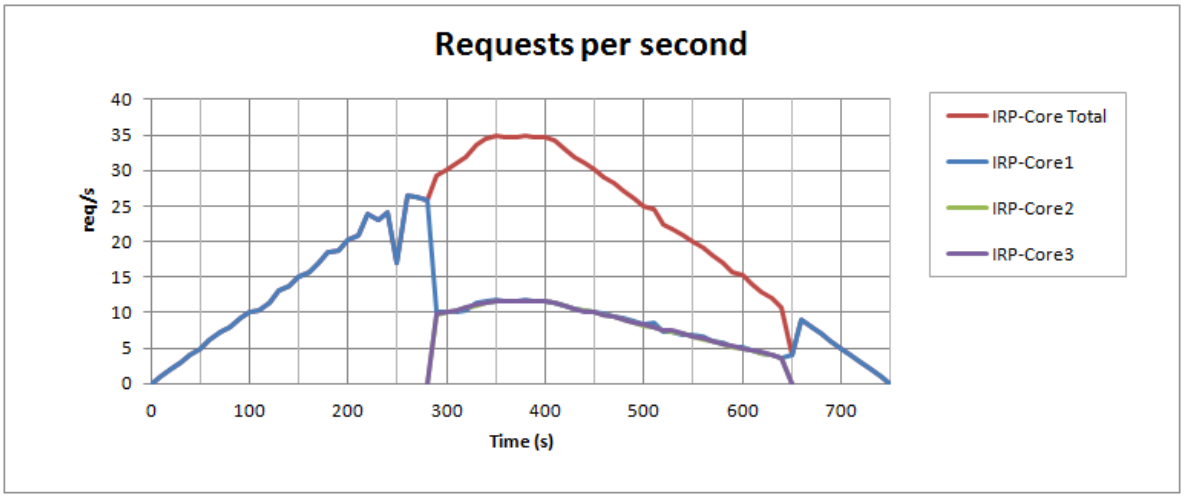

Figure 112 – ANDSFaaS: Scale IRP, Trigger Req, Profile 1, Increment 2 (number of requests)

6.2.2.5 Impacts on Physical Machine

The Figures below show the impact of the ANDSFaaS scaling operations on the physical machine resources. In particular, the Figure 113, Figure 114 and Figure 115 show the impacts on CPU, RAM and I/O utilization, respectively, along the deployment and release periods.

As a result, it can be seen that the most relevant impact on the physical machine relates to I/O utilization, although quite less significant than for the deployment operations. The CPU and RAM are slightly affected. It can also be seen that scale-in operations require less resources than scale out.

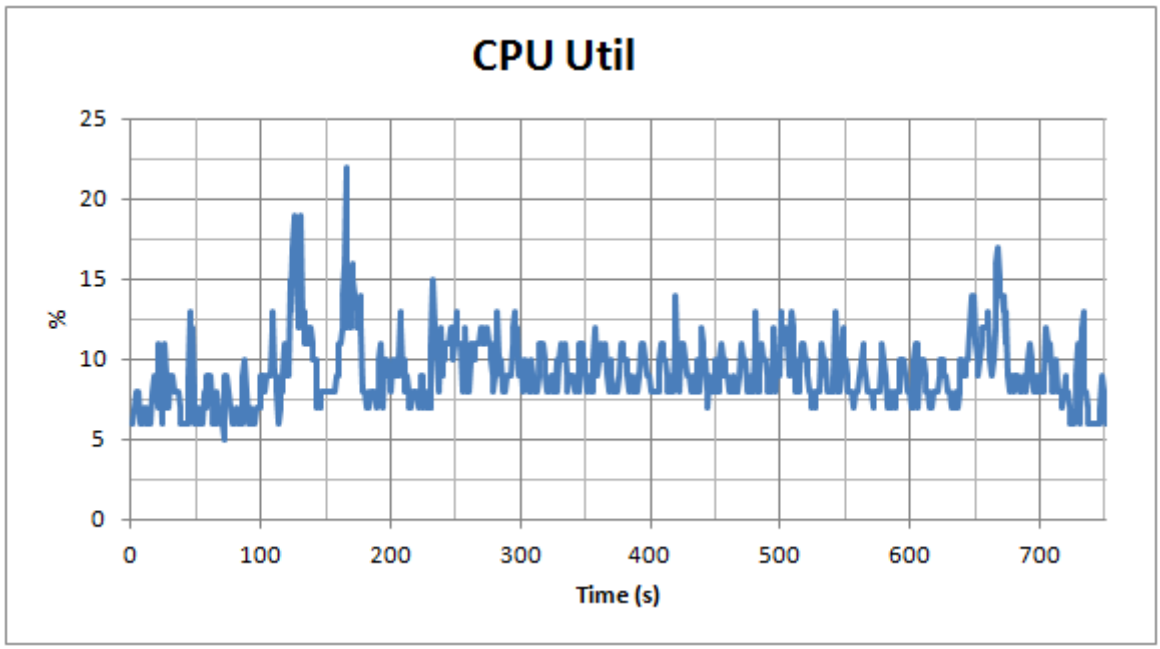

Figure 113 – ANDSFaaS: Scale IRP, Trigger Req, Profile 1, Increment 2 (PM CPU utilization)

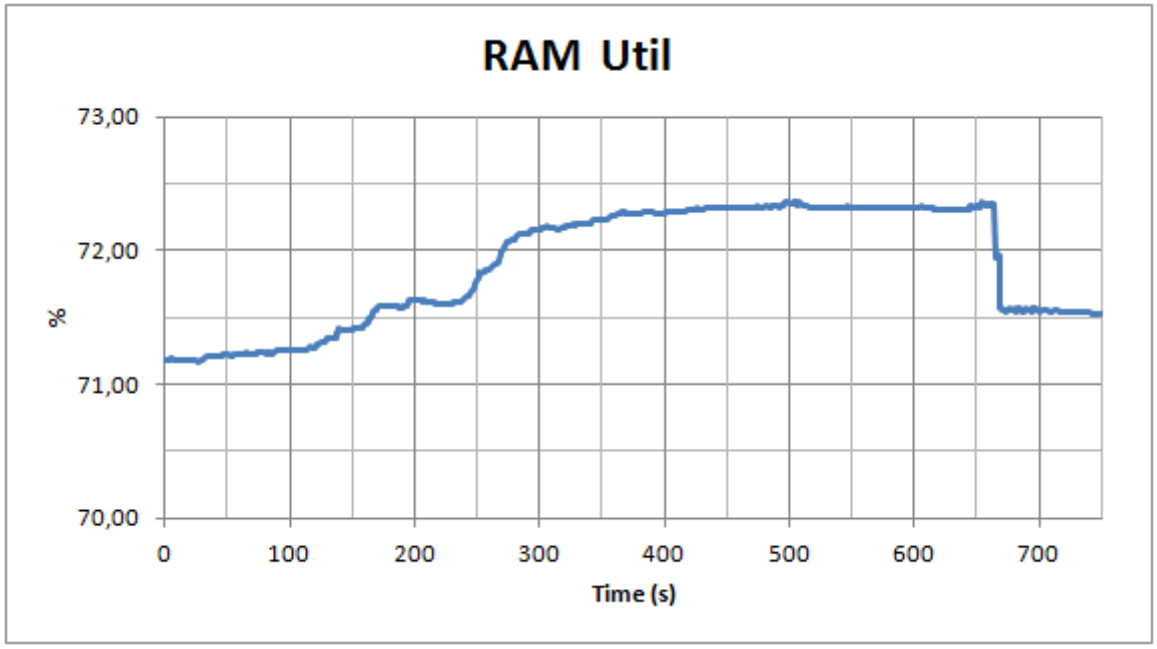

Figure 114 – ANDSFaaS: Scale IRP, Trigger Req, Profile 1, Increment 2 (PM RAM utilization)

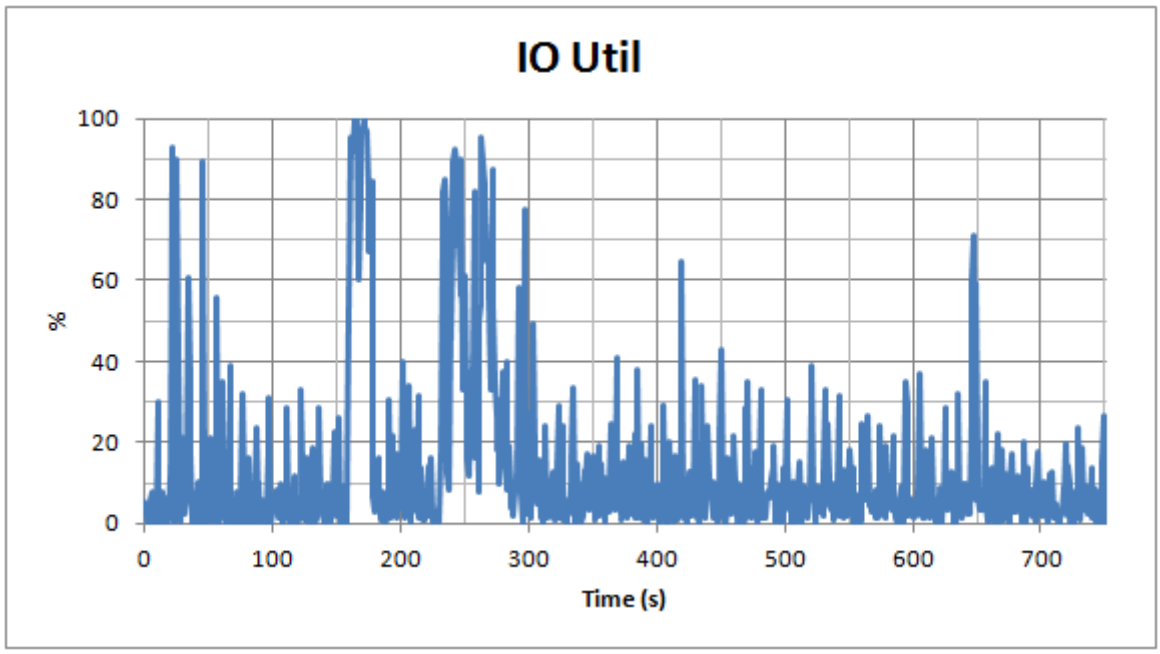

Figure 115 – ANDSFaaS: Scale IRP, Trigger Req, Profile 1, Increment 2 (PM I/O utilization)

6.2.3 Comparison with Trigger CPU

This section intends to compare the results of the Base Configuration (section 3.5.2.2.2.1), when the monitoring event that triggers scaling operations is the CPU (instead the Number of Requests).

This section considers scaling tests with the following parameterization.

- Traffic generation: Profile 1 (medium increasing and decreasing rate).
- Scaling increments: +-1 VM;
- Scaling trigger: CPU utilization (CPU);

6.2.3.1 Scale out

The Figure 116 depicts the time elapsed to scale out an ANDSFaaS instance on the IRP tier, showing the IRP2 and IRP3 instances (IRP1 was already instantiated) and the 3 instantiation steps (OS, VM Ready and Service Ready).

As a result, it can be seen that scaling out based on CPU threshold is also possible, although is harder to control (for testing purposes). It is also important to note that average CPU values must be used in order to avoid high variations and correspondent unstable scaling policies.

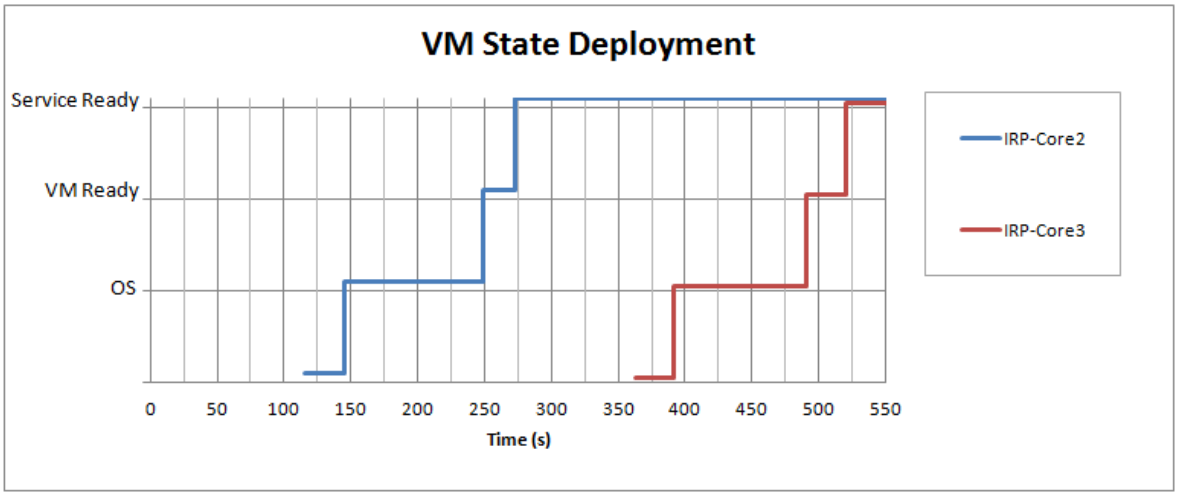

Figure 116 – ANDSFaaS: Scale IRP, Increment 1, Profile 1, Trigger CPU (scale out)

6.2.3.2 Scale in

The Figure 117 depicts the time elapsed to scale in an ANDSFaaS instance on the IRP tier, showing the IRP3 and IRP2 instances (IRP1 was already instantiated).

As a result, it can be seen that scaling out operations are also possible based on CPU.

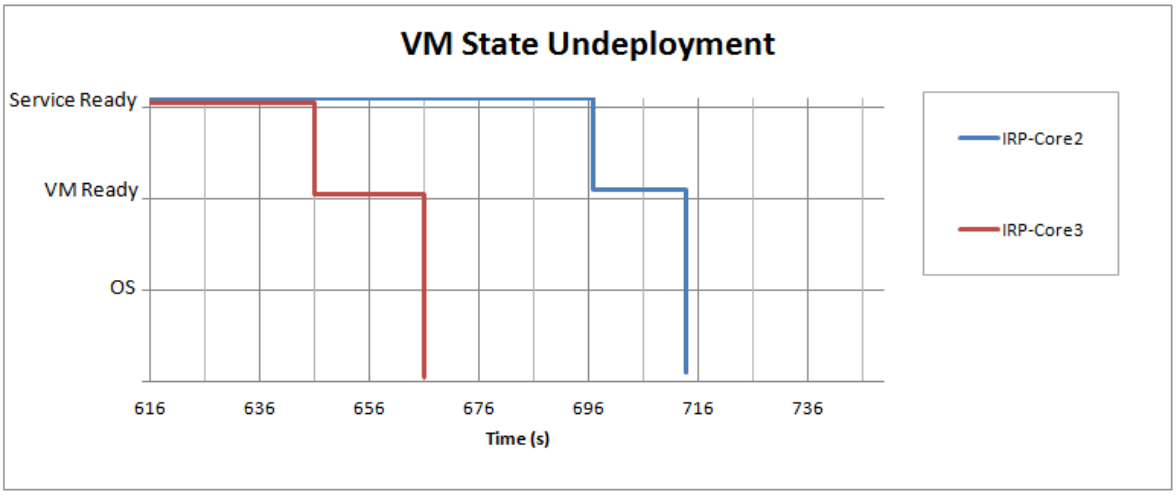

Figure 117 – ANDSFaaS: Scale IRP, Increment 1, Profile 1, Trigger CPU (scale in)

6.2.3.3 Scale out and in

The Figure 118 shows the combination of scaling out and in an ANDSFaaS instance on the IRP tier, as already shown above.

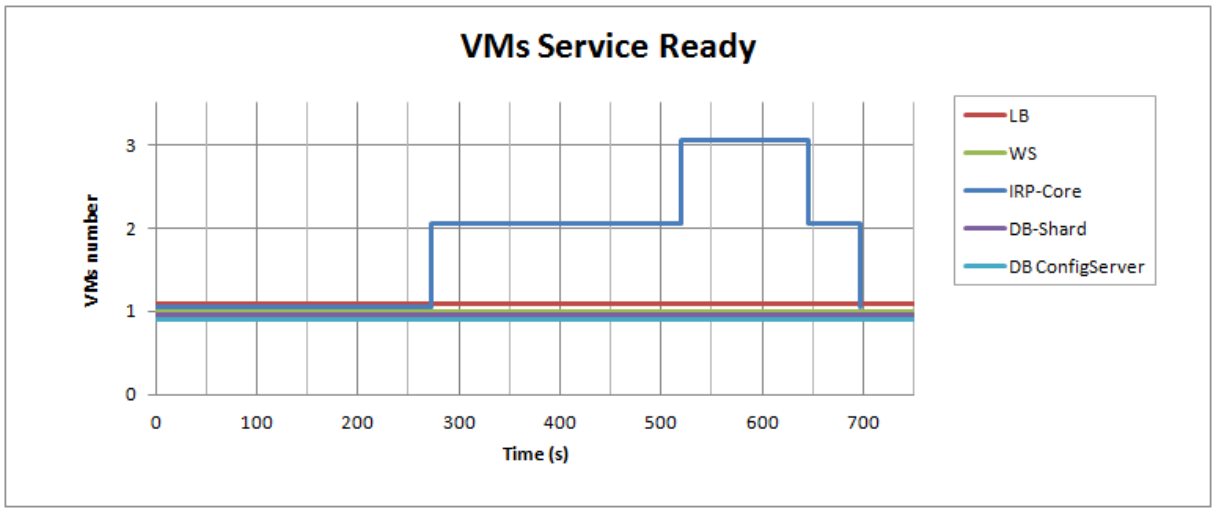

Figure 118 – ANDSFaaS: Scale IRP, Increment 1, Profile 1, Trigger CPU (scale out and in)

6.2.3.4 CPU Utilization

The Figure 119 depicts the service behaviour during ANDSF scaling in and out operations on the IRP tier, showing the total CPU utilization and per IRP component (VM).

As a result, it can be seen that the CPU utilization thresholds are applied and the CPU load is quite balanced among the multiple components. However, it takes more time to balance than using the number of requests per second (Req).

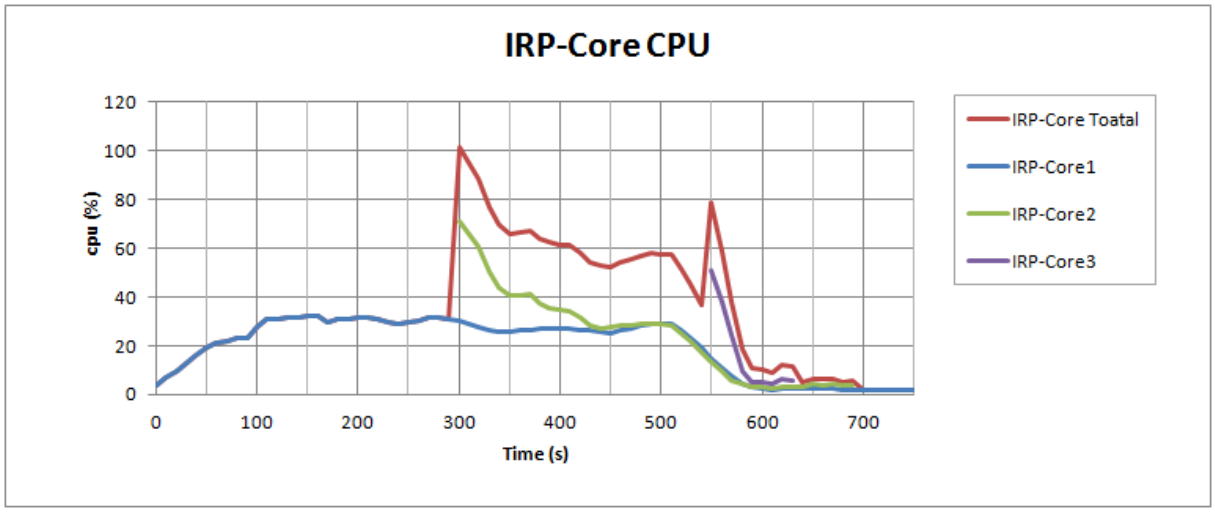

Figure 119 – ANDSFaaS: Scale IRP, Increment 1, Profile 1, Trigger CPU (number of requests)

6.2.3.5 Impacts on Physical Machine

The Figures below show the impact of the ANDSFaaS scaling operations on the physical machine resources. In particular, the Figure 120, Figure 121 and Figure 122 show the impacts on CPU, RAM and I/O utilization along the deployment period.

As a result, it can be seen that the results are similar to the triggering based on the number of requests.

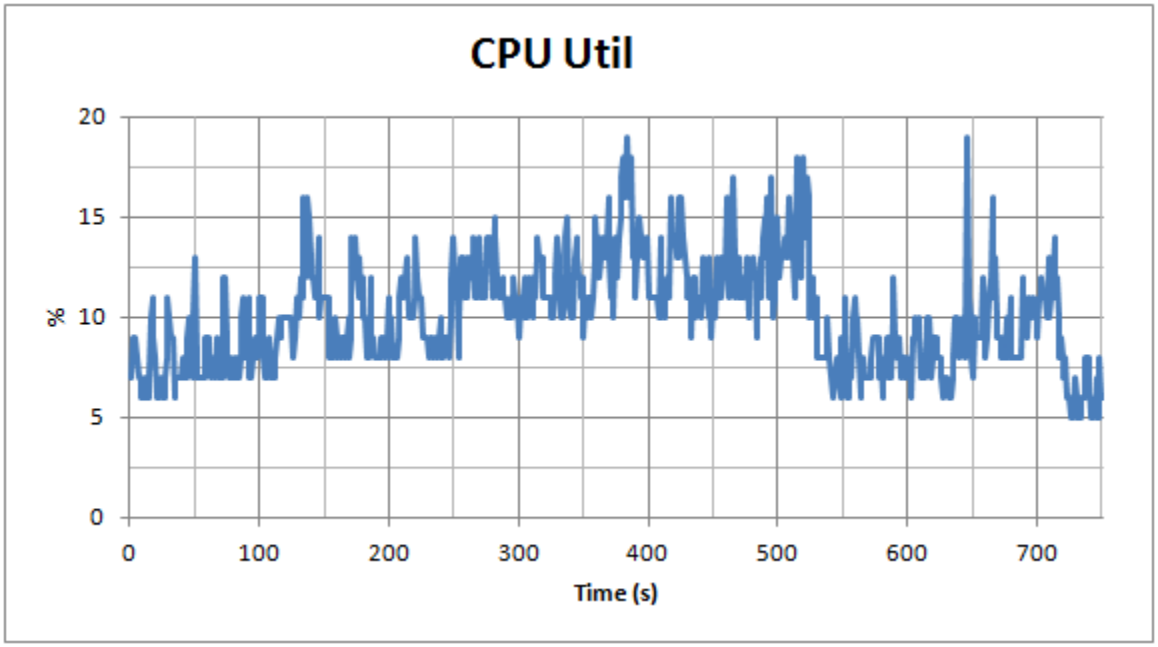

Figure 120 – ANDSFaaS: Scale IRP, Profile 1, Increment 1, Trigger CPU (PM CPU utilization)

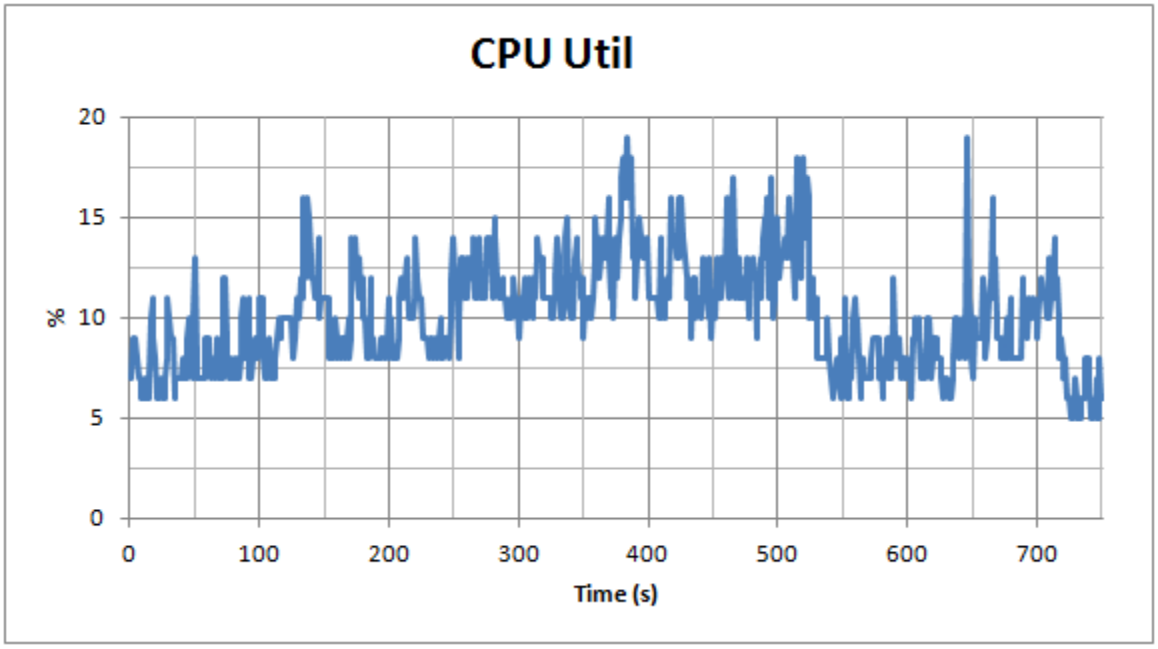

Figure 121 – ANDSFaaS: Scale IRP, Profile 1, Increment 1, Trigger CPU (PM RAM utilization)

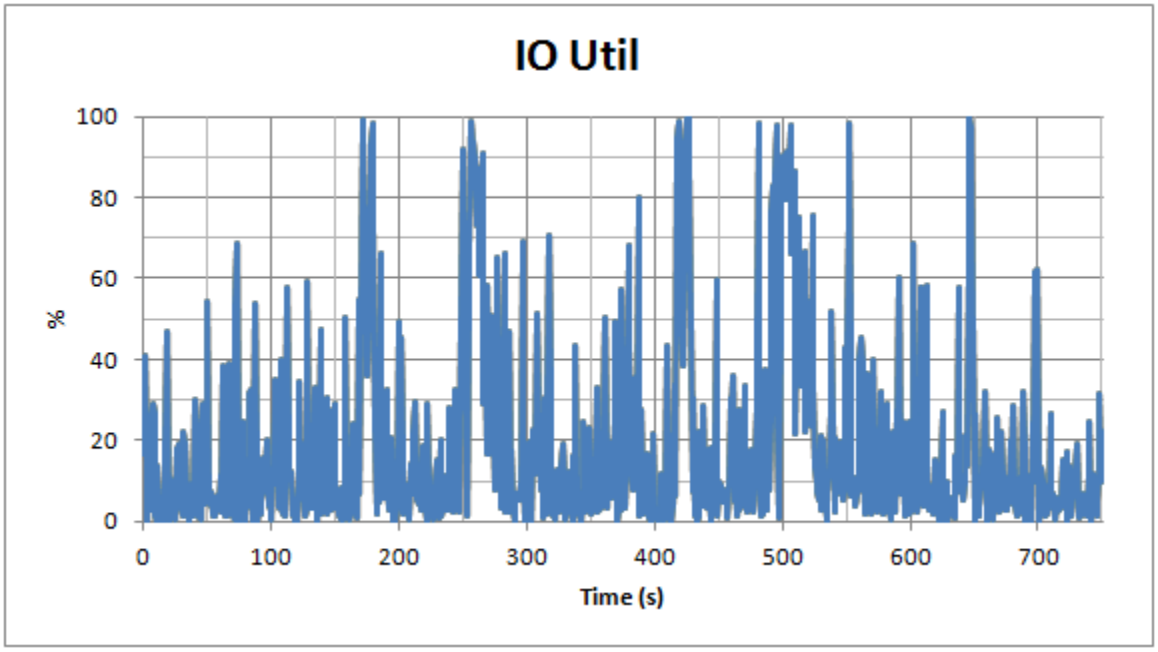

Figure 122 – ANDSFaaS: Scale IRP, Profile 1, Increment 1, Trigger CPU (PM I/O utilization)

6.3 DB Scaling

6.3.1 Comparison with Profile 2

This section intends to compare the results of the Base Configuration (section 3.5.2.2.3.1, when the traffic profile changes to a less aggressive increase (and decrease) in the number of requests.

This section considers scaling tests with the following parameterization..

- Scaling trigger: number of requests (Req);
- Scaling increments: +-1 VM;
- Traffic generation: Profile 2 (low increasing and decreasing rate).

6.3.1.1 Scale out

The Figure 123 depicts the time elapsed to scale out an ANDSFaaS instance on the DB tier, showing the Shard2 and Shard3 instances (Shard1 was already instantiated) and the 3 instantiation steps (OS, VM Ready and Service Ready).

As a result, it can be seen that the scaling out decision is taken later, considering that the threshold on the number of requests takes more time to be reached when compared with the Base Configuration (section 3.5.2.2.3.1). For example, here the first scaling out occurs around the 170s, while with Profile 1 it is around the 110s. A similar effect occurs for the second scaling out operation.

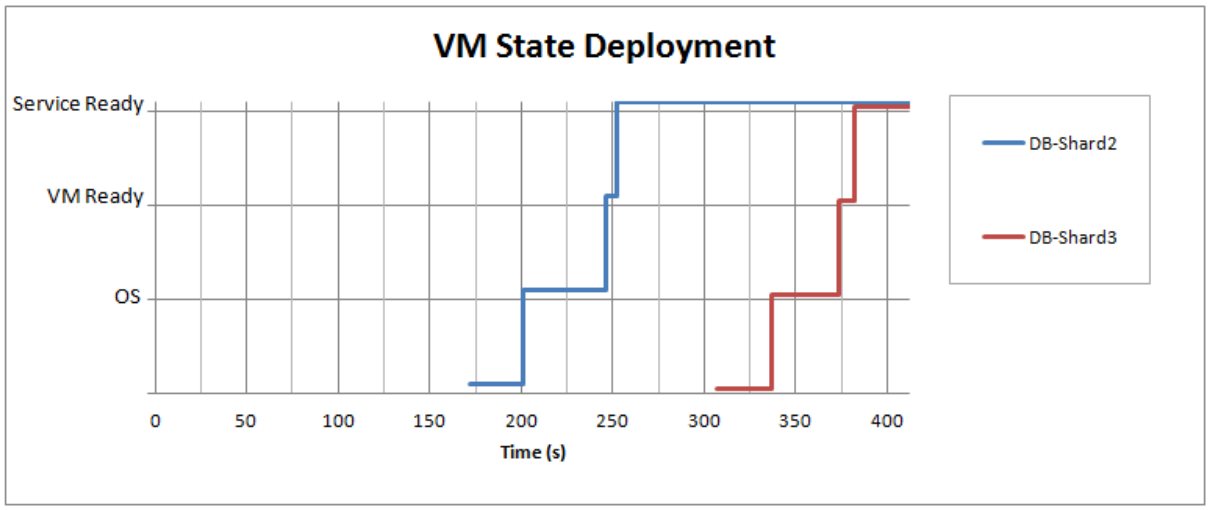

Figure 123 – ANDSFaaS: Scale DB, Trigger Req, Increment 1, Profile 2 (scale out)

6.3.1.2 Scale in

The Figure 124 depicts the time elapsed to scale in an ANDSFaaS instance on the DB tier, showing the Shard3 and Shard2 instances (Shard1 was already instantiated).

As a result, it can be seen that scaling in decision is taken later, considering that the threshold on the number of requests takes more time to be reached when compared with the Base Configuration (section 3.5.2.2.3.1). For example, here the first scaling in occurs around 965s, while with Profile 1 it is around 665s. The same occurs for the second scaling in operation.

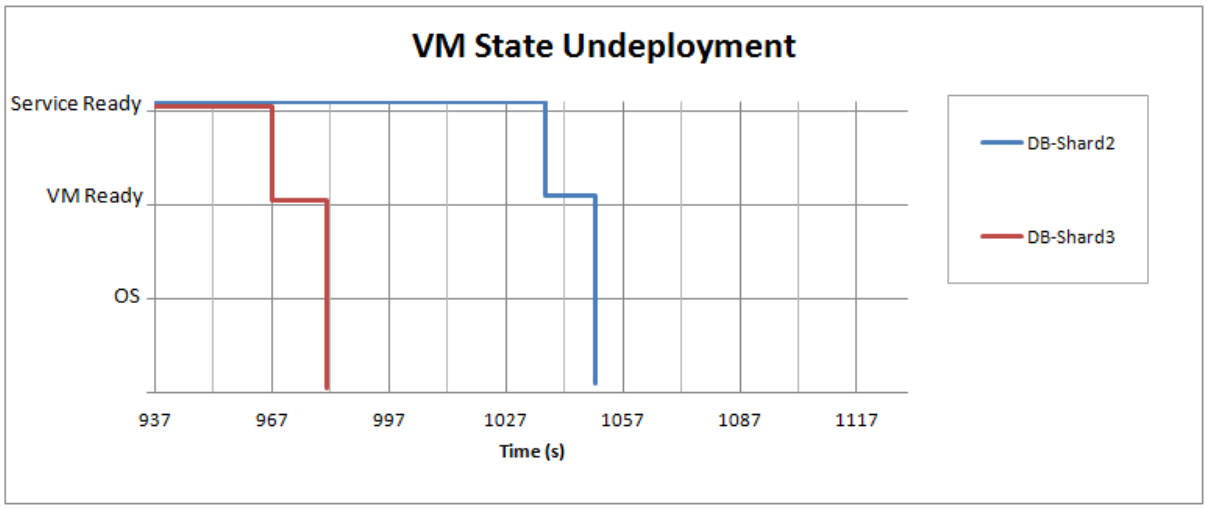

Figure 124 – ANDSFaaS: Scale DB, Trigger Req, Increment 1, Profile 2 (scale in)

6.3.1.3 Scale out and in

The Figure 125 shows the combination of scaling out and in an ANDSFaaS instance on the DB tier, as already shown above.

Compared to the usage of Profile 1 (high increasing rate), and as expected, the scaling operations are triggered later.

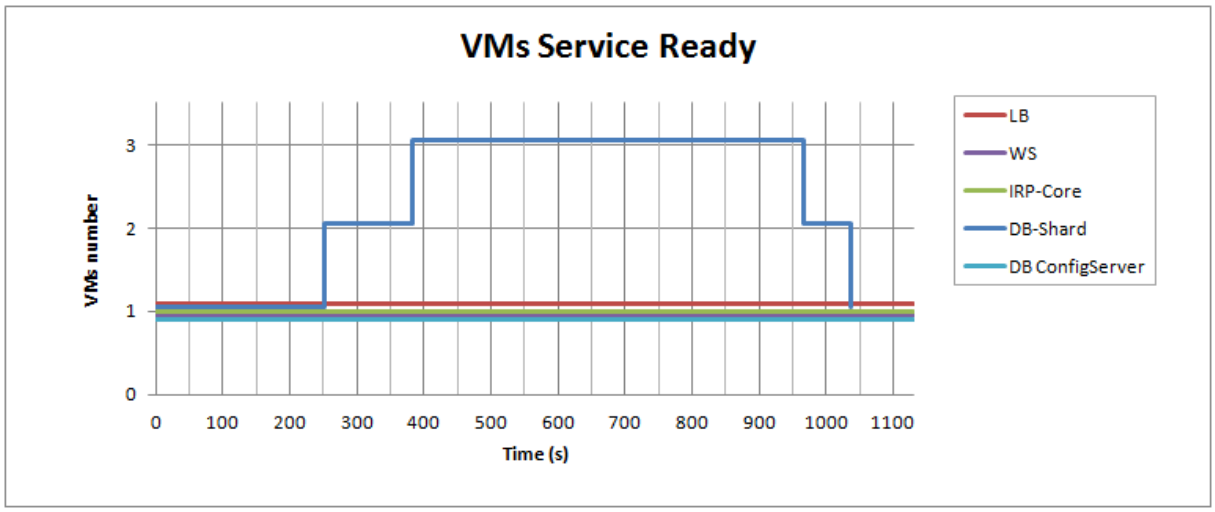

Figure 125 – ANDSFaaS: Scale DB, Trigger Req, Increment 1, Profile 2 (scale out and in)

6.3.1.4 Number of Requests

The Figure 126 depicts the service behaviour during ANDSF scaling in and out operations on the DB tier, showing the total number of requests and per Shard VM.

As a result, it can be seen that service loss is very low. The split of requests among the different Shards is not very balanced just as seen for Profile 1 (as for WS and IRP tiers). Shards launched earlier tend to receive more load than the others.

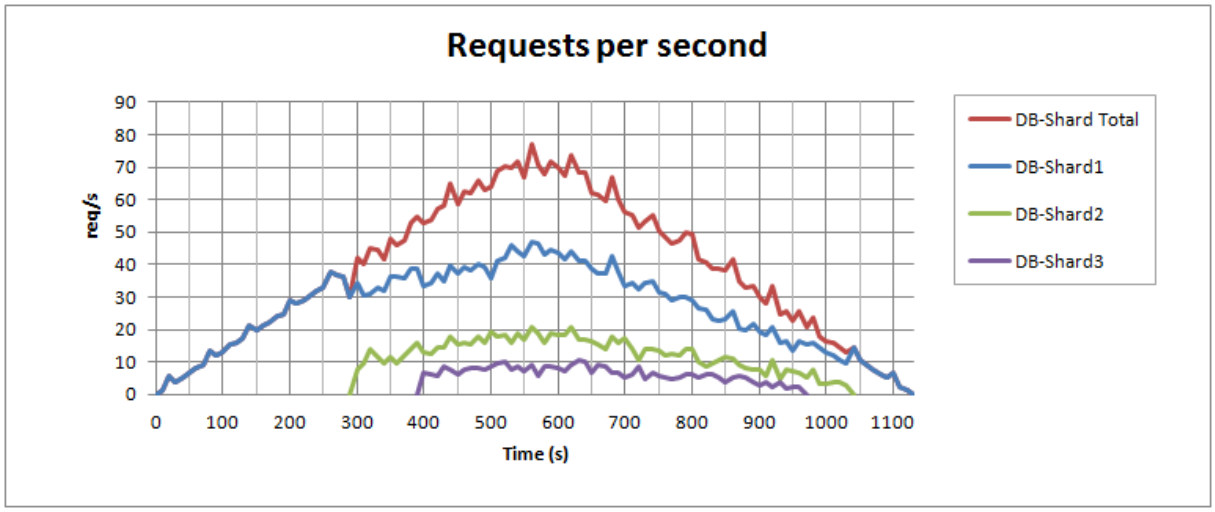

Figure 126 – ANDSFaaS: Scale DB, Trigger Req, Increment 1, Profile 2 (number of requests)

6.3.1.5 Impacts on Physical Machine

The Figures below show the impact of the ANDSFaaS scaling operations on the physical machine resources. In particular, the Figure 127, Figure 128 and Figure 129 show the impacts on CPU, RAM and I/O utilization, respectively, along the deployment and release periods.

As a result, it can be seen that the most relevant impact on the physical machine relates to I/O utilization, although quite less significant than for the deployment operations. The CPU and RAM are slightly affected. It can also be seen that scale-in operations require less resources than scale out.

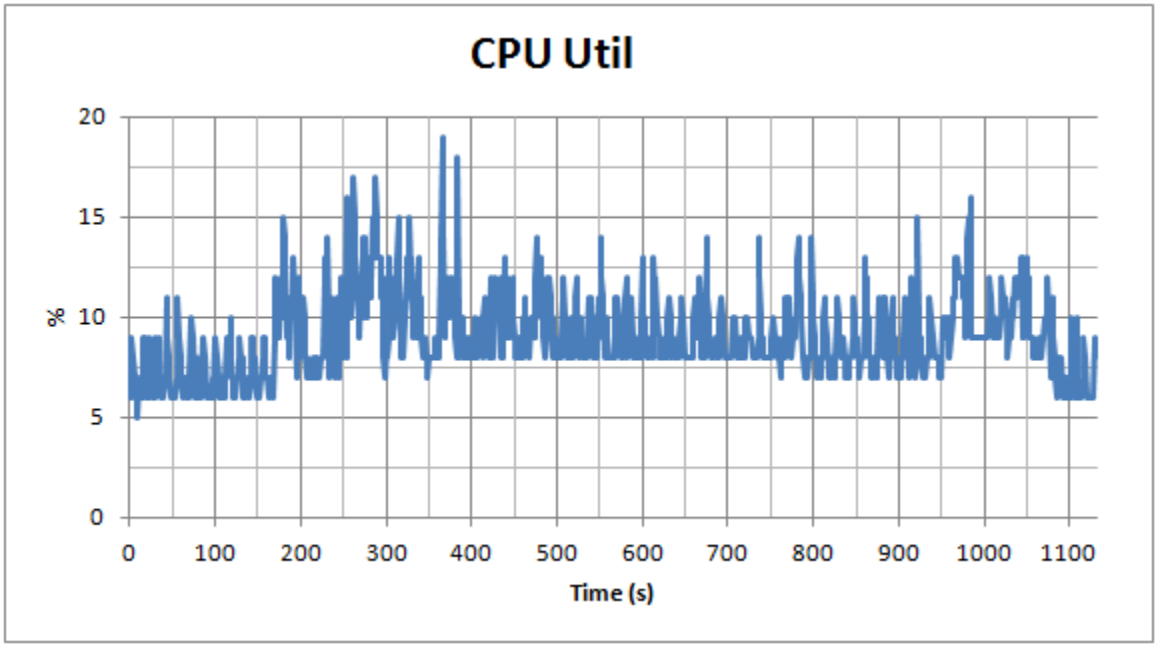

Figure 127 – ANDSFaaS: Scale DB, Trigger Req, Increment 1, Profile 2 (PM CPU utilization)

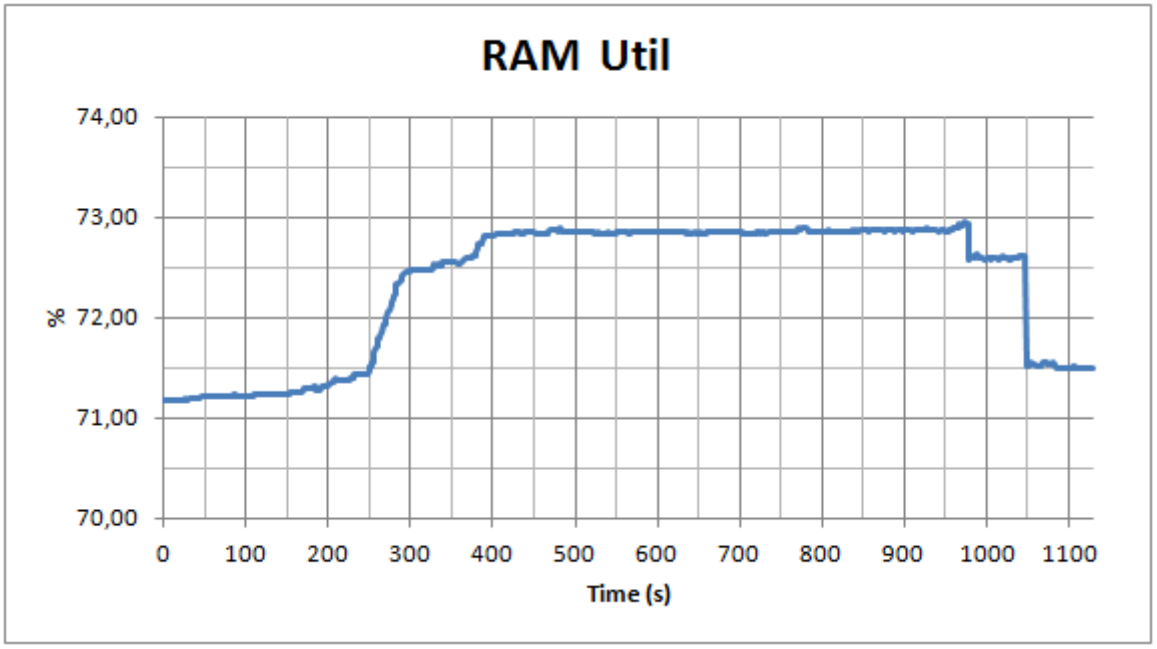

Figure 128 – ANDSFaaS: Scale DB, Trigger Req, Increment 1, Profile 2 (PM RAM utilization)

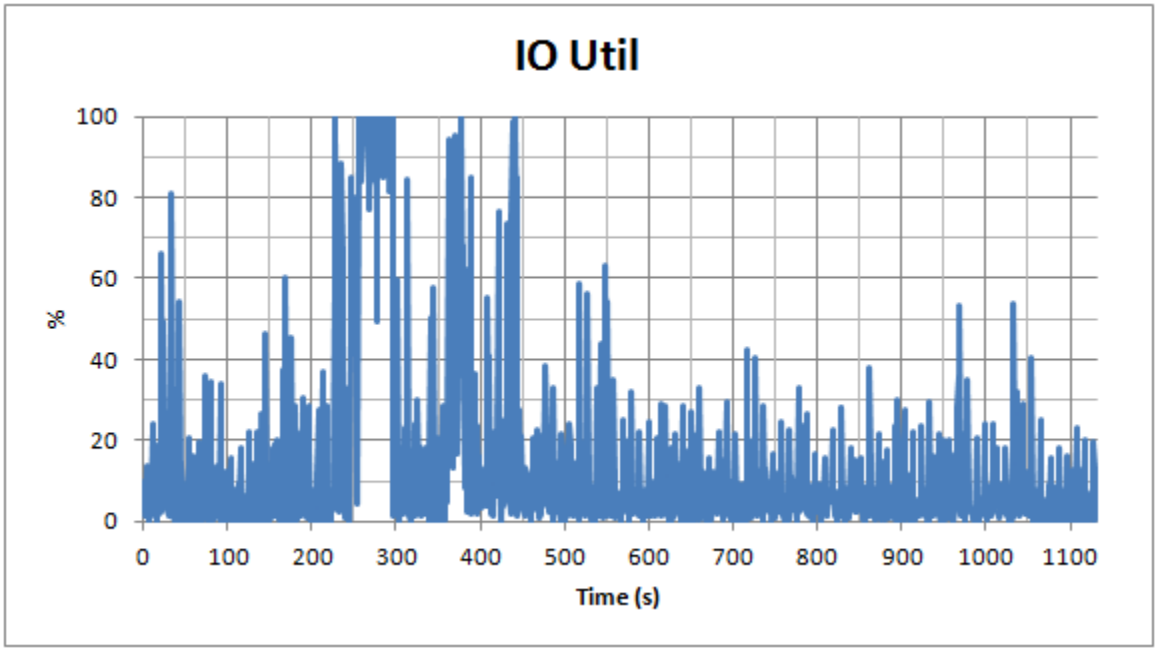

Figure 129 – ANDSFaaS: Scale DB, Trigger Req, Increment 1, Profile 2 (PM I/O utilization)

6.3.2 Comparison with Increment 2

This section intends to compare the results of the Base Configuration (section 3.5.2.2.3.1), when the increment of components (VMs) changes to a more aggressive increase (and decrease), i.e. each time a scaling out or in are performed, 2 VMs (Shards) are added or released, respectively.

This section considers scaling tests with the following parameterization..

- Scaling trigger: number of requests (Req);
- Traffic generation: Profile 1 (high increasing and decreasing rate).
- Scaling increments: +-2 VM;

6.3.2.1 Scale out

The Figure 130 depicts the time elapsed to scale out an ANDSFaaS instance on the DB tier, showing the Shard2 and Shard3 instances (Shard1 was already instantiated) and the 3 instantiation steps (OS, VM Ready and Service Ready).

As a result, it can be seen that the scaling out of 2 simultaneous components (VMs) take 155 seconds, 2 minutes and 35 seconds. This is a 30% more the time it takes to scale a single machine (110s). This is a larger difference than for IRP and WS layer, but anyway, this is still quicker than 1 by 1 scaling.

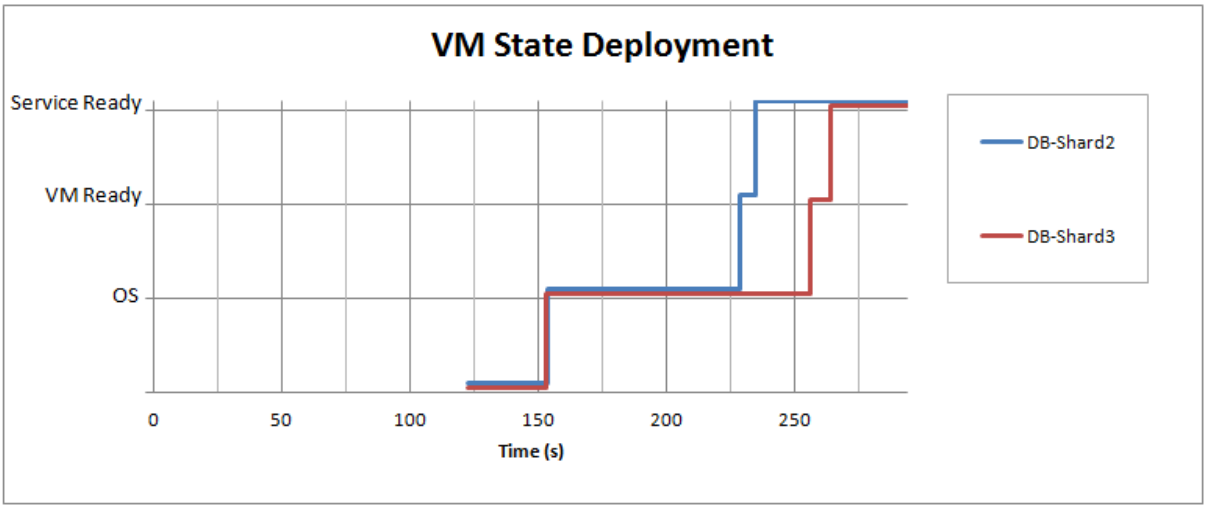

Figure 130 – ANDSFaaS: Scale DB, Trigger Req, Profile 1, Increment 2 (scale out)

6.3.2.2 Scale in

The Figure 131 depicts the time elapsed to scale in an ANDSFaaS instance on the DB tier, showing the Shard3 and Shard2 instances (Shard1 was already instantiated).

As a result, it can be seen that scaling in operations take 25 seconds, which is 60% more time than scaling in a single component (15s). Although it is more efficient on IRP and WS tier, this can also be used in DB when fast scaling in operations are required.

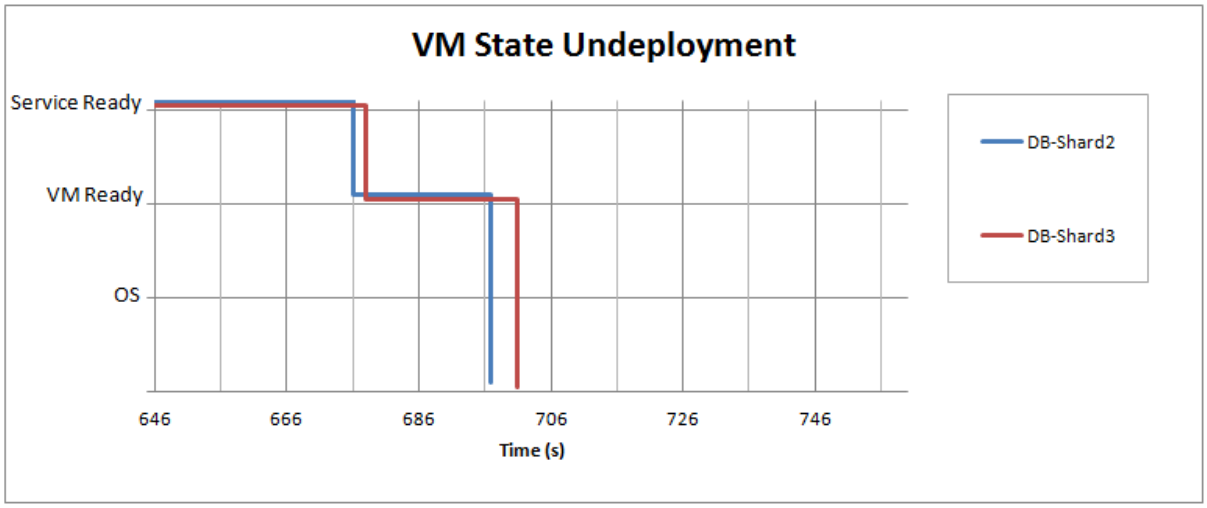

Figure 131 – ANDSFaaS: Scale DB, Trigger Req, Profile 1, Increment 2 (scale in)

6.3.2.3 Scale out and in

The Figure 132 shows the combination of scaling out and in of an ANDSFaaS instance on the DB tier, as already shown above.

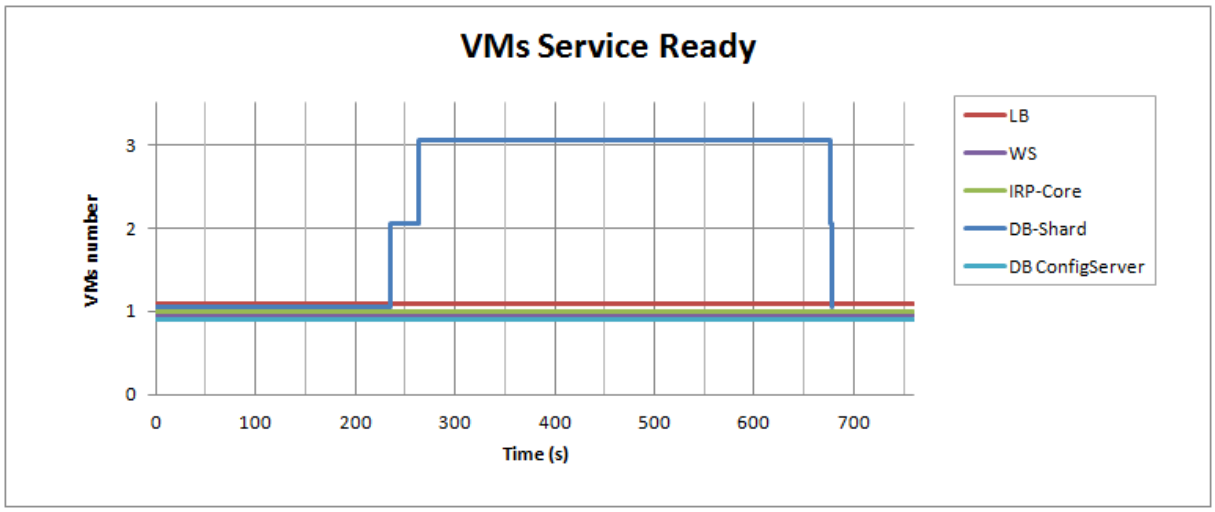

Figure 132 – ANDSFaaS: Scale DB, Trigger Req, Profile 1, Increment 2 (scale out and in)

6.3.2.4 Number of Requests

The Figure 133 depicts the service behaviour during ANDSF scaling in and out operations on the DB tier, showing the total number of requests and per Shard component (VM).

As a result, it can be seen that service loss is very low. The split of requests among the different Shards is not very balanced (as for WS and IRP tiers). Shards launched earlier tend to receive more load than the others.

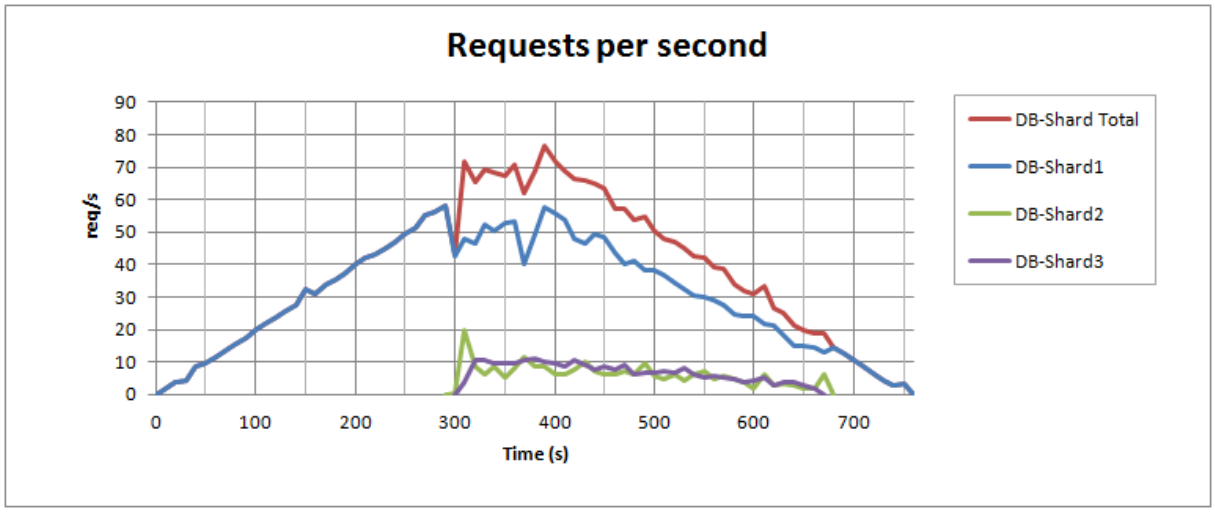

Figure 133 – ANDSFaaS: Scale DB, Trigger Req, Profile 1, Increment 2 (number of requests)

6.3.2.5 Impacts on Physical Machine

The Figures below show the impact of the ANDSFaaS scaling operations on the physical machine resources. In particular, the Figure 134, Figure 135 and Figure 136 show the impacts on CPU, RAM and I/O utilization, respectively, along the deployment and release periods.

As a result, it can be seen that the most relevant impact on the physical machine relates to I/O utilization, although quite less significant than for the deployment operations. The CPU and RAM are slightly affected. It can also be seen that scale-in operations require less resources than scale out.

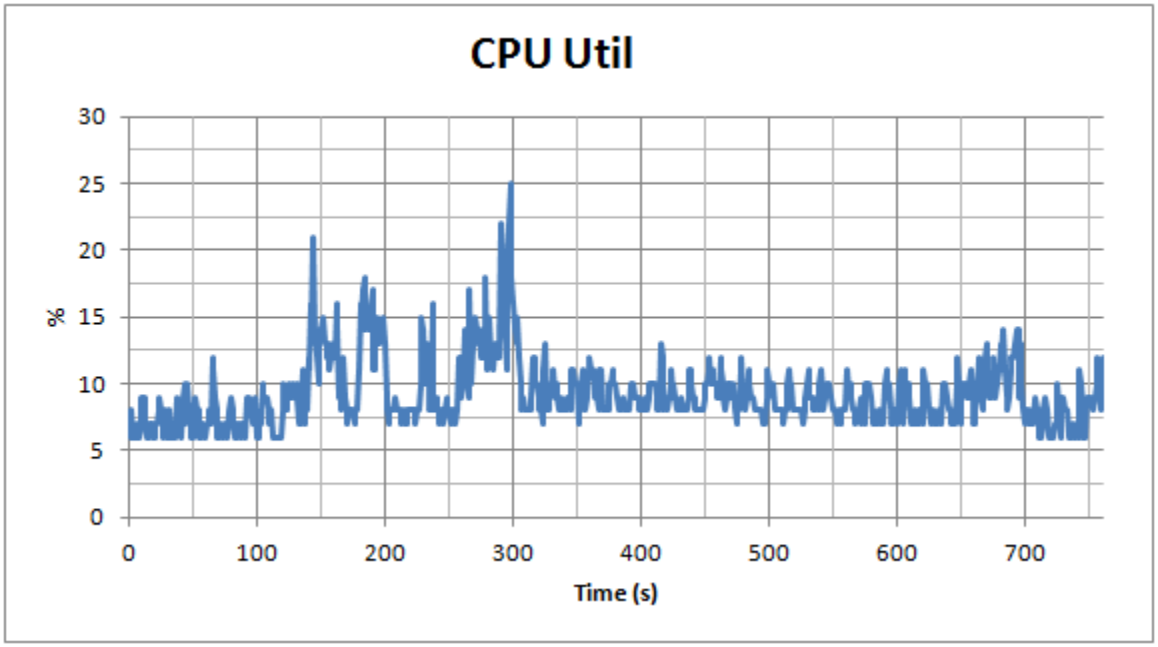

Figure 134 – ANDSFaaS: Scale DB, Trigger Req, Profile 1, Increment 2 (PM CPU utilization)

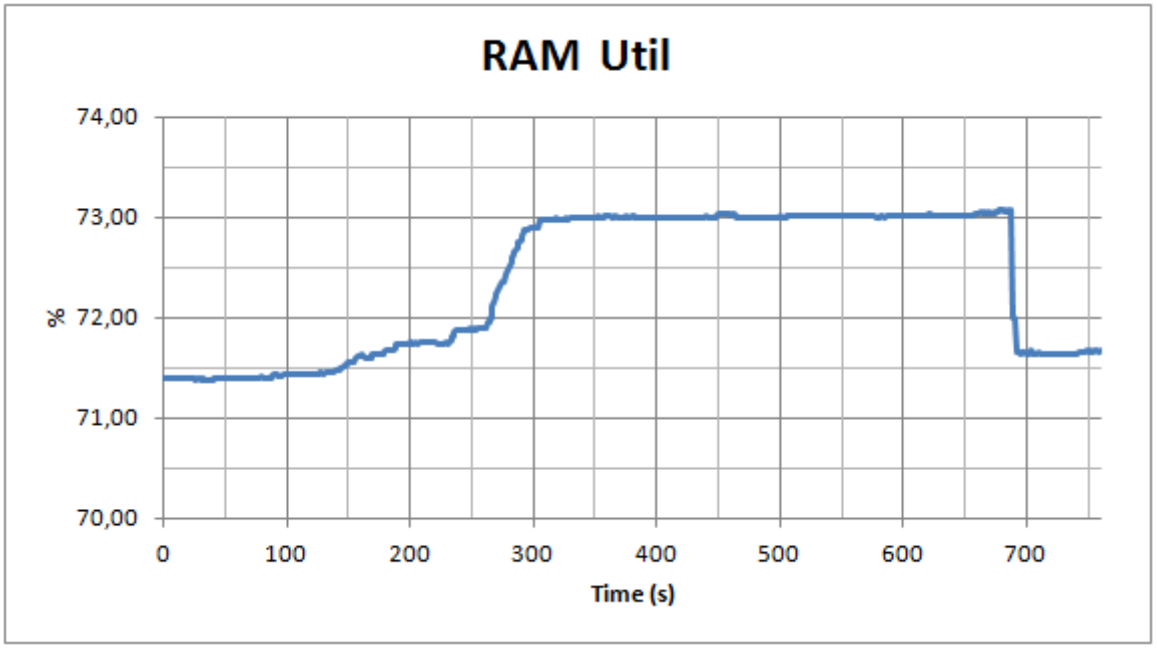

Figure 135 – ANDSFaaS: Scale DB, Trigger Req, Profile 1, Increment 2 (PM RAM utilization)

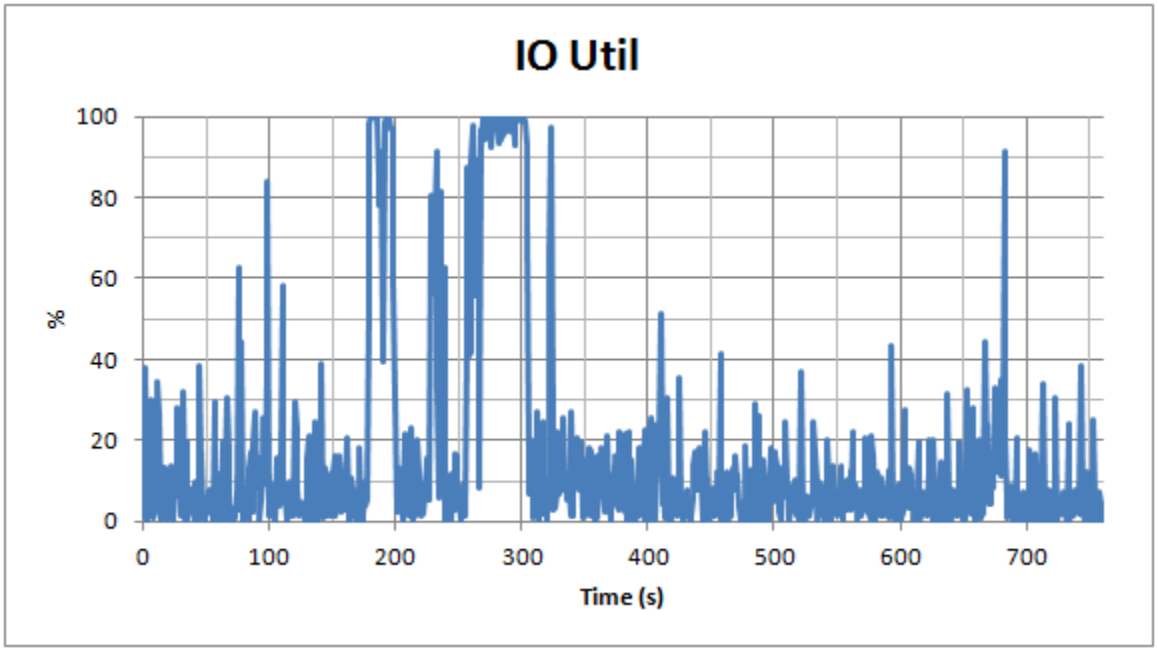

Figure 136 – ANDSFaaS: Scale DB, Trigger Req, Profile 1, Increment 2 (PM I/O utilization)

6.3.3 Comparison with Trigger CPU

This section intends to compare the results of the Base Configuration (section 3.5.2.2.3.1), when the monitoring event that triggers scaling operations is the CPU (instead the Number of Requests).

This section considers scaling tests with the following parameterization..

- Traffic generation: Profile 1 (medium increasing and decreasing rate).
- Scaling increments: +-1 VM;
- Scaling trigger: CPU utilization (CPU);

6.3.3.1 Scale out

The Figure 137 depicts the time elapsed to scale out an ANDSFaaS instance on the DB tier, showing the Shard2 and Shard3 instances (Shard1 was already instantiated) and the 3 instantiation steps (OS, VM Ready and Service Ready).

As a result, it can be seen that scaling out based on CPU threshold is also possible, although is harder to control (for testing purposes). It is also important to note that average CPU values must be used in order to avoid high variations and correspondent unstable scaling policies.

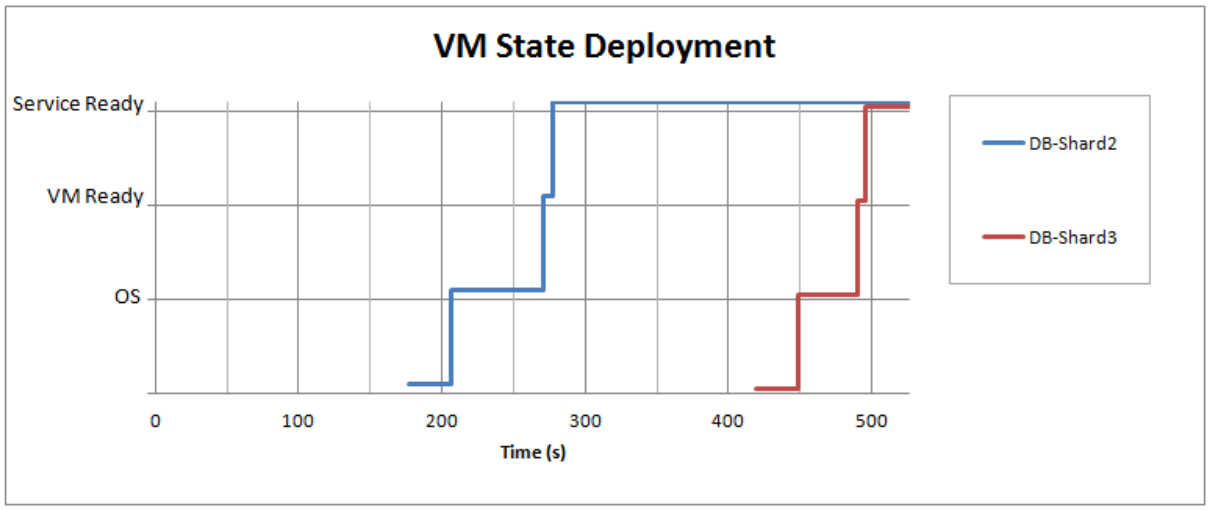

Figure 137 – ANDSFaaS: Scale DB, Increment 1, Profile 1, Trigger CPU (scale out)

6.3.3.2 Scale in

The Figure 138 depicts the time elapsed to scale in an ANDSFaaS instance on the DB tier, showing the Shard3 and Shard2 instances (Shard1 was already instantiated).

As a result, it can be seen that scaling out operations are also possible based on CPU.

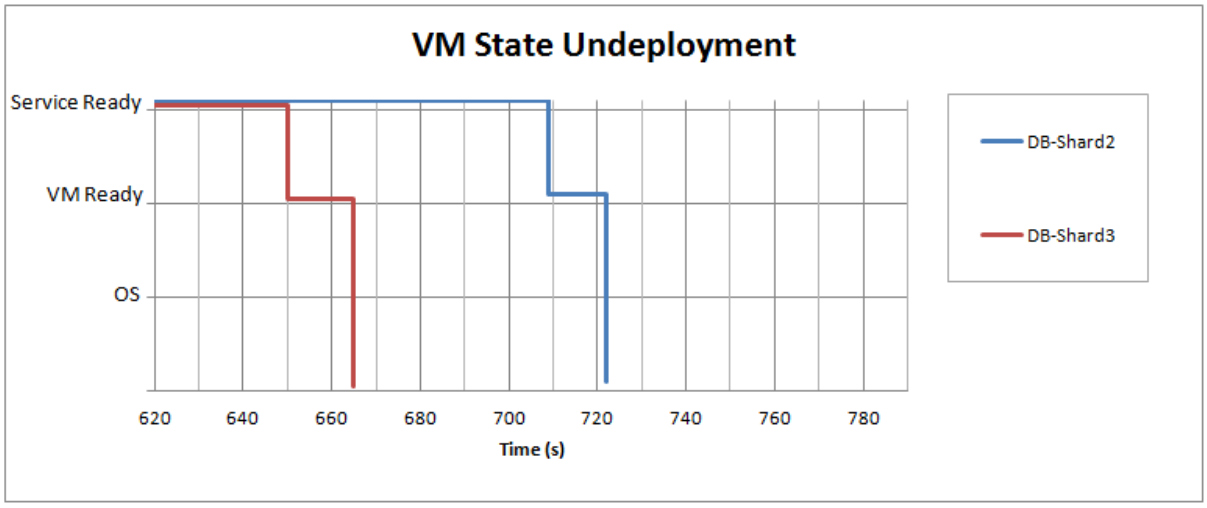

Figure 138 – ANDSFaaS: Scale DB, Increment 1, Profile 1, Trigger CPU (scale in)

6.3.3.3 Scale out and in

The Figure 139 shows the combination of scaling out and in an ANDSFaaS instance on the DB tier, as already shown above.

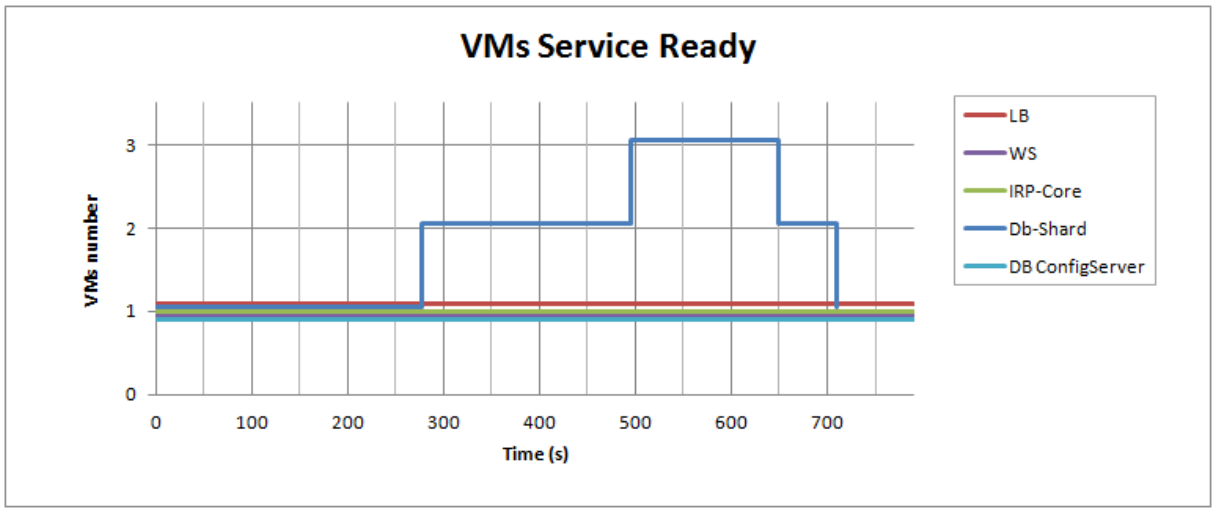

Figure 139 – ANDSFaaS: Scale DB, Increment 1, Profile 1, Trigger CPU (scale out and in)

6.3.3.4 CPU Utilization

The Figure 140 depicts the service behaviour during ANDSF scaling in and out operations on the DB tier, showing the total CPU utilization and per DB component (VM).

As a result, it can be seen that the CPU utilization thresholds are applied and the CPU load is quite balanced among the multiple components, although not so balanced as for IRP and WS tiers. However, it takes more time to balance than using the number of requests per second (Req). Note that, as DB tier is not CPU intensive, the CPU utilization is lower than for IRP and WS cases.

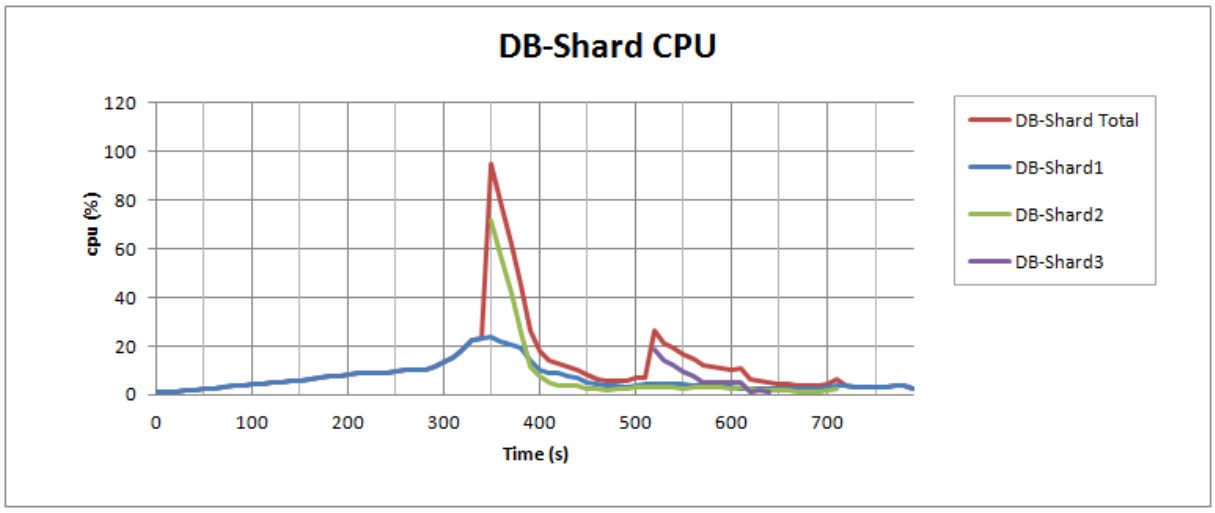

Figure 140 – ANDSFaaS: Scale DB, Increment 1, Profile 1, Trigger CPU (number of requests)

6.3.3.5 Impacts on Physical Machine

The Figures below show the impact of the ANDSFaaS scaling operations on the physical machine resources. In particular, the Figure 141, Figure 142 and Figure 143 show the impacts on CPU, RAM and I/O utilization along the deployment period.

As a result, it can be seen that the results are similar to the triggering based on the number of requests.

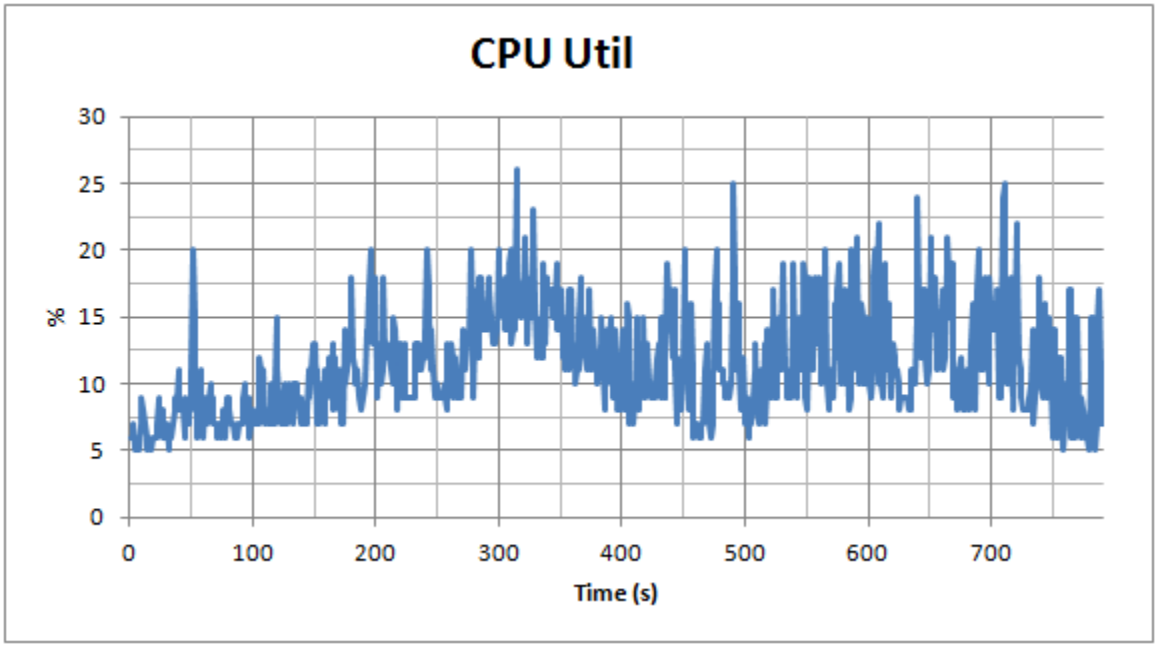

Figure 141 – ANDSFaaS: Scale DB, Profile 1, Increment 1, Trigger CPU (PM CPU utilization)

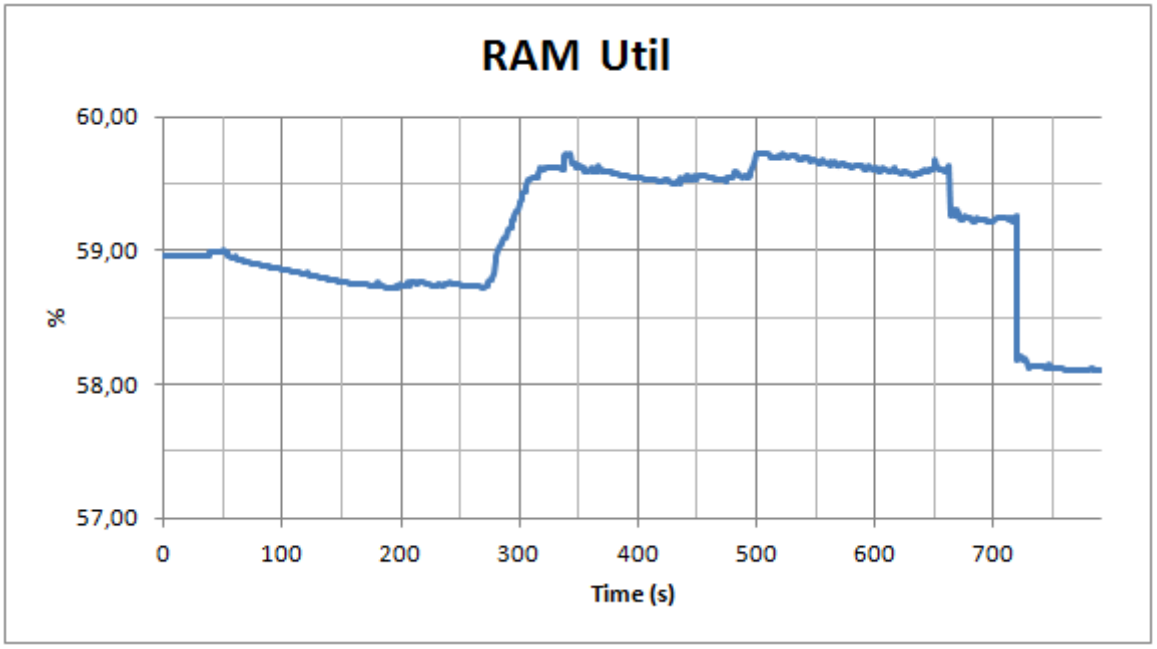

Figure 142 – ANDSFaaS: Scale DB, Profile 1, Increment 1, Trigger CPU (PM RAM utilization)

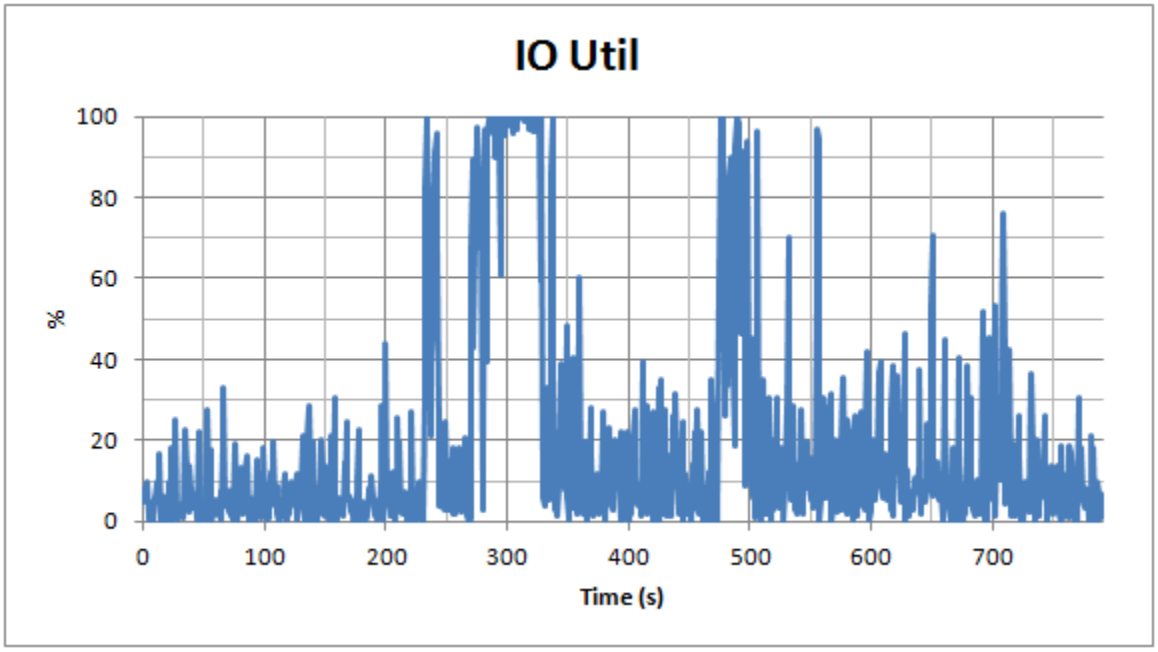

Figure 143 – ANDSFaaS: Scale DB, Profile 1, Increment 1, Trigger CPU (PM I/O utilization)

C Virtualized EPC – Additional Results

C.1 Low load assessment

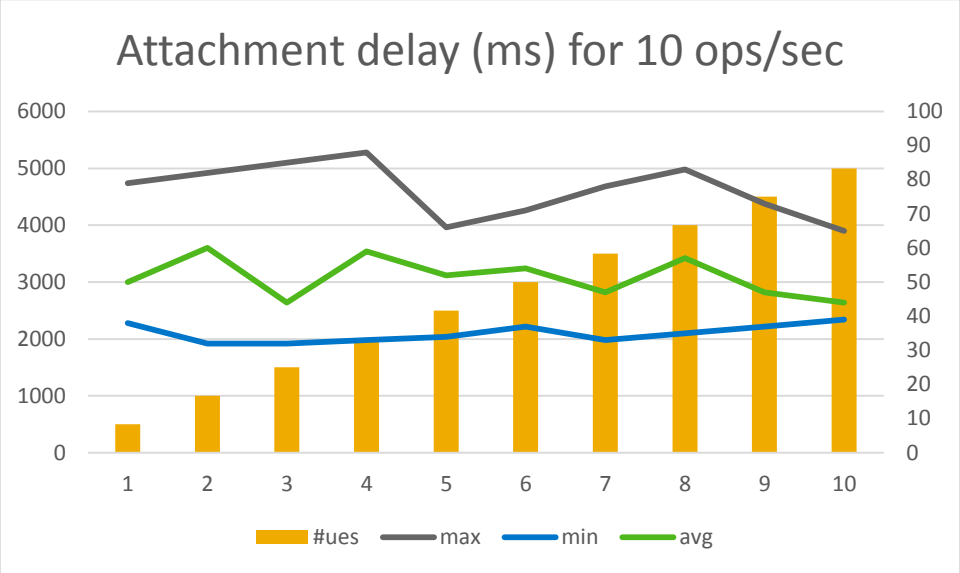

Figure 144 – Attachment delay (executed for 5000 UEs)

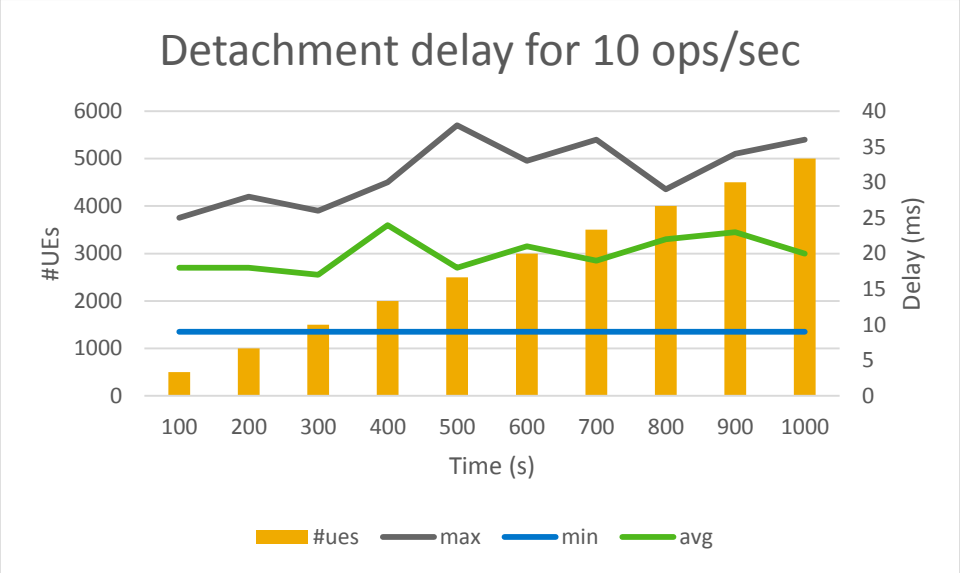

Figure 145 – Detachment delay for 10 ops/sec (executed for 5000 UEs)

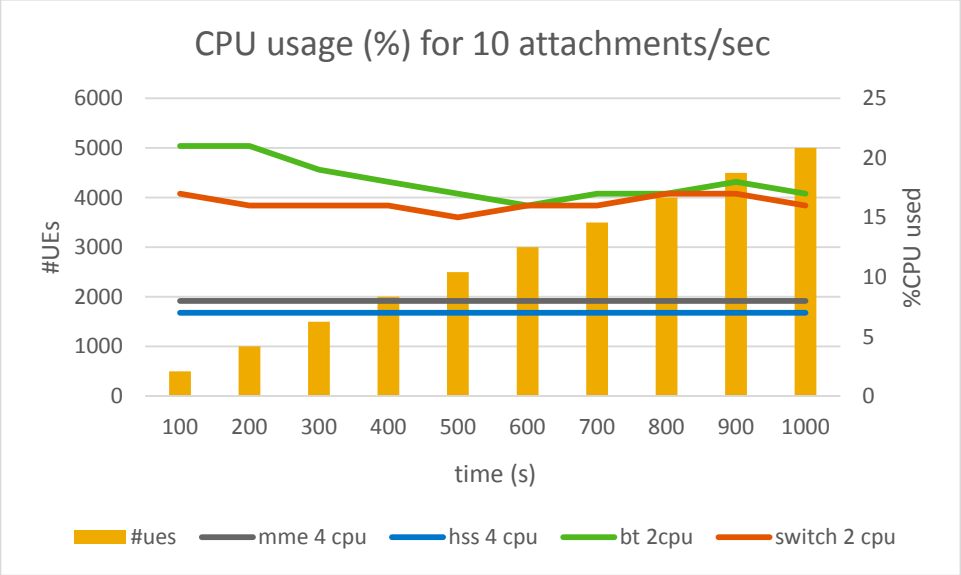

Figure 147 – CPU Usage (%) for 10 ops/sec (for 5000 users)

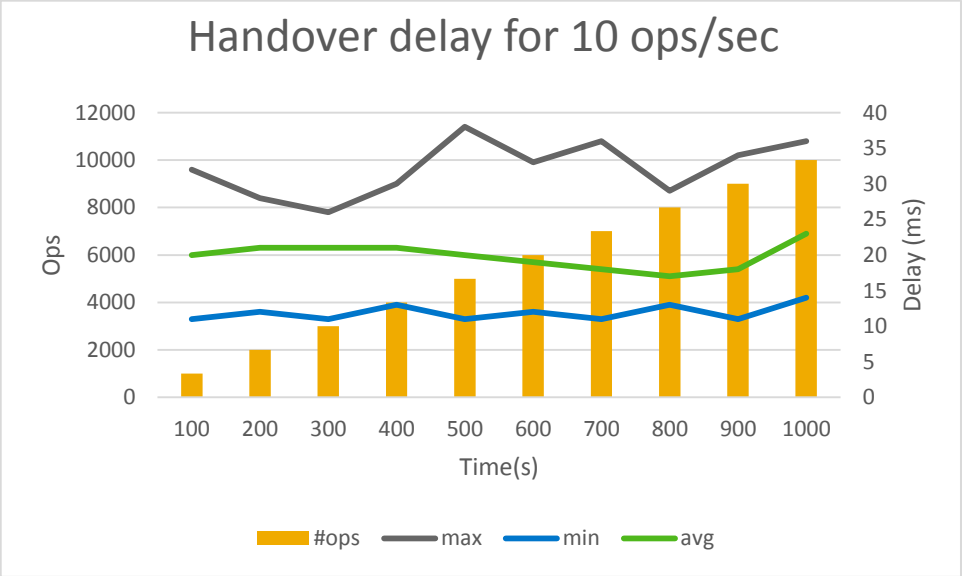

Figure 146 – Handover delay for 10 ops/sec (for 10000 UEs)

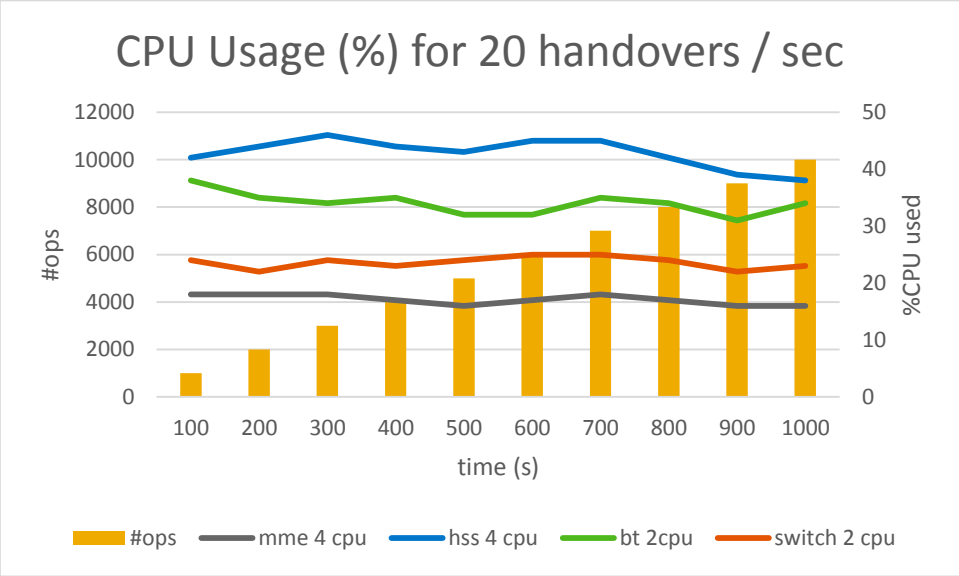

Figure 148 - CPU usage (%) for 20 handovers / sec