RELIABLE CAPACITY PROVISIONING AND ENHANCED REMEDIATION FOR DISTRIBUTED CLOUD APPLICATIONS



Deliverable D5.2: Final Data Acquisition and Analytics Models

Work Package 5. Data Collection, Visualization and Analysis

This project has received funding from the European Union's Horizon 2020 Research and Innovation Programme Under Grant Agreement Number 732667

Copyright ©2019 RECAP Project

Title:	Final Data Acquisition and Analytics Models
Author(s):	Rafael Garcia (IMDEA), Paolo Casari (IMDEA), Jörg Domaschka (UULM), Frank Griesinger (UULM), Linus Narvä(TIETO), Miguel Angel López (SATEC), Thang Le Duc (UMU), Peter Willis (BT)
Editor(s):	Paolo Casari (IMDEA), Jörg Domaschka (UULM)
Reviewed By:	Mark Leznik (UULM), Malika Bendechache (DCU)
Document Nature:	REPORT
Date:	30/04/2019
Dissemination Level:	Public
Status:	FINAL
Copyright:	Creative Commons Attribution No Derivative Licences (CC BY-ND 4.0)

Revision History

Version	Editor (s)	Date	Change
0.1	Rafael Garcia	15/06/2018	Identified changes wrt D5.1. Prepare Table of Contents.
0.2	Rafael Garcia, Paolo Casari	21/02/2019	First draft on methodology.
0.3	Rafael Garcia	09/04/2019	Draft on Auto Machine learning part. Exploratory data Analysis of Linknovate data.
0.4	Rafael Garcia	11/04/2019	Draft on Exploratory Data Analysis of Satec data. Draft on neural network architecture search.
0.5	Rafael Garcia	15/04/2019	Extended version of AutoML part
0.6	Jörg Domaschka	17/04/2019	Added section “Data Acquisition and Storage”
0.7	Rafael Garcia	22/04/2019	Version for internal review
0.8	Malika Bendechache	24/04/2019	Review of the entire deliverable
0.9	Thang Le Duc	25/04/2019	Added exploratory data analysis for BTvCDN use case
0.10	Thang Le Duc, Rafael Garcia	29/04/2019	Added exploratory data analysis for Tieto use Case. Addressed comments by reviewers.
1.0	Jörg Domaschka, Frank Griesinger	30/04/2019	Final updates and QA

Executive Summary

The former Deliverable D5.1 Initial Data Acquisition and Analytics Models provided a detailed description of the tasks planned in the context of RECAP work package WP5, from the point of view of data processing and statistical models. The document also provided an architectural design of the different elements that composed the WP5 solution for data retrieval, management and storage. The document covered both offline (trace-based) and online (monitoring-based) data sources.

Deliverable D5.2 extends and complements the information included in Deliverable D5.1, but also includes many new results that have been produced in the meantime. These new results include:

i) The insight gathered during the exploratory data analysis of all use cases.
ii) A detailed methodology of how to proceed for the analysis of new scenarios not included in the RECAP project.
iii) A description of the research activities performed in the area of auto-machine learning.
iv) An application of the new research tools to the analysis of RECAP Use Cases.

D5.2 also updates the information of D5.1 where applicable, for example, about the progress and status of the different developments, or by delivering an updated version of the metrics being collected from the Use Cases.

The collection of deliverables D5.1 and D5.2 provide all the required information, from a data collection and analysis point of view, and lay the grounds for ongoing taks in work packages WP6 (workload and application modelling), WP7 (simulation) and WP8 (application placement and infrastructure optimisation).

Table of Contents

Revision History 2
Executive Summary 3
Table of Contents 4
Table of Figures 6
List of Tables 7
List of Abbreviations 8
1 Introduction 9
1.1 Changes with respect to Deliverable D5.1 9
1.2 Global Architecture 10
2 Data Acquisition and Storage 12
2.1 Monitoring Infrastructure 12
2.1.1 Data Sinks 12
2.1.2 Probes 13
2.1.3 Dispatching 13
2.2 Data Structure for Storage 14
2.3 Implementation 14
2.3.1 Implementation Technology 14
2.3.2 Data Sets and Data Collection 15
2.3.3 Use Case Coverage 15
3 Data Analytics 16
3.1 Exploratory Data Analysis of Use Cases 16
3.1.1 Use Case A: Infrastructure and Network Management 16
3.1.2 Use Case B: Data Analytics Engine 18
3.1.3 Use Case C: Edge/Fog Computing for Smart Cities 22
3.1.4 Use Case D1: vCDN 23
3.2 A Methodology for the Analysis of New Datasets 27
3.2.1 Problem Definition and Data Assembling 27
3.2.2 Metric for the Evaluation of the Results 28
3.2.3 Data Curation and Enhancement 29
3.2.4 Model Development 29
3.2.5 Regularization and Hyperparameter Selection 30
4 Automated Machine Learning Algorithms 31
4.1 The Minimum Nescience Principle 32
4.2 Decision Trees 34
4.3 Auto Machine Learning with Neural Networks 37

4.4 Auto-Machine Learning applied to Use Case D.1 38
4.4.1 Feature Selection 38
4.4.2 Neural Networks 39
5 Conclusion 41
References 42
6 Annex 44
6.1 Monitoring Metrics per Layer 44
6.1.1 Metrics on the Physical Layer 44
6.1.2 Metrics on the Virtual Layer 45
6.1.3 Metrics on the Container Layer 45
6.1.4 Metric Attributes: Tagging 46
6.2 Use Case Coverage 47
6.2.1 System Metrics per Use Case 47
6.2.2 Application Metrics per Use Case 49

Table of Figures

Figure 1 WP5 Conceptual Architecture..... 11

Figure 2. Histogram of the distribution of the Linknovate search engine's responses..... 20

Figure 3. Histogram of the distribution of the Linknovate search engine's response time..... 20

Figure 4. Scatterplot of the response size and time for the Linknovate search engine. 21

Figure 5. Time series of the Linknovate search engine's workload (data aggregated over windows of 30 minutes)..... 21

Figure 6. Histogram of the distribution of the number of cars in each city zone for the SATEC Use Case. 23

Figure 7.Evolution of the number of cars in each region over one day's span in the SATEC Use Case. 23

Figure 8 Serving Cache Traffic: Original Traffic Data 24

Figure 9 Serving Cache Traffic: Hourly Resampled Traffic Data 24

Figure 10 Service Traffic of a Cache: Daily Resampled Traffic Data 25

Figure 11 Decomposition of serving traffic of a cache..... 25

Figure 12 Decomposition of received traffic at a cache. 26

Figure 13 Seasonal compositions of cache traffic. 26

Figure 14. A summary of the main steps of the methodology for exploratory data analysis of new datasets..... 27

Figure 15. Synthetic dataset employed to show the capabilities of the nescience-based AutoML algorithm for decision tree construction. 35

Figure 16. Accuracy achieved by our nescience-based decision tree construction algorithm as compared to the accuracy achieved by the standard CART algorithm..... 36

Figure 17. Number of tree nodes obtained by our nescience-based decision tree construction algorithm as compared to the number of nodes in models output by CART. Our algorithm maintains a stable number of tree nodes, suggesting its better ability to interpret the essence of the Gaussian blob dataset, avoiding CART's overoptimization. 36

Figure 18 First snapshot of the Automated Neural Network Creation 38

Figure 19 Second Snapshot of the Automated Neural Network Creation 38

Figure 20. Third Snapshot of the Automated Neural Network Creation..... 38

List of Tables

Table 1 RECAP data sinks from Monitoring Sub system 12

Table 2 Overview of RECAP-provided monitoring functionality 13

Table 3 New metrics exposed by the RECAP monitoring sub system 14

Table 4 Overview of baseline technology for monitoring components 15

Table 5 Data Set Collected in RECAP 15

Table 6. Statistics of the response codes returned by the Linknovate search engine. 19

Table 7. The top 10 IP addresses directing the largest number of requests to the Linknovate search engine. 19

Table 8: Descriptive statistics of the number of cars in each city zone for the SATEC Use Case.. 22

Table 9 Relevant Features According to Nescience 39

Table 10 Relevant Features According to CART 39

Table 11 Features Used in the Neural Network 40

Table 12 Metrics used on physical layer 44

Table 13 Metrics used on virtual layer 45

Table 14 Supported metric attributes per metric set 46

List of Abbreviations

Abbreviation	Full Name
ARIMA	Auto Regressive Integrated Moving Average
API	Application Programming Interface
CDN	Content Delivery Network
EDA	Exploratory Data Analysis
ETL	Extraction, Transformation and Loading
GARCH	Generalized Auto Regressive Conditional Heteroskedasticity
IoT	Internet of Things
k-NN	K Nearest Neighbours
LAN	Local Area Network
LSTM	Long Short Term Memory neural network
MNP	Minimum Nescience Principle
MPLS	Multi-Protocol Label Switching
NFV	Network Function Virtualization
NN	Neural Network
PCA	Principal Components Analysis
PD	Packet Delivery
PE	Provider Edge
RNN	Recurrent Neural Network
RTD	Round Trip Delivery
SLA	Service Level Agreement
SVM	Support Vector Machines
VPN	Virtual Private Network

1 Introduction

The work package *WP5 Data Collection, Visualization and Analysis* of the RECAP project has been designed to deliver the tools required for the acquisition, processing and visualisation of the data needed by other parts of the RECAP project, in particular in WP6 (workload modelling), WP7 (simulation), and WP8 (application modelling). The same tools will be also required to understand and develop new models of the behaviour of systems and applications. More concretely, the objectives of WP5 are the following:

- To provide an infrastructure for the acquisition and processing of data (both from applications and from systems);
- The development of mathematical models to identify relevant features and forecast future values;
- To provide the tools required for the easy retrieval and visualisation of data, cf. (D5.4).
- To provide an open dataset with traces collected from real infrastructure, so that the same data can be used for further research, cf. (D5.3).

In order to fulfil these goals, the work package has been organized into four tasks: T5.1 is devoted to the acquisition of data, its transport, transformation, and storage; T5.2 is intended for the analysis of the available data and the development of new models based on machine learning techniques; T5.3 deals with the generation of simulated artificial workloads; and T5.4 is responsible for the visualisation and the provision of APIs for data retrieval.

Deliverable D5.2, being the second deliverable of WP5, extends and complements the information included in Deliverable D5.1 (Casari and Domaschka 2018), but also includes many new results that were not available at that time. These new results include:

i) The insights gathered during the exploratory data analysis of RECAP Use Cases.
ii) A detailed methodology of how to proceed for the analysis of new scenarios outside the RECAP project.
iii) A description of the research activities performed in the area of auto-machine learning.
iv) An application of the new research tools to the analysis of RECAP Use Cases.

This document also updates some of the information included in (Casari and Domaschka 2018), for example, about the progress and status of the different developments, or an updated version of metrics being collected from Use Cases.

- This document focuses on the activities performed in the scope of Task 5.1 (Data Acquisition, Assembly and Publication) and Task 5.2 (Feature Engineering and Analytics). The activities related to Task 5.3 (Artificial Workload Generation) and Task 5.4 (Visualisation and APIs) will be described in detail in their own deliverables (D5.4) and (D5.3) respectively.

1.1 Changes with respect to Deliverable D5.1

Deliverable D5.2 extend and complements Deliverable D5.1 - Initial data acquisition and analytics models (Casari and Domaschka 2018). In order to make the deliverable self-contained while at the same time avoiding content duplication as much as possible, some parts already presented in Deliverable 5.1 have been removed or summarized. The following list provides an overview of the main changes with respect to Deliverable 5.1:

- Former Chapter 2. *Use Case Description* has been removed. Use Cases have not significantly changed with respect to the description included in (Casari and Domaschka 2018), therefore we do not repeat the same information here, except for a very short summary at the beginning of each data analysis section. We refer the reader to (Casari and Domaschka 2018) for a more detailed description of the Use Cases from the point of view of data acquisition and modelling.
- Former Chapter 3 (now Chapter 2) has been updated. The changes introduced in the monitoring infrastructure are described, and the list of collected metrics is updated with the latest additions and removals. A new section describes the volume of data collected so far, and the expected volume of data to be collected at the end of the project.
- Chapter 4 (now Chapter 3) has been rewritten. Chapter 4 in (Casari and Domaschka 2018) described a collection of tools and techniques for the exploratory analysis of statistical data, and some preliminary results of the applicability of these techniques to the RECAP datasets. In this new version of the chapter, we include the descriptive analysis performed over the RECAP datasets. Also, a new section is included where we propose a methodology for the analysis of new datasets, so that the results of RECAP can be applied to new Use Cases.
- Former Chapter 5 *Beyond Acquisition and Analytics* is not included. This chapter was intended as a short introduction to the topics "artificial workload generation" and "visualization", with the aim to clarify the scope of Task 5.3 Artificial Workload Generation and Task 5.4 Visualization and APIs, and to describe the expected results of these tasks. The content of the chapter is still relevant, and we refer to (Casari and Domaschka 2018) for more information. The work performed in Task 5.3 and Task 5.4 will be fully described in their own deliverables (D5.3) and (D5.4).
- A new Chapter 4 describes the research activities performed in WP5. In particular, the minimum nescience principle (see Section 4.1), a method to build optimal machine learning models is presented, and its applicability to building optimal decision trees and optimal neural networks are described. The applicability of the new techniques to the RECAP project is described as well.

1.2 Global Architecture

Figure 1 depicts the global architecture of the WP5 Data Acquisition and Analytics Models as proposed in (Domaschka, Griesinger and Leznik, et al. 2018). The conceptual architecture depicts the tasks of WP5 as groups of methods and technologies (colored rectangles). Arrows symbolize the interconnection of tasks within WP5 as well as outside the work package, respectively, the Simulation and Optimisation efforts (WP6, WP7, WP8). A rectangle within a group depicts a conceptual element needed to perform the specific tasks. Main technologies involved are hightlighted by the respective icon of the product. The blue group is the "Data Acquisition, Assembly and Publication" task, grouped together with the "Visualisation and Dev API" task as logical unit, due to the strong technical and methodigical alignment, i.e. the monitoring paradigms we apply. The conceptual elements reach from collecting monitoring data from the use cases, transform and store it in the RECAP infrastructure, and finally publish it through adequate interfaces. In this way, it is connected to the Simulation in order to provide data in a well-known specified manner. Further, it is connected with the "Artificial Workload Generation" in respect to storage and interface development. "Feature Engineering and Analytics" is a mainly connected via the API, and further serves the Prediction Models to the Optimisation components.

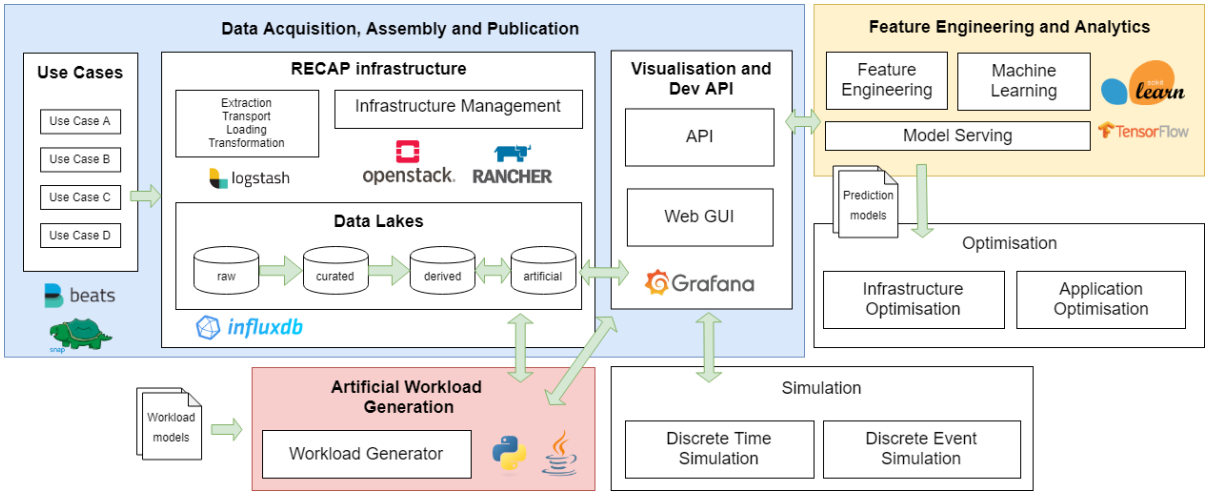

Figure 1 WP5 Conceptual Architecture

2 Data Acquisition and Storage

Data acquisition is the basis for all data analytics to be performed in RECAP as well as a major building block for all optimisation actions triggered on both the application and the infrastructure level (Domaschka, Griesinger and Leznik, et al. 2018). RECAP Deliverable D5.1 described the initial approach to data acquisition and storage (Casari and Domaschka 2018) including a database schema for both file based storage and time series databases. This section provides a concise summary of the earlier information and adds advances where applicable.

2.1 Monitoring Infrastructure

The monitoring infrastructure consists of three major components: (i) Probes that get monitoring data from any of the monitoring domains. (ii) Data sinks that represent integration points for the RECAP system. (iii) The data dispatcher that functions as the central integration point between the various probing and monitoring systems and the data sinks.

2.1.1 Data Sinks

Much of the modelling and optimisation done in RECAP is realised through the RECAP data repository (cf. Section 1.2). Yet, technically, this repository cannot be realised as a single entity, as it has to satisfy different, contradicting requirements: The data analytics and machine learning functionality (cf. (Domaschka, Griesinger and Leznik, et al. 2018)) requires access to large chunks of (csv formatted) data; the visualisation components (cf. (D5.4) requires the capability to flexibly query for data upon a user request usually spanning smaller time spans than used for analytics; RECAP components, particularly application optimisers (Le Duc and Östberg 2018) and infrastructure optimiser (Loomba, Quinn und Kennedy, et al. 2018), require access to a live stream of data. For each of these application domains, RECAP provides a dedicated data sink as discussed in (Casari and Domaschka 2018) and summarised in Table 1.

Table 1 RECAP data sinks from Monitoring Sub system

sink name	description	main application domain
data lake	A durable storage able to persist large amounts of monitoring data for longer time in a compact representation.	data analytics and machine learning tasks
database	Through an underlying indexed search engine, it is able to support live queries of current and past data.	Visualisation, but also optimisation algorithms
stream	relays a configurable sub set of live monitoring data to other parts of the RECAP infrastructure.	optimisation engines, error detection

Deployment Structure and Data Management

RECAP is intended for use in geo-distributed scenarios with multiple data centres (called sites) connected through wide-area network technology. As detailed in (Domaschka, Griesinger and Leznik, et al. 2018) and (Domaschka, Narvä, et al. 2019), each of the sites provides each of the sinks as integration point.

In order to ease the management of collected data, RECAP supports a time-to-live for all data collected in the database sink and the data lake sinks. In particular edge sites have limited hardware capabilities and hence reduced capabilities for long term data storage. Once the time-to-live has expired, the data is purged from the site.

For long-term storage of data in the data lake sink, RECAP supports the definition of a global data lake and a periodic synchronization from local to global data lakes.

2.1.2 Probes

Probes are responsible for getting telemetry data out of monitored systems. For each metric different probes may be needed and each probe may emit data in a different format and at different time intervals. RECAP requires that besides metric timestamp and metric value, an emitted data item contains metric properties for identifying the source and scope of the data point (cf. Section 2.2).

Table 2 Overview of RECAP-provided monitoring functionality

sink name	description	background technology
snap-esxi	Using the interface provided by Vmware's ESXi and vSphere tools, collects metrics about the virtual machines and their hypervisors	integrated in SNAP
vm-list	Using the libvirt command line tool, collectes the list of currently deployed VMs on a host.	standalone, collected through filebeat
Impi	Using the IPMI interface, collects power metrics from the servce power supply units (PSUs).	standalone, collected through filebeat
flowexporter	Using sflow, collects flow data capturing information flow between virtual machines and hence between services. Also used to extract statistics on network devices.	standalone, collected through filebeat

Existing infrastructures and applications usually already come with probes to capture the state and health of the system. Hence, in order to support a seamless integration with existing systems, RECAP does not impose a dedicated monitoring system. This eases the uptake of RECAP technology for both RECAP infrastructure providers as well as operators of RECAP applications. In contrast, the RECAP dispatcher (cf. Section 2.1.3) functions as an integration point that ensures data normalisation and proper naming of metrics.

Despite being probe-agnostic RECAP ships with pre-defined probes that are being used for monitoring the Use Cases and the testbeds (Domaschka, Narvä, et al. 2019). Table 2 summarises custom implementations of probes for collecting dedicated metrics used for the second iteration of testbeds, cf. (D4.3).

2.1.3 Dispatching

RECAP's data dispatcher is the central integration point of monitoring data per site. It receives all data from the probes on any of the layers of the infrastructure, normalises the representation per layer (cf. Section 2.2), and emits the data to the data sinks when needed.

The normalisation step is dependent on the probes. In particular, an individual transformation is needed per metric and per type of probe. The dispatcher adds additional properties to the collected data; in particular, information on the site and the layer of the collected data.

It is worth noting that from a logical point of view, the dispatcher is a singleton element. Due to its stateless-ness, however, it can be scaled out of the box so that it does not constitute a bottleneck in the infrastructure.

2.2 Data Structure for Storage

In Deliverable D5.1, we defined a normalised data schema applicable for all three types of data sinks (Casari and Domaschka 2018). This schema has been proven as stable and beneficial and has not been changed since then. Yet, it has been enhanced to capture additional properties summarised in Table 3 as discussed below.

The full and final list of all supported metrics on any of the layers except the application layer is available in the annex of this document (cf. Section 6.1). Beyond adding the metrics from Table 3, no changes were made compared to (Casari and Domaschka 2018). The list of application-specific metrics per Use Case is subject to Section 2.3.3.

Table 3 New metrics exposed by the RECAP monitoring sub system

Metricset	layer	Metrics	Description
host.ipmi	physical	power_status	the current state of the host (on/off)
		power_consumption	the current power consumption in watts
host.flow	physical	bytes_send	number of bytes sent for a dedicated flow
		bytes_received	number of bytes received for a dedicated flow
switch.flow	physical	*as for host.flow*	
switch.network	physical	in_bytes in_packets	Number of bytes/packets received
		in_dropped	The number of dropped incoming packets.
		in_errors	The number of errors while receiving.
		out_bytes out_packets	Number of bytes/packets sent.
		out_dropped	The number of dropped outgoing packets.
		out_errors	The number of errors when sending.
vm.flow	virtual	*as for host.flow*	
vswitch.flow	virtual	*as for switch.flow*	

The metric set `host.ipmi` captures monitorig information as received through the IPMI protocol. This particularly includes information on the host's power state and the information on the current power consumption. To represent network statistics not only on host-level, we introduce the metricset `switch.network` that contains information about bytes sent/received on a per-port basis. The metric sets `host.flow`, `switch.flow` as well as `vm.flow` contain information about UDP and TCP flows, in particular the amount of bytes sent and received on that flow. Hence, using them ,we are able to derive communication patterns between hosts and virtual machines. `vswitch.flow` contains the very same information about virtual switches on the virtual layer.

2.3 Implementation

In Section 2.3.1 we summarise the technology used for realising the data acquisition and storage parts of RECAP. The details of the technologies are presented in (Domaschka, Narvä, et al. 2019). Section 2.3.2 presents statistics on the data sets collected over the project's run-time.

2.3.1 Implementation Technology

The implementation technology is widely unchanged compared to (Casari and Domaschka 2018). Wherever possible, RECAP uses established techniques for collecting, processing, and storing monitoring data. Table 4 summarises the respective components and their main technology used in the project for data collection and data storage. (D4.4) will present the technical details while (D4.3) explains the technical set-up in the RECAP testbeds.

Table 4 Overview of baseline technology for monitoring components

Component	Name	Reference
dispatcher	Logstash	https://www.elastic.co/products/logstash
database sink	InfluxDB time series database	https://www.influxdata.com/
emitter sink	Apache Kafka publish subscribe	https://kafka.apache.org/
data lake sink	CSV files	---
data lake synchronization	Rsync	https://rsync.samba.org/
probes	Metricbeat	https://www.elastic.co/products/beats/metricbeat
	Filebeat	https://www.elastic.co/products/beats/filebeat
	SNAP	https://github.com/intelsdi-x/snap
	Flowexport	https://github.com/cha87de/flowexport
	ipmi exporter	https://github.com/lovoo/ipmi_exporter

2.3.2 Data Sets and Data Collection

This section summarises the data sets collected for RECAP since the beginning of the project. For each use case and testbed, we present the collected metrics, the size of the data sets and the periods for which data collection has been enabled.

Table 5 Data Set Collected in RECAP

data set	data points	compressed	description
Ulm testbed	6.26 billion	82 GB (comp. rate ~94%)	Monitoring data from the UULM testbed (infrastructure level) captured since (Domaschka, Griesinger and Willis, et al. 2017)
Tieto (use case A)	4.38 billion	16 GB (comp. rate ~95%)	Monitoring data from the Tieto testbed (infrastructure) and use case A (application layer)
Linknovate (use case B)	1.13 billion	12 GB (comp. rate ~94%)	Monitoring data from the Linknovate production system captured since (Domaschka, Griesinger and Willis, et al. 2017)
BT CDN (use case D.1)	0.09 billion	917 MB (comp. rate ~94%)	Online monitoring data from the BT vCDN use case with metrics from infrastructure to application level. Other data has been provided as offline monitoring data.
BT (use case D.2)	1.07 billion	11 GB (comp. rate ~95%)	Monitoring data from the BT cloud connect use case with metrics from infrastructure to application level.

2.3.3 Use Case Coverage

A full list of metrics collected per use case can be found in Annex Section 6.2.

3 Data Analytics

In this chapter we provide an exploratory data analysis (EDA) of the Use Cases identified in RECAP. Also, we include a Data Analytics Methodology where we describe how the proposed RECAP methodology can be applied to new Use Cases beyond the scope of the project.

3.1 Exploratory Data Analysis of Use Cases

In this section, we provide an Exploratory Data Analysis (EDA) of the datasets corresponding to the four Use Cases identified in the RECAP project. This EDA is a preliminary step for the more advanced analysis of the data (interpretation of the results) and the development of the predictive models (see deliverables (D6.2), (D7.4) and (D8.4)).

3.1.1 Use Case A: Infrastructure and Network Management

Tieto focuses on offering new innovative solutions leveraging on the possibilities enabled by 4G-and-beyond mobile technologies in conjunction with fog/cloud computing. This includes solutions for industry verticals like eHealth, eCommerce and automotive. As it was described in (Domaschka, López, et al. 2018), Use Case A demonstrates through the output of RECAP how the profiling and simulation of infrastructure, network functions and service function characteristics can be automated to ensure the desired QoS for the different networks managed by Tieto. As part of this use case, Tieto has provided access for the project to Tieto's testbed, which emulates a common physical infrastructure spanning over multiple data centres with different characteristics. It can provide end-to-end communication and content services for different categories of horizontal content services with different QoS requirements. In this section, we provide a descriptive data analysis of the data collected from this testbed.

We aim mostly at the workload behaviour represented by the network traffic on both the control plane and data plane, and also plan to identify the patterns of end-to-end delay, the relationship between the traffic on these two planes, and that between the network traffic and the system resource (CPU, memory) usages. The analysis corresponds to the data collected from multiple simulating scenarios.

In the simulations, data sent by end-users (simply called *users*) and data pulled from the Internet services (simply called *servers*) are generated by two different data generators. Generated data will be passed through SFCs (service function chains) including eNodeB, Serving Gateway (SGW), and Packet Data Network Gateway (PGW). The simulation model and the structure of such an SFC can be found in deliverable (Le Duc and Östberg 2018).

Figure 1 presents the behaviour of workload collected at users and servers. Note that traffic sent by users, which would be considered as the traffic caused by user requests to servers, will be received at the servers. And in the same manner, traffic sent by servers is consumed mostly by the users.

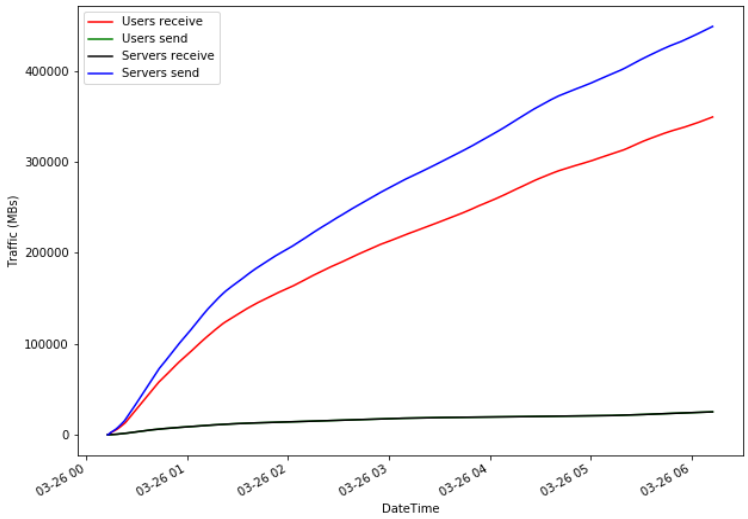

Figure 1. Network traffic on user plane.

Traffic sent by users is almost the same as that received by servers. Due to that it is impossible to see two separate lines (green and black) in the figure. It is noted that there is a gap between the traffic sent by servers back to the users. This could be because of some rate of packet drops in the simultions due to some policy settings. A comparison between the volume of data sent and received by users and that of servers can also be extracted from the figure. It is observable that the traffic pulled from servers is much higher than traffic sent by users (which are mostly the user requests of contents). This reflects situations in real life where most of the contents accessed by users nowadays are HD streaming contents which are massive volume and consume high network bandwidth.

For more details of the data flow on the SFCs, by Figure 2 we demonstrate how the traffic sent by users (which is the green line in the Figure 1) traverse or flow in the given SFC. It is observable that traffic traverse from eNodeB to SGW, and then to PGW are exactly the same, which is presented by three non-distinguishable lines (orange, green, and red) in the figure. A slight different between the traffic generated/sent by users to eNodeB and the traffic received/forwarded at/from eNodeB to SGW. This could be because of some heading data added to the original user traffic by the infrastructural network or its trasmitting protocols.

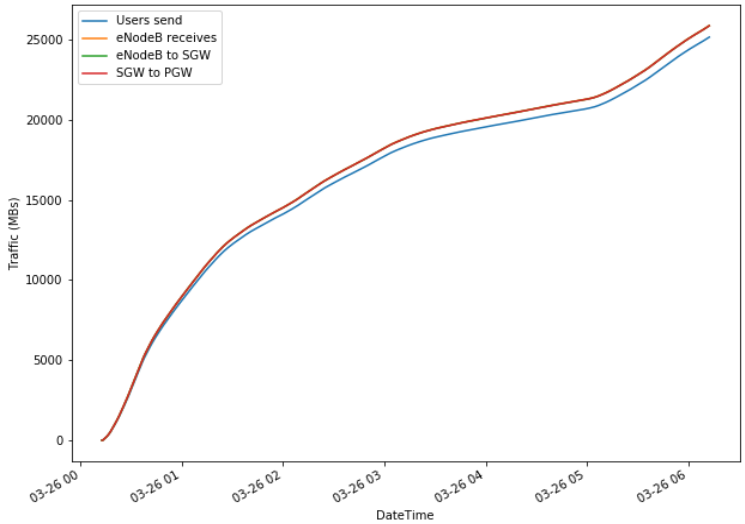

Figure 2. Network traffic flow on user plane.

Figure 3 illustrates the behaviour of workload on control plane. Because of some static configuration/strategy of communications between VNFs to other entities, traffic sent and received by users (at user-devices) are almost a constant. In deliverable in deliverable (D6.2). Final Workload, Load Propagation, and Application Models will provide an analysis of the characteristics regarding the relationship between the traffics on user plane and control plan, and the impact of traffics on control plane on other metrics.

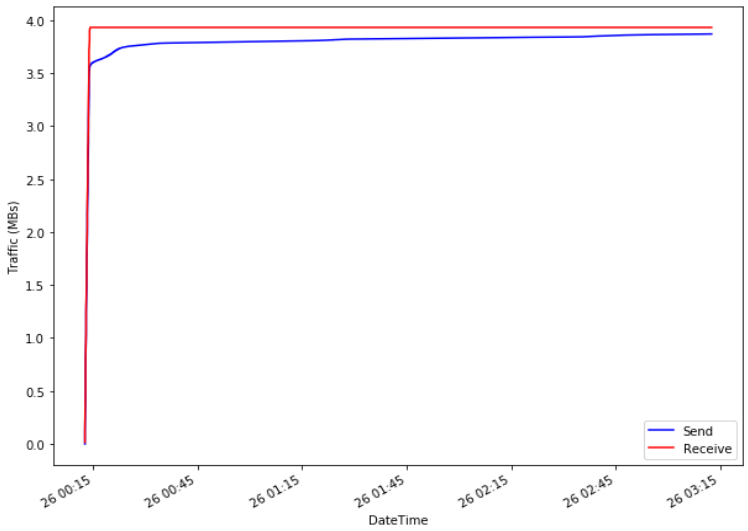

Figure 3. Network traffic flow on control plane.

3.1.2 Use Case B: Data Analytics Engine

Linknovate.com is a company devoted to the generation of knowledge insight by aggregating large amounts of research and scientific data, by using data mining and data analytics techniques (20 million documents, over 30 million expert profiles, over 2 million entity profiles, and more than 200 million innovation topics). Technology-wise, Linknovate.com runs on a distributed architecture composed of several nodes deployed in the cloud. In order to keep reduced response times when request peaks occur, Linknovate currently over-provisions the number of nodes. This is a clear source of inefficiency, which can be optimised in order to reduce costs and energy consumption. As it was described in (Domaschka, López, et al. 2018), this use case is about the dynamic provisioning of computing resources, together with the geographical co-location of nodes and users.

In this section we provide an EDA of the response size, response time, and workload (i.e., number of requests) of the main entry point of the Linknovate search engine. Please, refer to (Domaschka, López, et al. 2018) for more information about the use case. The analysis of the workload is based on the log files generated by the nginx web server during a period of six months.

Quality of the Data Set

As the initial step of the EDA, we evaluate the quality of the datasets. Given that the workload is based on the number of user queries, we evaluate the errors (number of invalid requests) reported by the web server:

Table 6. Statistics of the response codes returned by the Linknovate search engine.

HTTP Status	Description	Number of Entries
200	OK	1,692,070
302	Moved Temporarily	175,577
301	Moved Permanently	111,109
503	Service Unavailable	82,905
404	Not Found	45,439
304	Not Modified	27,782
206	Partial Content	8,024
499	Client Closed Request	2,497
403	Forbidden	1,190
400	Bad Request	699
500	Internal Server Error	117
405	Method Not Allowed	100
502	Bad Gateway	72
504	Gateway Timeout	19

Although 78.78% of the queries were successfully answered by the search engine, the number of errors is too high for this kind of service. *It is worth mentioning that our partner Linknovate took early advantage of the EDA performed in the context of the RECAP project*. Such EDA prompted Linknovate to move their operations to a more advanced configuration based on an Apache web server.

Origins of Requests

Table 7 provides information about the source of the requests. The table has been ordered by number of requests, and only the first 10 entries are shown. Note that the IP addresses and other confidential data were anonymized by Linknovate before providing the data to RECAP.

Table 7. The top 10 IP addresses directing the largest number of requests to the Linknovate search engine.

anonymous IP	count
IP_1	184,381
IP_2	124,093
IP_3a	907,11
IP_3b	85,367
IP_3c	78,317
IP_3d	70,576
IP_4	69,950
IP_5	55,806
IP_6a	44,459
IP_6b	29,771

A remarkable 45% of the requests are originated by a few IP addresses. Linknovate has confirmed that these IPs correspond to web-spider bots from large search companies, like Google, Yandex or Iparadigms. We have computed a 55% percent of the total data transfer, and a 34% of the total processing time to be a consequence of the requests from these search engines. However, since Linknovate considers it important to appear in the results of these search engines, no remediation action was performed.

Response Size

Figure 2 shows a histogram of the sizes of the responses of the search engine. The x-axis is in logarithmic scale, due to the presence of a number of outliers with very large response sizes. Some of these outliers correspond to internal tests performed by Linknovate engineers, but others are legitimate requests from actual users.

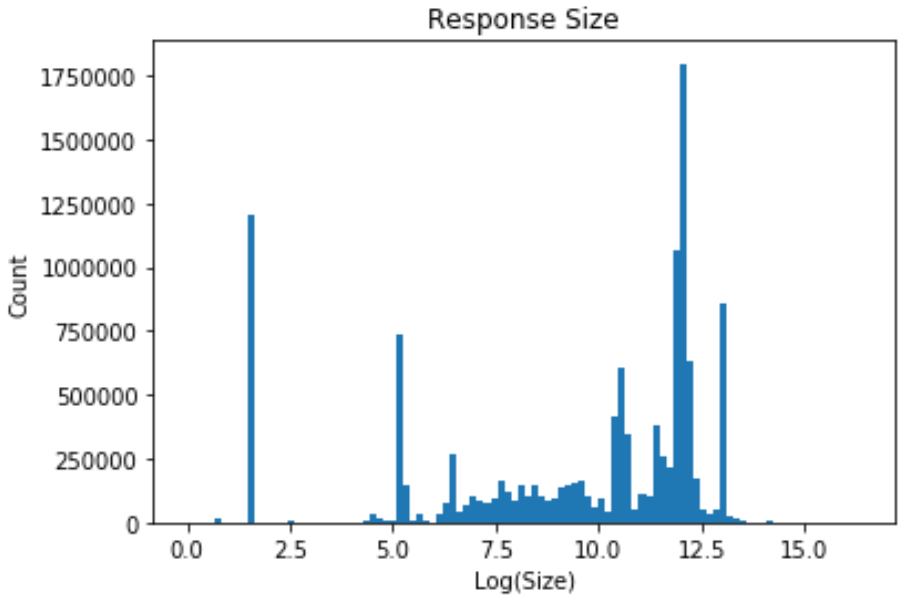

Figure 2. Histogram of the distribution of the Linknovate search engine's responses.

Response Time

In the same way, Figure 3 shows a histogram of the (total) time that the search engine requires to provide an answer to user queries. Although it also shows a strong left skew, the number of outliers is less relevant than in case of the response size.

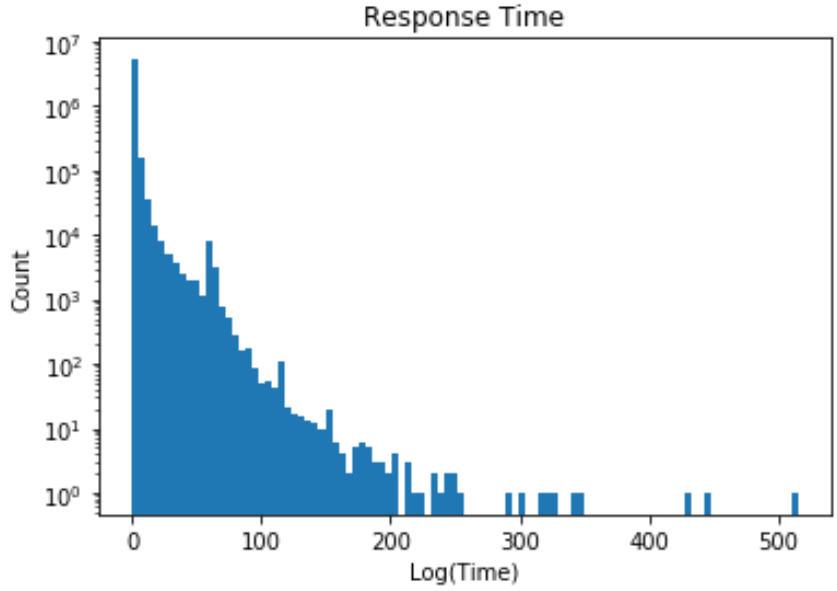

Figure 3. Histogram of the distribution of the Linknovate search engine's response time.

Response Size vs. Response Time Correlation

We have also studied the correlation between the response size and the response time. The Pearson correlation coefficient is 0.18, or 0.2 if we consider only the 200 (OK) requests, suggesting a positive (although weak) correlation between both values. The following figure shows a combined scatter plot for both variables.

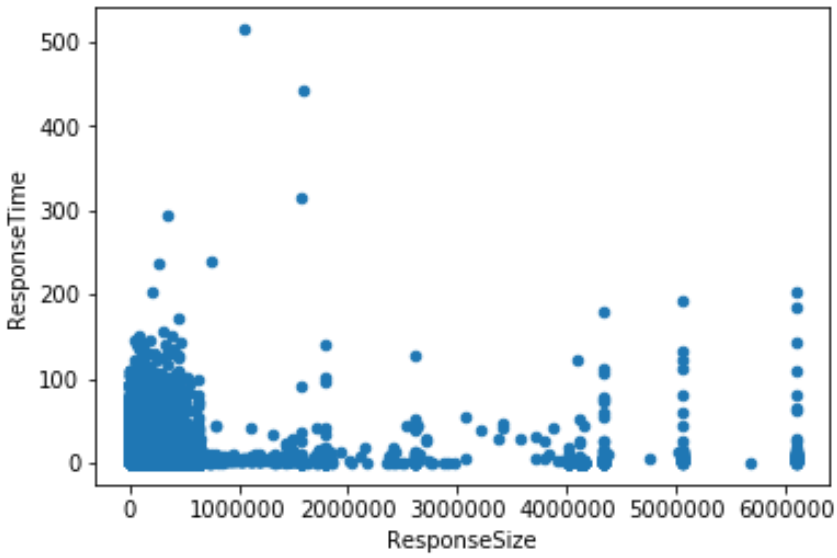

Figure 4. Scatterplot of the response size and time for the Linknovate search engine.

Number of Requests

Finally, we have derived a time series with the number of requests that the server receives, aggregated over intervals of 30 minutes. This time series is the target workload that is analysed and modelled in other RECAP deliverables (see, for example, (D6.2).

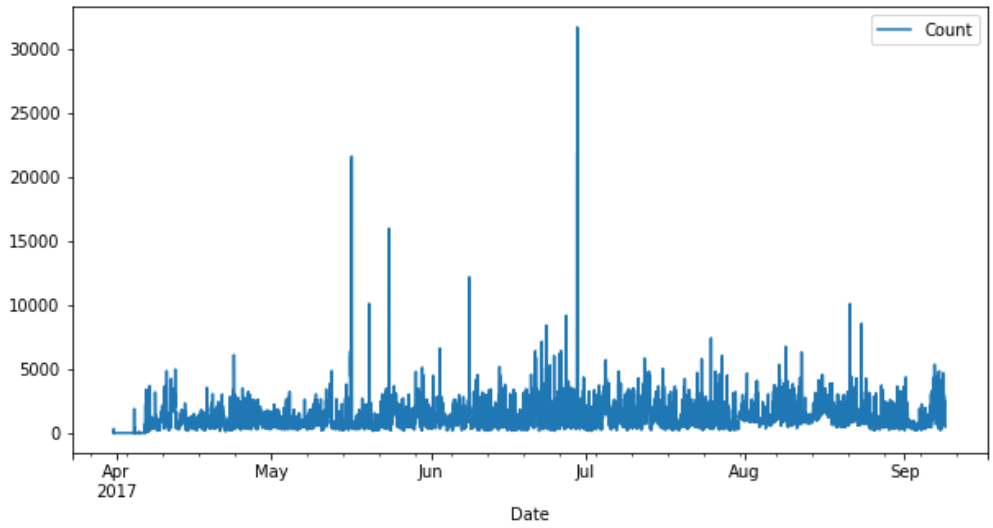

Figure 5. Time series of the Linknovate search engine's workload (data aggregated over windows of 30 minutes).

The peak workload at the end of June correlates with an interview to Manuel Noya, founder and CEO of Linknovate, being broadcast by a radio station. Other peaks correspond to similar events.

3.1.3 Use Case C: Edge/Fog Computing for Smart Cities

The Internet of Things (IoT) is the inter-networking of different physical devices and other items embedded with electronics, software, sensors, actuators, and network connectivity which enable these objects to collect and exchange data. The current cloud computing paradigm does not provide a solution for large-scale IoT scenarios like smart cities: the data produced and the information obtained from them, has value mainly in the scope of the city, but in order to extract this value the data need to be hauled through data networks to be stored and processed. Afterwards, the derived information has to be moved back to the city's infrastructure where they can be forwarded to the interested entities and authorities. One of the current challenges of fog/edge computing is to look for new approaches that allow data produced in a local context are stored and processed as close as possible to that context. As it was described in (Domaschka, López, et al. 2018), the scenario proposed for Use Case C is a Smart City, where different sorts of devices (weather stations, Low voltage energy meters and traffic data, for instance) are deployed over the city area, and they send data continuously to different systems and applications in order to be processed.

In this section, we provide an exploratory data analysis for the Smart Cities Use Case. In particular, we analyse the behaviour of cars in a city. The data correspond to a two-day traffic trace captured in the city of Cologne, Germany. Traffic has been simulated using the urban simulation tool SUMO[1]. The city has been divided into five different areas (centrum, east, west, north and south). Every minute, we compute the number of cars present in each area. The number of cars is used in other deliverables (for example in (D6.2)) to compute workload, measured in terms of volume of data transferred between the cars and the edge of the network.

Table 8 shows a descriptive analysis of the aggregated information (full days) for cars, studied per areas.

Table 8: Descriptive statistics of the number of cars in each city zone for the SATEC Use Case.

	Centrum	East	North	West	South
sample count	2,492	2,492	2,492	2,492	2,492
mean	6,048.34	781.14	729.74	377.64	617.86
std	2,555.97	315.02	312.44	211.83	274.73
min	2,023	341	245	96	181
25%	3,246	526	429	172	352.75
50%	6,693	682.5	735	365.5	599.5
75%	8,382.5	1033	987	533	880
max	10,287	1753	1395	912	1085

As we can see from the summary table, most of the cars are traveling around the city centre. This fact raises the need to have a tool to better divide the city in areas, that is, split the workload evenly, cf. (Loomba, Quinn und Ruane, et al. 2019).

The sub-figures in Figure 6 show the empirical distribution of the number of cars in the different zones.

[1] http://sumo.sourceforge.net/

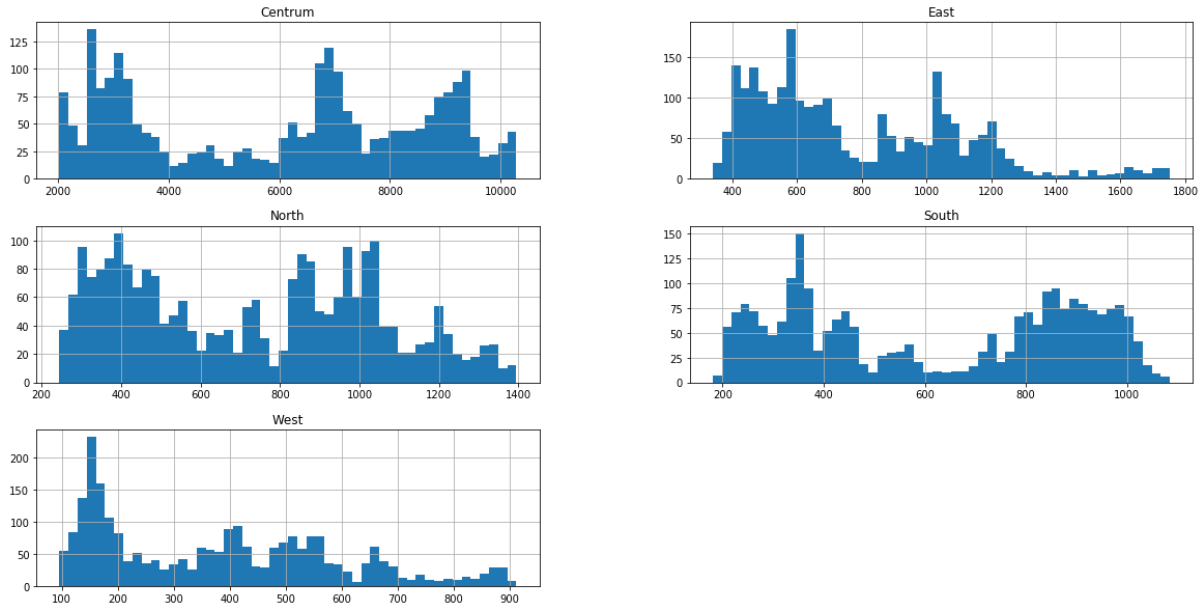

Figure 6. Histogram of the distribution of the number of cars in each city zone for the SATEC Use Case.

The multi-peak nature of all histograms suggests the need to model different traffic scenarios. For example, the three peaks in the Centrum area suggest that there is a scenario with a very low number of cars, a second scenario with moderate number of cars, and a third scenario with high volume of traffic. On the contrary, in the South area, either we have a very low number of cars, or a very high number, with no apparent intermediate situation, whereas in case of the West area, there is no evidence of high traffic. The evolution of traffic over time along one day's span for the different zones is depicted in Figure 7.

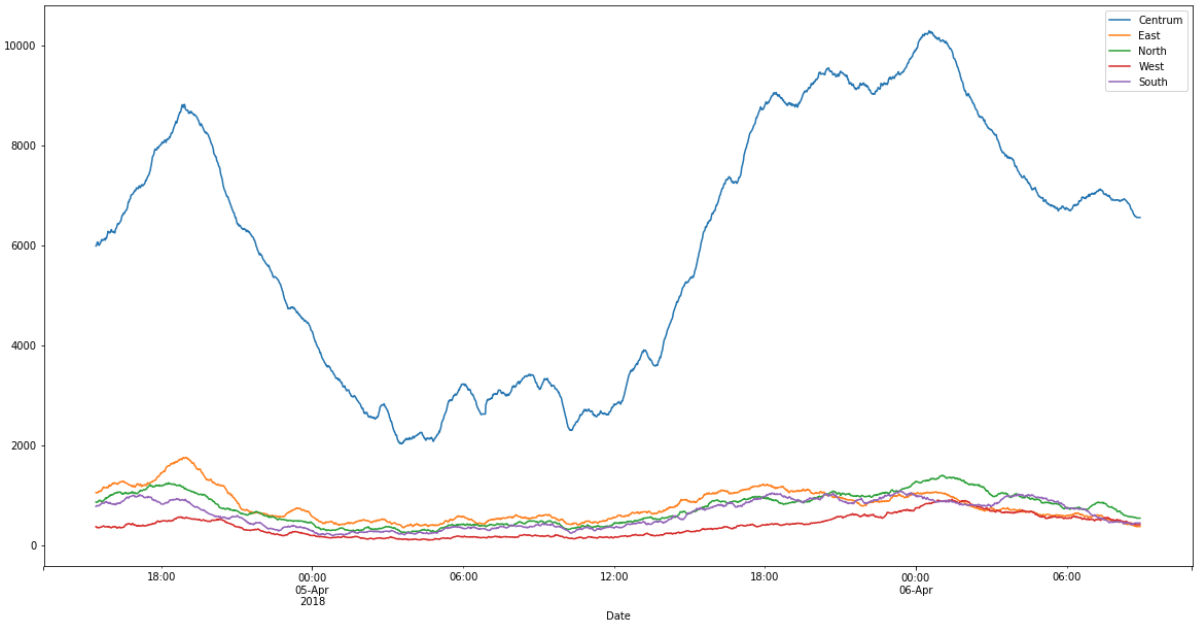

Figure 7.Evolution of the number of cars in each region over one day's span in the SATEC Use Case.

3.1.4 Use Case D1: vCDN

Content Distribution Networks (CDNs) offer a distribution service to content providers that puts content on caches closer to the content consumers or end-users. A network operator such as BT

is likely to have hardware from several CDN operators deployed at strategic points in its network. This creates several potential issues: it is hard to organise sufficient physical space to support all the CDN operators; a lot of energy is needed to power and cool all the equipment; a lot of physical effort is needed when a new CDN operator arrives or an existing one disappears. Such factors make it attractive to consider a Virtual CDN (vCDN) approach that aims to replace the multiple customised physical caches with a standard server and storage running multiple virtual applications per CDN operator. As it was described in deliverable (Domaschka, López, et al. 2018), Use Case D1 vCDN provides BT and CDN operators with the tools to plan the optimum location and amount of resources required and deploy the vCDN systems and infrastructure just in time to optimise CDN performance and resource utilisation.

In this section, we will provide an exploratory data analysis for the vCDN use case. Specifically, the workload data collected at caches connected to BT's core network is analysed to identify characteristics and/or bring an insight into the workload behaviour which will be used for workload modelling in WP6. The workload at a cache is represented by two kinds of traffic: traffic generated by pulling contents from origin content servers and traffic served by the cache. Datasets provided by BT are collected at one-minute interval within the recent months. Data is collected at different cache servers at different sites. We have analysed all the datasets using the same procedure. In this deliverable, the analytical results obtained for one caches located at the core sites of the BT's network are presented.

Serving Traffic

Because most of cache optimisation tasks rely on the behaviour of workload served by caches, we primarily aim to demonstrate the analytical results of the traffic generated by pulling cached contents from caches to serve client's requests, namely *serving traffic*.

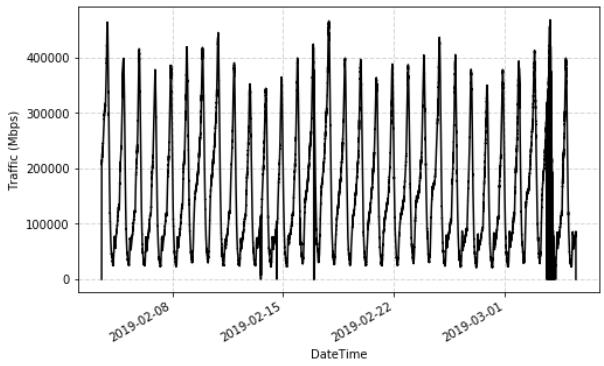

Figure 8 Serving Cache Traffic: Original Traffic Data

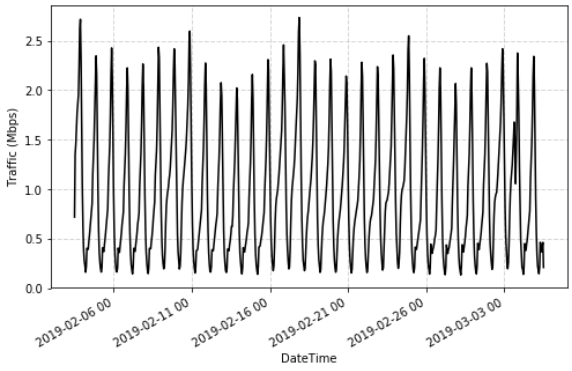

Figure 9 Serving Cache Traffic: Hourly Resampled Traffic Data

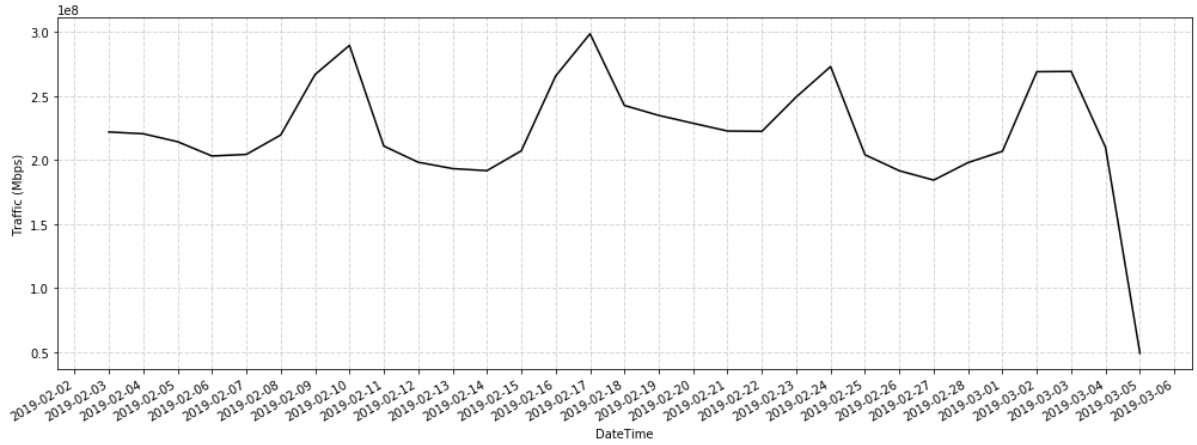

Figure 10 Service Traffic of a Cache: Daily Resampled Traffic Data

In order to see the aggregated traffic per hour and per day, besides presenting the origin serving traffic in Figure 8, we also provide Figure 9 and Figure 10 that show the resampled data (hourly resampling and daily resampling, respectively) which are derived from the original data. We observe a weekly seasonality/pattern of user access in Figure 10, which shows that the serving traffic is at the high peak at the weekends.

Traffic Decomposition

Figure 11 and Figure 12 show the decompositions of the serving traffic of the cache and the traffic generated by pulling contents from origin content servers (called *received traffic*), respectively.

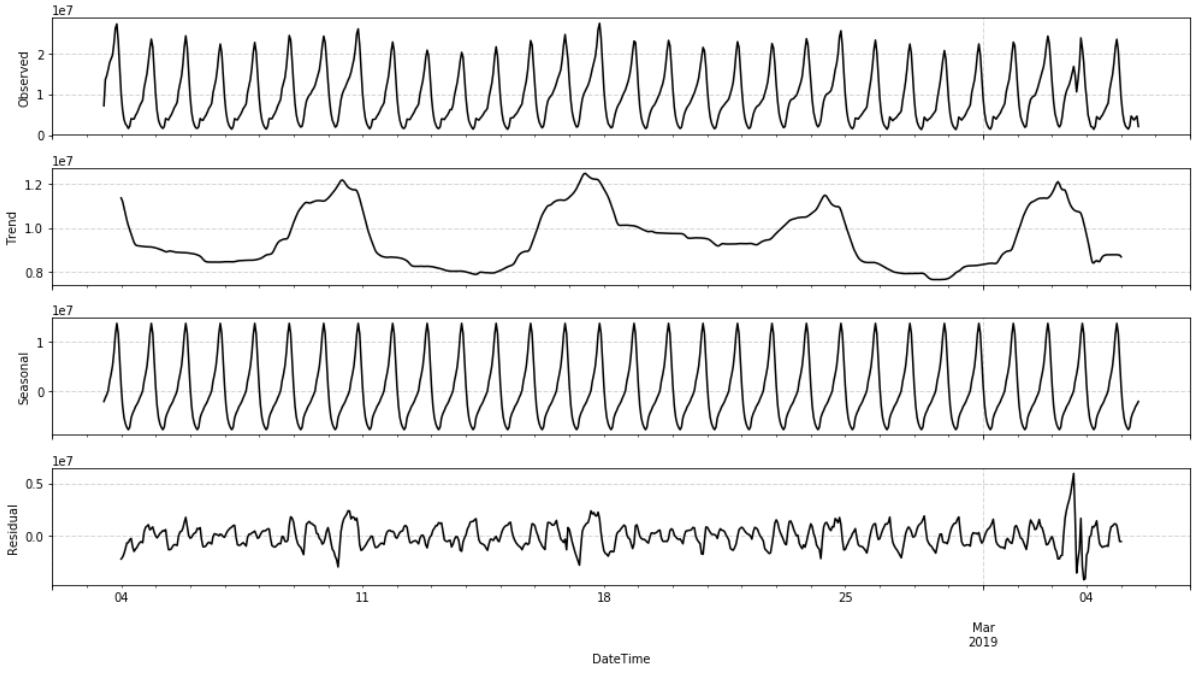

Figure 11 Decomposition of serving traffic of a cache.

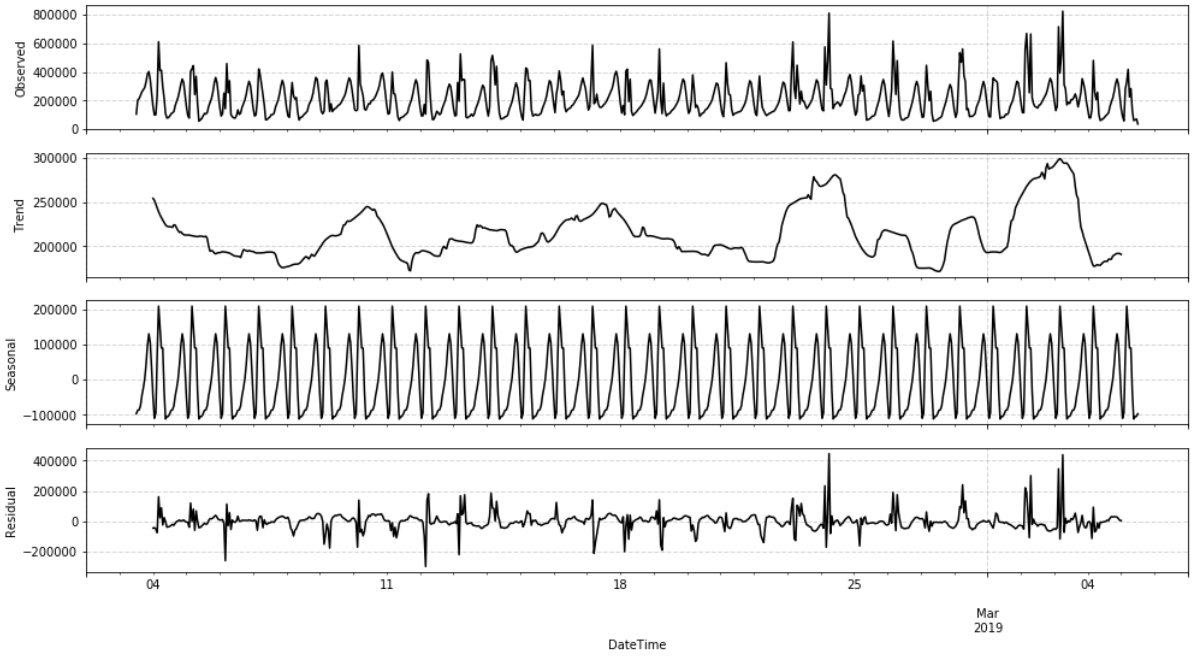

Figure 12 Decomposition of received traffic at a cache.

Through the two above figures, it can be observed that the serving traffic is extremely higher than the received traffic. This exactly reflects the nature of caching systems: pulling and caching a content item is instrumental to serve a massive number of requests from clients, which means the content item is pulled from an origin server once, but will be downloaded by multiple clients at different times. While there are no specific trends for both kinds of traffic, we are able to observe seasonality in the traffic. And there is only one peak point in serving traffic, but two points in received traffic. For further clarification and comparison, Figure 13 demonstrates a subset of data points extracted from the seasonal composition of the two kinds of traffic.

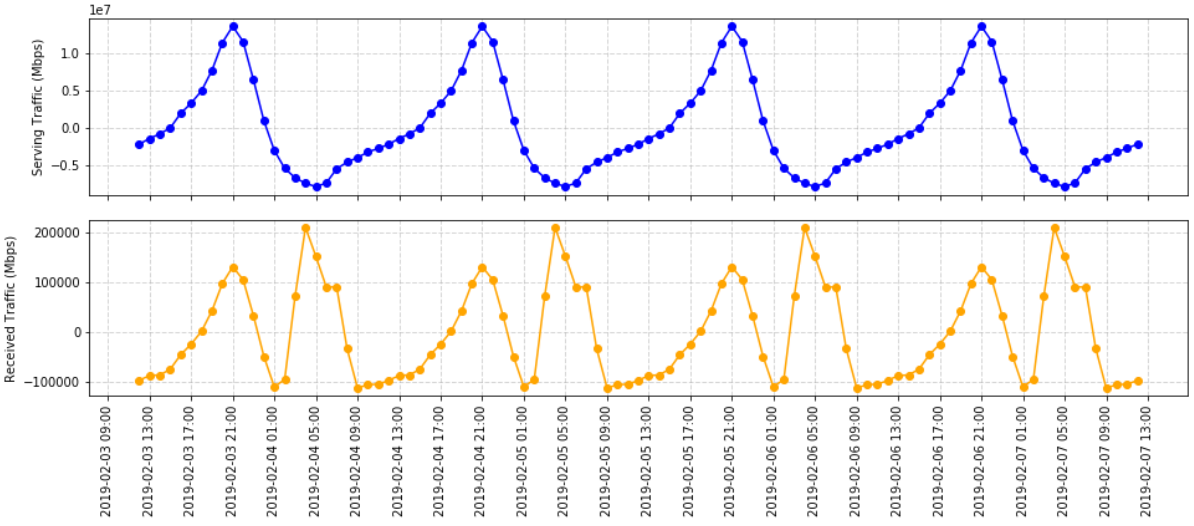

Figure 13 Seasonal compositions of cache traffic.

In Figure 13, a high volume of serving traffic is observed at 20:00 – 22:00 every day with the peak is at 21:00. This illustrates that users access Internet contents the most during this period of time in a day. This is also corresponding to a peak in the received traffic. It is understandable that the user access to contents highly likely causes the pulls of the contents from origin servers. In

addition, there is another peak at 4:00 every day in the received traffic. The reason behind could be because of the content synchronisation or refreshing at caches. The cache system would be configured to refresh its caches at that time.

3.2 A Methodology for the Analysis of New Datasets

In this section, we are going to describe the RECAP methodology for the analysis of new datasets and the development of new machine learning algorithms. The methodology is intended to support the application of RECAP's results to new problems related to optimal resource allocation and capacity planning.

The new methodology composed of five main steps is summarized in Figure 14. The tasks to be performed at each individual step, and the decisions to be taken are detailed in the following sections.

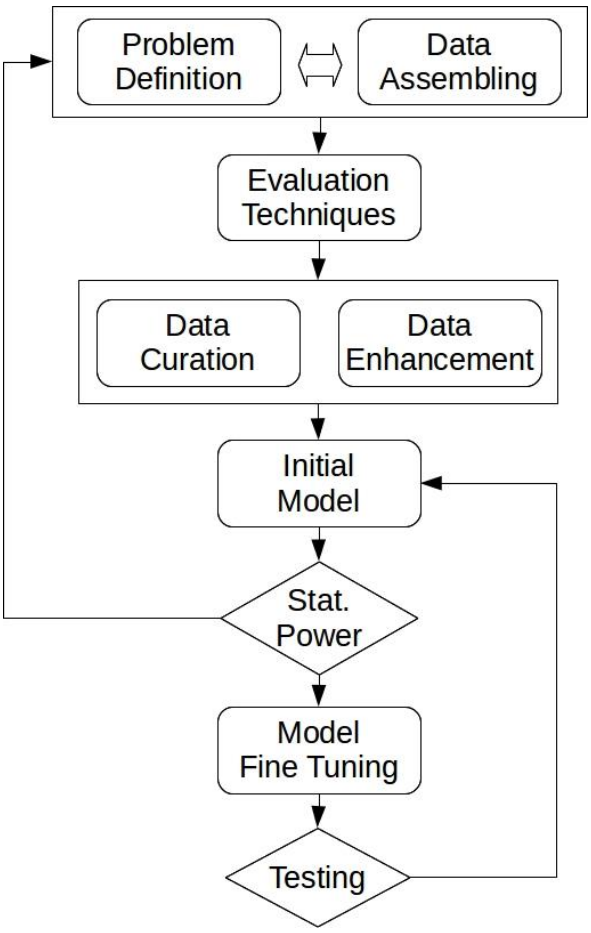

Figure 14. A summary of the main steps of the methodology for exploratory data analysis of new datasets.

3.2.1 Problem Definition and Data Assembling

The initial step in every machine learning project is to clearly *identify the problem* to be solved. The second step is to *identify the available data* that can help solve the problem by applying machine learning techniques. We have merged these two steps, problem definition and data assembling, in one single task due to the high inter-dependence between them. That is, we can solve a problem using a machine learning algorithm only if we have the right datasets to train the models. If the datasets available are not sufficient, we have to change our expectations about the problem identified, find or collect additional data, or cancel the project in the worst case where the data is

definitely insufficient for the problem at hand. It is important not to continue with the methodology described in this section unless we are sure that we have a sufficient amount of relevant data.

For example, in Linknovate's Big Data Engine Use Case, we are interested in identifying the optimal number of data nodes that can deal with the application's workload. In order to solve this problem, we had to train a time series model to predict future workloads, and the training part requires historical information on the performance of the search engine. Since that information was not available at the beginning of the RECAP project, our priority was to start collecting data as early as possible.

As an alternative, in (D5.3), we explain how to enrich existing datasets with synthetic datasets generated using models trained on the same workload data we have been collecting for the RECAP Use Cases. This will speed up future developments based on similar cases and will serve as a baseline to train machine learning models for the optimization of distributed computing systems.

The definition of a problem requires to answer the following questions:

- **Format of the input data**: How many features are we measuring? Are these features continuous or categorical?
- **Format of the output data**: What is the value we are trying to predict? Are we dealing with a classification problem or a regression problem? In case we are dealing with classification, which are the categories?
- **What kind of evidence do we have that the output can be in fact derived from the input?** (e.g., in-house knowledge, academic literature, results from RECAP or other projects, etc.)
- **Are the input features sufficient to predict the output?** Or rather do we need to collect additional information?
- **Is it the problem stationary?** Or rather does it change over time?

Examples of problems and their definition can be found in (Domaschka and Noya, Initial requirements 2017), and examples of datasets and metrics descriptions can be found in (Casari and Domaschka 2018).

3.2.2 Metric for the Evaluation of the Results

In order to evaluate the performance of a trained model, that is, to quantitatively measure how good are the predictions made by the model, we have to select an objective metric and a protocol for the evaluation.

Selecting the metric to evaluate the results of our model is a critical element in the process of building a successful model, since that metric is exactly what the training algorithm will optimize. If the output of the model is a continuous variable, a good example of evaluation metric could be the Root Mean-Square Error (RMSE). In case of a categorical response, typical metrics used are accuracy, or the area under the receiver operating characteristic (ROC) curve (AUC).

A more detailed description of how to select the right metric and how to properly evaluate the results of models is included in (Forsman, et al. 2018).

There are multiple standard techniques in the machine learning literature about how to properly evaluate the performance of a machine learning model, and to early detect problems due to the overfitting of models. Examples of these techniques include:

- **Train/test**: split our dataset in two subsets in such a way that the data used for the training part is different from the one used for the evaluation.
- **N-fold cross-validation**: similar to the train/test split, but repeating the same process N times with the same dataset: the fraction of data used for training remains the same, but a different subset of the training data (chosen at random) is dedicated to the training part, and the remaining to the cross-validation part.
- **Bootstrap**: similar to the above, but applying a random sampling with replacement.

In the RECAP methodology, we recommend to use train/test splits for the early prototyping of models, and to apply a cross-validation to the final models before using them in production, since cross-validation is a heavy computational approach that requires powerful machines.

3.2.3 Data Curation and Enhancement

Some machine learning algorithms are highly sensitive to the presence of errors in a dataset. Moreover, some algorithms do not work at all if some data is missing. A data curation process, i.e., the removal of errors, anomalies and the fixing of missing data, is an important preparatory step before training a model. A visualization of the dataset and a descriptive analysis, could provide highly valuable information about the quality of the data being used in the project. Analytical techniques like outlier detection, or simply the identification of N/A (Not Available) values, could be applied as well. In (Casari and Domaschka 2018) - Section 4.1 Descriptive Statistics, we covered the techniques and procedures used to improve the data quality.

The data has to be prepared in a way that can be fed into the training phase of the machine learning models. The actual transformations to be applied to the data depend on the specific machine learning algorithms used. For example:

- in the case of **neural networks**, it is highly recommended to normalize the values so that all features fall in the same range of values;
- in the case of **decision trees**, this normalization step is not needed.

It might be also necessary to enhance the data by means of deriving new features based on the features that already exist. This data enrichment could significantly improve the predictive capabilities of models. For example, we could have measured the features “total memory in use” and “total number of running processes” for a virtual machine, but in order to train the models it could be more interesting to derive a new feature “average memory in use per running process”.

In (Casari and Domaschka 2018) - Section 4.2 Advanced Analytics, we cover feature engineering.

3.2.4 Model Development

Sometimes it is difficult to identify the right model to be applied to a particular problem. There could be many different alternatives for a single problem. For example, in the case of a classification problem, we could apply techniques like K-means, decision trees, support vector machines, or neural networks. Moreover, each technique could have different alternative configurations: for example, there may be many different neural network architectures that could be adequate to tackle a given problem. An approach to speed up the selection of the right family of models is to test the statistical power of the machine learning techniques. A test of statistical power consists in performing a simple and fast training of the model, perhaps with a subset of the dataset, and check if it presents significantly better predictive capabilities than random guessing. Any family of models with no predictive power should be discarded. If no machine learning

technique present statistical power, perhaps it is the case that our input data is insufficient to solve the selected problem, or that our hypothesis about the relation between the data and the problem is incorrect. If this is the case, we should go back to step one of the methodology (see Section 3.2.1).

A description of the model usually employed to solve issues related to resources allocation and capacity planning is included in In (Casari and Domaschka 2018) - Section 4.3 Prediction and Forecasting.

Once we have selected a machine learning technique and performed an initial training, we have to decide if the model's predictive power is sufficient to solve the problem at hand. In (Forsman, et al. 2018) we describe the requirements that a machine learning model should have in terms of predictive power in order to satisfy the quality of service and the defined service level agreements.

As an alternative to the manual selection of the best architecture for a neural network, we could use the automated machine learning techniques introduced in Section 4.

3.2.5 Regularization and Hyperparameter Selection

The final step of the methodology is to tune the configuration of the model's hyperparameters, that is, configurable parameters whose values must be set before the learning process begins. Hyperparameter optimization makes it possible to obtain the best predictive capabilities from a machine learning model. However, care should be exercised, as hyperparameter optimization exposes a model to a higher risk of overfitting, i.e., the model may perfectly describe the training dataset, but has almost no predictive capabilities.

Each iteration of the hyperparameter optimization requires to select an hyperparameter, change its value, re-train and re-validate the model. The process can be automated. However, such automation increases the risk of overfitting the data, and thus of yielding a model that is not amenable to generalization. In order to avoid overfitting, many algorithms prescribe the use of a regularization parameter, i.e., a penalty added to the accuracy of a model if the model is too complex for the task at hand.

Once all the hyperparameters have been optimized, the final step is to test the model against a test dataset, that is, a dataset different from the datasets used for training and validation. The performance of the model in this last test dataset should be similar to the performance achieved in the validation datasets. A divergent result is a clear sign of an overfitted model. Moreover, if the results are too different, we should discard the model and go back to the previous step of the methodology, and select a different machine learning algorithm.

The algorithms for automated machine learning described in Section 4 of this document have been specifically designed to avoid the risk of overfitting a dataset. These new methods reduce, and in some cases even eliminate the need to optimize hyperparameters. For this reason, they can be used as an alternative to the traditional techniques for the selection of the hyperparameters of a machine learning algorithm.

4 Automated Machine Learning Algorithms

Automated machine learning (AutoML) focuses on automating the process of applying machine learning to real-world problems top-to-bottom. In a typical machine learning application, data engineers must apply the appropriate data pre-processing and feature extraction methods that make the dataset amenable to machine learning. Following those pre-processing steps, data scientists choose the appropriate algorithms and perform hyper-parameter optimization to maximize the performance of their final machine learning model. In some cases, a team of analysts may be required to address the cases that the model could not classify or forecast correctly, and thus provide improved training data. As these steps are often beyond the abilities of non-experts, and may require significant human, software, and computational infrastructure, AutoML was proposed as artificial intelligence-based solution to the ever-growing challenge of applying machine learning (Thorton 2013). Automating the application of machine learning potentially yields simpler solutions in a shorter time than by engineering the data and algorithms manually; moreover, the final models obtained through AutoML often outperform models designed by hand.

One of the objectives of the RECAP project is to facilitate the adoption of the machine learning techniques by the small and medium size enterprises and research institutions that do not have in-house knowledge or technical expertise, that do not have the capacity to hire expensive professionals such as engineers and mathematicians, and that do not have access to high-cost specialized hardware.

Currently, there exist many off-the-shelf packages that have been developed with the aim of providing automated machine learning capabilities. Among them, the most well-known ones include:

- **AutoWEKA**[2], based on the WEKA[3] package for machine learning, is a tool for the simultaneous selection of a machine learning algorithm and its hyper-parameters.

- **Auto-sklearn**[4], based on the scikit-learn[5] package, is a drop-in replacement for regular scikit-learn classifiers and regressors.

- **H2O AutoML**[6] provides automated tools for data visualization, feature engineering, model interpretability and low-latency deployment.

- **Auto-Keras**[7] is a wrapper on top of the Keras[8] library that facilitates the development of neural networks for image classification.

- **Cloud AutoML**[9] is a product by Google that has been specifically designed for the selection of the architecture of a deep learning model.

Most of the algorithms employed to search the optimal architecture for a neural network are too expensive from a computational point of view (Cai 2018). An alternative approach to reduce the complexity and time of search is by means of applying network morphism (Hu 2018), that is, by

[2] http://www.cs.ubc.ca/labs/beta/Projects/autoweka
[3] https://www.cs.waikato.ac.nz/ml/weka
[4] https://automl.github.io/auto-sklearn/stable
[5] https://scikit-learn.org
[6] https://www.h2o.ai/products/h2o-driverless-ai
[7] https://autokeras.com
[8] https://keras.io
[9] https://cloud.google.com/automl

adapting the architecture of the neural network while maintaining its functionality. Algorithms based on network morphism could save search time, since we can reuse the weights of previously trained neural networks. However, network morphism has the limitation that, in general, no global optimal architecture is found. Further research is required to provide an efficient algorithm for searching near-optimal neural network architectures.

Another relevant point to take into account is that the current success of machine learning is mostly due to the availability of large datasets and the increments in the computational power of workstations and servers, not to the better algorithms. In fact, there is a general (unspoken) agreement in the machine learning community that the development of novel, original algorithms, is mostly stagnating (Hinton 2017). A possible cause could be the fact that researchers are focused on the analysis of the accuracy of the models, as a measure of the capability to predict unobserved data. However, other model characteristics can be equally important, but are traditionally neglected when evaluating machine learning models.

For example, even with the aid of large datasets, it could still happen that the data used for training the models constitute a mis-representation of the original entities under study. However, no substantial effort has focused on studying this type of error, beyond applying cross-validation and other ad-hoc techniques to avoid its consequences (mostly, overfitting). Moreover, the analysis of the model themselves has been totally relegated to a secondary plane, and even some reputable data scientists claim that the Occam razor principle no longer applies in the big data era (Domingos 2015).

4.1 The Minimum Nescience Principle

Nescience is an old-fashioned English word that means “lack of knowledge or awareness”. The *theory of nescience* (Leiva 2019) has been developed with the aim of quantitatively measuring how much we do not know. If we want to compute how much we do not know about a research topic or problem represented by a dataset we need to quantify two things: how much we already know about that topic, and the theoretical limit of what can be known. What we already know is given by our current best model for a data set. The theoretical limit of what can be known about a topic is given by its perfect model, i.e., the shortest possible algorithm that allows us to reconstruct the original dataset.

Unfortunately, in general, we do not know that shortest possible model, since our knowledge is incomplete, and our representation might be incomplete as well, or even wrong. However, we know that this description must be necessarily an incompressible string, otherwise it would contain redundant elements that can be removed, and so, it would not be the shortest possible one. By computing how much we can compress our current model, we can estimate how far we are from the perfect model.

In order to find the model that yields the minimum nescience, we have to characterize the multiple sources of errors that can affect the performance of a model. The first one is due to the fact that our measuring instruments or data collection devices might not be perfect. That is, if the ideal representation of a topic E is the dataset D, in practice we could be working with a dataset that it is close to the ideal one, but not equal. We call this type of error *miscoding*. The second source of error is due to the fact that our models are not perfect. Ideally a model M should allow us to fully reconstruct the dataset D, but in practice it produces a dataset which is, again, close to D but not equal. This second type of error is what we call *inaccuracy*. Finally, the third type of error deals with the models themselves. We are interested in the shortest possible model M that describes D, so

we can understand the original topic and derive conclusions (for example, predictions). However, our current best known model probably will be different, usually longer than the perfect one. This third type of error is what we call *surfeit*. In this project we propose to combine these three types of errors, miscoding, inaccuracy and surfeit, in a single quantity called *nescience*. In this way, nescience will be a quantitative measure of how much we do not know about a topic or problem, given our current best known model. Nescience will allow us to automate the process of finding the best possible model for a dataset. In the rest of this section we describe in detail these three types of errors and how they can be applied in practice.

Our measure of the ***inaccuracy*** of a model *M* for a dataset *D* will be the length of the shortest computer program that, given as input the model *M*, is able to print the dataset *D*. Intuitively, we seek to measure how difficult (in terms of number of symbols) it is to fix the errors introduced by the model. Formally, the inaccuracy of a dataset *D* and a candidate model *M* is

$$\iota(D,M) = \frac{K(D/M)}{K(D)}$$

where $K(D/M)$ is the conditional Kolmogorov complexity of the dataset *D* given the model *M*. The normalization factor $K(\mathrm{D})$, the Kolmogorov complexity of the dataset *D*, is introduced to guarantee that miscoding, inaccuracy and surfeit have the same scale. We recall that Kolmogorov complexity is a non-computable quantity. Therefore, we will approximate $\iota(D,M)$ by the ratio *Comp*(*E*) / *Comp*(*D*), where *Comp*(*E*) is the length of the compressed version of the subset *E* of *D* composed by those points that have been misclassified by the tree, and *Comp*(*D*) is the length of the compressed version of the full dataset.

Surfeit is a quantity that measures the amount of unnecessary elements contained in a model. Given a dataset D, the surfeit of a model M for that dataset, denoted by $\sigma(M,D)$, is defined as the relative difference between the length of this model and the length of the optimal model for that particular dataset. In the minimum nescience theory, the theoretical limit of what can be known about a dataset is by definition its best or perfect model. This is given by the shortest possible model that allows us to fully reconstruct that dataset. Such length is provided by the concept of Kolmogorov complexity (Vitanyi 2008) of that dataset. Formally:

$$\sigma(M,D) = 1 - \frac{K(D)}{L(M)}$$

The Kolmogorov complexity is non-computable, so we have to find a suitable approximation, by using standard compression algorithms, or the results of coding theory.

Miscoding refers to the problem of having a dataset that does not fully reflect all the characteristics and nuances of the topic under study. For example, if the dataset contains measurement errors, it is too small, or it includes features that are not relevant for this particular study. From a theoretical point of view, miscoding is given by the relative length between the shortest computer program that can fix the wrong dataset D and the Kolmogorov complexity of an (unknown) ideal dataset $\widehat{D}$ that fully describe the entity. Formally:

$$\mu(E,D) = \frac{K(\widehat{D}/D)}{K(\widehat{D})}$$

In this case, not only the quantity is based on the non-computable concept of Kolmogorov complexity, but it is also requires the use of the abstract concept of oracle Turing machines (Soare 2016) to deal with the unknown dataset $\widehat{D}$.

Nescience is a measure of how well we understand a research topic or problem, represented by dataset, and given our current best model. Nescience, denoted by $\nu(M, D)$, is given by a function of the already defined quantities miscoding, surfeit and inaccuracy. Formally:

$$\nu(M, D) = f(\mu, \sigma, \iota)$$

There are multiple candidate functions that can be used in practice to compute nescience. Examples of functions that have been already evaluated at IMDEA include the Euclidean distance, the inner product, the arithmetic mean, the harmonic mean, and the geometric mean, all yielding comparable results. The optimal function still requires further research, but ideally it should be continuous, differentiable, convex, and present a low computational complexity.

The goal of the RECAP project is to provide an algorithm to find a model that minimize our nescience given a dataset. Unfortunately, this is an inherently non-computable problem. However, it can be approximated if we restrict the search space to a given family of models (for example, a particular method in the family of *machine learning algorithm*, which are our main focus in this context) and we allow a suboptimal nescience computation. In the following, we review how to employ the minimum nescience principle in order to improve two of the most popular machine learning algorithms used in everyday practice: **decision trees**/random forests, and **deep neural networks**.

Decision trees allow us to apply modern machine learning techniques while keeping the resulting models interpretable by humans. This combines the best of two worlds, machine learning and scientific awareness; moreover, decision trees have been proven to be highly successful in addressing the problem of feature selection, that is, in identifying the collection of relevant features of a given dataset that have to be considered in a study. This is one of the main problems to be addressed by auto machine learning methods.

In the last five years, deep neural networks have had a dramatic impact in the performance of machine learning algorithm, especially when applied to difficult problems like image recognition or automated text translation. Today, deep neural networks outperform classical image segmentation and classification algorithms, and even achieve better performance than humans in some cases (see for example board games such as Go). There is a general agreement among data science researchers that the accuracy and applicability of deep neural networks will increase given the availability of larger training datasets, more computational power (e.g., as found in special-purpose processors) and improvements in learning algorithms.

4.2 Decision Trees

As it was described in (Casari and Domaschka 2018), a decision tree is a non-parametric supervised method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. Decision trees go from observations about an item (represented in the branches of the tree) to conclusions about the item's target value (represented in the leaves). Decision trees are important in the context of RECAP because they allow us to identify the relevant features that have to be used to build the application and workload models (feature engineering). For example, in the

BT Use Case, the original monitoring dataset provided by VMware's vcenter tool was composed of more than one thousand variables.

In RECAP, we have proved that optimal, non-overfitting, binary decision trees can be automatically constructed by applying the minimum nescience principle. The developed algorithm is based in a breadth-first tree traversal. For each level and each branch of the tree, the algorithm decides if the addition of a new node decreases our nescience about the dataset; that is, if the increment in the accuracy of the model compensates the increment of its redundancy. If the nescience does not decrease, the node is closed, and no further development of the tree is performed in this branch. The new algorithm has been fully integrated with the open-source, publicly available, scikit-learn machine learning framework[10] to the mutual benefit of both our algorithm (integration with cross-validation techniques, etc.) and the rest of the library (automatic development of tree ensembles, e.g., in random forest training algorithms).

Besides the reduced risk of overfitting the dataset, and the better interpretability of the resulting models, an important advantage of our algorithm is that it eliminates the need to optimize hyper-parameters. For example, with the new method we do not need to limit the depth of the tree to a certain, pre-defined value, nor are we expected to avoid over-optimization by specifying the minimum number of samples that should be represented by each leaf. Moreover, our algorithm does not require to apply post-processing tree-pruning techniques to refine the resulting trees.

Non-trained data scientists tend to focus on the accuracy of algorithms, rather than in the capacity of models to generalize to unseen data (overfitting). This risk has been removed from our algorithm, since, by design, models do not overfit data. Moreover, our algorithm has no hyperparameters to optimize, a task that requires advanced knowledge of the inner workings of the algorithms. This advantages make our algorithm suitable for its use by non-expert users.

The performance of the new algorithm has been evaluated with synthetic datasets, and applied to RECAP datasets. The synthetic dataset used for the experiments is depicted in Figure 15:

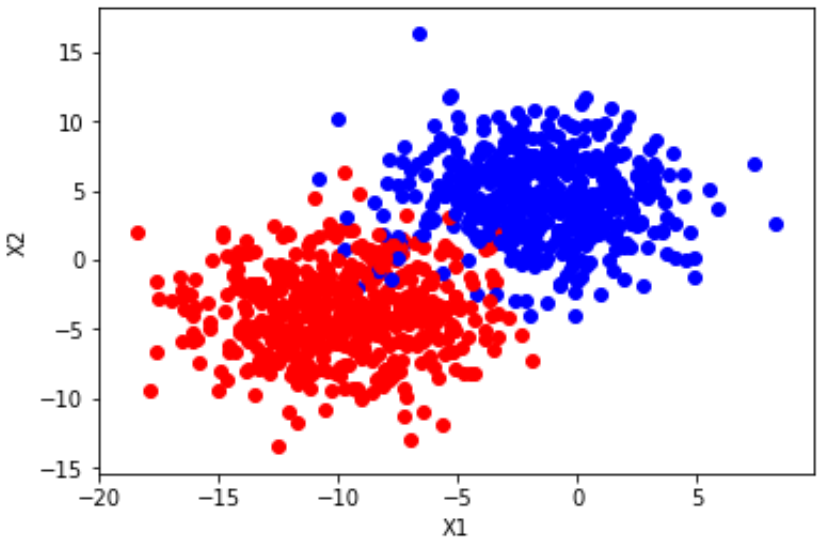

Figure 15. Synthetic dataset employed to show the capabilities of the nescience-based AutoML algorithm for decision tree construction.

[10] https://scikit-learn.org/stable/

The dataset is composed of two isotropic Gaussian blobs that partially overlap. We start with a standard deviation of 2.5 for each Gaussian cluster, so they are easy to separate, and we increase the standard deviation in increments of 0.01, until we reach 4.5, which causes significant overlap. For each value of the standard deviation, we randomly generate 100 blobs and we compute the average accuracy for the algorithm using different sets of points for training and testing. The same approach has been performed with the CART (Classification and Regression Trees) algorithm, which is the standard algorithm used for building decision trees in the machine learning industry. The results of the experiment are shown in Figure 16. On average, our algorithm provides 1.5% greater accuracy than the CART algorithm.

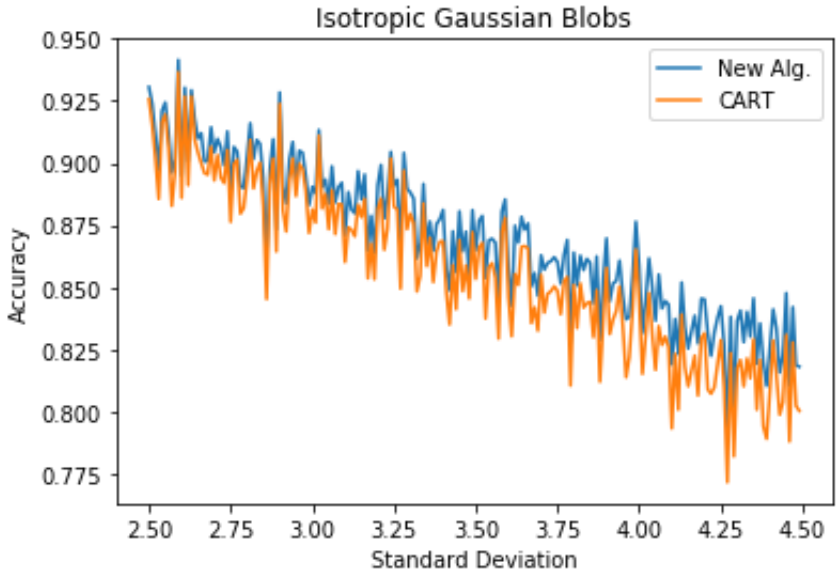

Figure 16. Accuracy achieved by our nescience-based decision tree construction algorithm as compared to the accuracy achieved by the standard CART algorithm.

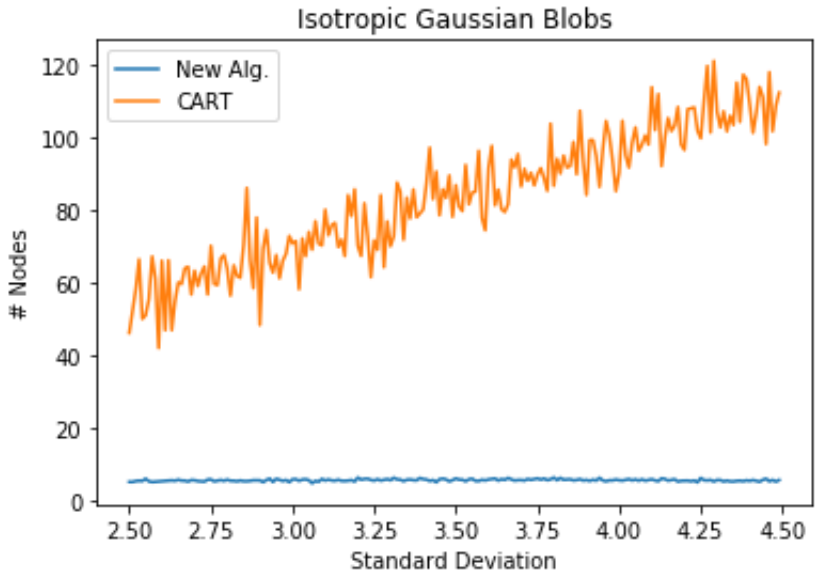

Figure 17. Number of tree nodes obtained by our nescience-based decision tree construction algorithm as compared to the number of nodes in models output by CART. Our algorithm maintains a stable number of tree nodes, suggesting its better ability to interpret the essence of the Gaussian blob dataset, avoiding CART's overoptimization.

For each iteration of the experiment, we have also computed the average number of nodes, including internal and leaf nodes, required by the models to properly classify the clouds in the dataset. The results of this measurement are show in Figure 17. Our algorithm requires an average of 5.7 nodes compared to the average of 83 nodes for the CART algorithm. Moreover, our algorithm is more stable than CART, in the sense that it produces models of similar complexity when it gets similar input datasets (a standard deviation of 0.3 compared to 17.7 for CART).

4.3 Auto Machine Learning with Neural Networks

As it was described in (Casari and Domaschka 2018), a *neural network* (NN) is a machine learning technique inspired by the biological neural networks that constitute animal brains. A NN is based on a collection of connected units or nodes called neurons. Each connection between neurons makes it possible for these neurons to transmit signals across the link. The neuron that receives the signal can process it and then signal neurons connected to it. The signal is usually a real number, and the output of each neuron is calculated by a non-linear function of the sum of its inputs. Connections typically have a weight that gets adjusted as learning proceeds. The weight increases or decreases the strength of the signal at a connection. Neurons are organized in layers. Different layers may perform different kinds of transformations on their inputs. Signals travel from the first (input), to the last (output) layer.

In the context of RECAP, we have developed an Auto Machine Learning algorithm for the identification of near-optimal neural network architectures. The algorithm is based on the application of the minimum nescience principle to the search of optimal neural networks architectures. Starting with a minimal neural network, each iteration of the algorithm evaluates the best way in which the network can grow, that is, by adding a new layer, by adding a neuron to an existing layer, or by adding a new attribute from the dataset being studied. During each iteration it is also evaluated whether shrinking the network instead of growing it increases the accuracy of the model for the given dataset.

Contrary to decision trees, where adding a new node to the tree guarantees that the accuracy remain the same or increases, neural networks do not exhibit a non-decreasing accuracy. Given the stochastic nature of the training process of a neural network, it might happen that a good candidate model is more inaccurate than an otherwise sub-optimal model, only due to the good model getting stuck in a local minimum during the training process. In order to deal with this problem, the algorithm makes it possible to increase the size of the network even if the accuracy does not decrease, subject to the analysis of the stability of the accuracy as a signal (if the accuracy is sufficiently stable, no lower-accuracy neural networks will be accepted).

A proof-of-concept of the feasibility of the application of the minimum nescience principle to the derivation of neural networks has been implemented in the context of RECAP with positive and highly promising results. Figure 18, Figure 19, and Figure 20 shows snapshots of the neural network building process applied to the MNIST dataset (of hand-written digit images). The pictures show the candidate neural network being evaluated, and the evolution of the different collected metrics (miscoding, inaccuracy and surfeit), their combination (nescience) and an out-of-sampling validation score (accuracy achieved using a collection of images not used during training). The final model discovered by our algorithm was able to classify correctly more than 90% of the images in an out-of-sample dataset, what it is considered the state-of-the-art for the classification of images using multilayer perceptrons, and this accuracy was achieved using only 12 pixels from the 64 possible ones.

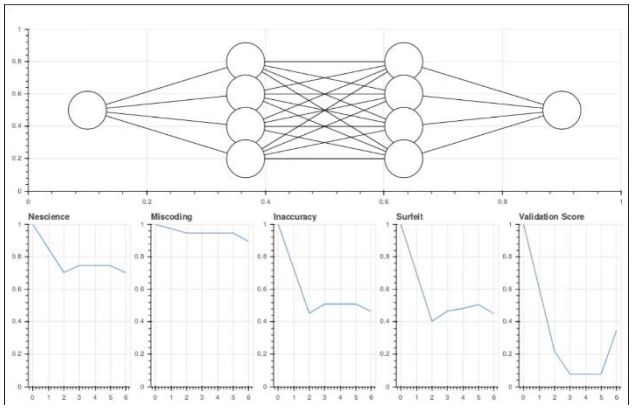

Figure 18 First snapshot of the Automated Neural Network Creation

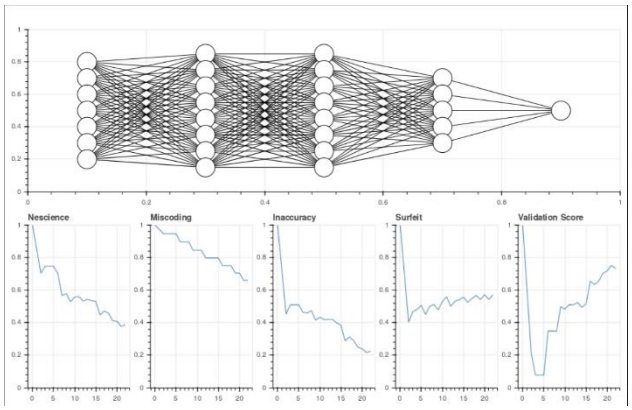

Figure 19 Second Snapshot of the Automated Neural Network Creation

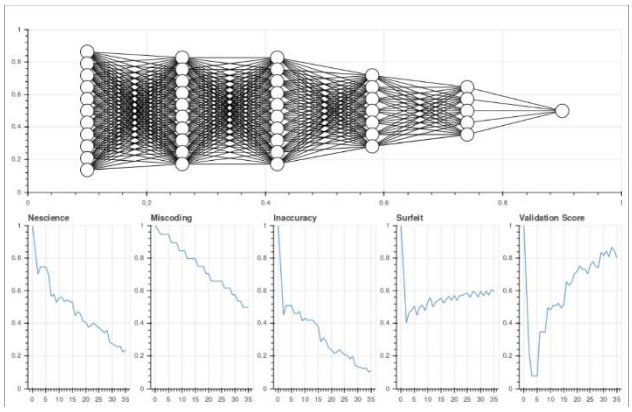

Figure 20. Third Snapshot of the Automated Neural Network Creation

4.4 Auto-Machine Learning applied to Use Case D.1

4.4.1 Feature Selection

The decision tree algorithm has been applied to the analysis of the dataset provided by BT for the Use Case D1 Network Function Virtualization. The dataset corresponds to the monitoring data provided by the esxtop (VMware's ESXi) tool running in batch mode. The dataset is composed by more than twenty thousand metrics monitored (CPU, memory, network, hardware, OS, virtual machines, ...). The problem at hand is to identify which attributes are more relevant to characterize the CPU load of a server from this huge collection of possible metrics (feature selection).

Table 9 shows the list of the attributes that have been identified as relevant by our nescience decision tree algorithm in order to predict/explain the CPU load of a server in a given interval, given the information measured in the previous interval:

Table 9 Relevant Features According to Nescience

Metric
Power consumption
Free Memory
Average CPU load

The CART algorithm produces a tree with a depth of eight levels, that is, 256 attributes. Since what we are interested in is to identify the most relevant attributes, the tree has been pruned to a depth of tree levels in order to select the seven most relevant features. Table 10 shows the attributes identified by the (pruned) CART algorithm.

Table 10 Relevant Features According to CART

Metric
Total CPU
User CPU
Memory in use
System CPU
Packets received
Power usage in watts
Total virtual CPU

The Nescience algorithm prefers to replace the collection of instant attributes Total CPU – User CPU – System CPU – Total Virutal CPU identified by CART by the single attribute Average CPU. Both algorithms identify Memory (in use or free) and Power consumption as two relevant attributes. CART also suggest to use the Network activity (packets received) as a predictor, something the Nescience algorithm does not consider as relevant.

4.4.2 Neural Networks

The automated neural networks architecture search has been applied to the analysis of the dataset provided by BT for the Use Case D1 Network Function Virtualization. The dataset corresponds to the monitoring data provided by the esxtop (VMware's ESXi) tool running in batch mode. The dataset is composed by more than twenty thousand metrics monitored. Our target variable is to predict the average CPU load for the next minute.

After applying our AutoML tool to this dataset, the algorithm concluded that only ten of these twenty thousand metrics are needed in order to predict the average CPU load. Table 11 shows the predictive metrics selected by the algorithm.

Moreover, the optimal architecture found by the algorithm was a shallow neural network with only two layers of size three. This shallow neural network is able to achieve an accuracy of more than 70% predicting future workloads.

Table 11 Features Used in the Neural Network

Metrics
Physical Cpu Load - Cpu Load (5 Minute Avg)
Physical Cpu(_Total) - % Processor Time
Physical Cpu(_Total) - % Util Time
Memory - Kernel Mbytes
Memory - Free Mbytes
Numa Node(0) - Free Mbytes
Numa Node(1) - Free Mbytes
Power - Power Usage Now Watts
Vcpu(2:system:32794:netCoalesce2World) - % Used
Vcpu(2:system:32794:netCoalesce2World) - % Run

5 Conclusion

The collection of this deliverables (Casari and Domaschka 2018) provide all those elements needed in the RECAP project from the point of view of data acquisition, data storage, and data analytics. The main goal of both deliverables is to provide a sufficient level of detail about the data processing and modelling pipeline, so that software developers can write RECAP data solutions without any additional information.

This document extends over (Casari and Domaschka 2018) and covers the following critical elements for a successful data analytics part in the project:

v) An updated version of metrics being collected from Use Cases.
vi) The insights gathered during the exploratory data analysis of Use Cases.
vii) A detailed methodology of how to proceed in the analysis of new scenarios outside the RECAP project.
viii) A description of the research performed in the area of auto-machine learning, and its application to RECAP Use Cases.

Given the descriptions, decisions and results included in this document, not only software developers could finish the developments of the tasks that are part of WP5, but also the planned developments for the work packages WP6, WP7 and WP8 that are related to data processing and data analysis (e.g. predictive models building).

References

Cai, H. C. „Efficient architecture search by network transformation.“ *AAAI18.* 2018.

Casari, Paolo, and Jörg Domaschka. "Initial Data Acquisition and Analytics Models." RECAP Project Deliverable, 2018.

Domaschka, Jörg, and Manuel Noya. “Initial requirements.” RECAP Project Deliverable, 2017.

Domaschka, Jörg, et al. „Final Requirements and Validation Plan.“ RECAP Deliverable, 2018.

Domaschka, Jörg, et al. „Initial Integrated Prototype.“ RECAP Deliverable, 2019.

Domaschka, Jörg, Frank Griesinger, Mark Leznik, Rafael Garcia, and Radhika Loomba. “Initial System Architecture and Integration.” RECAP Project Deliverable, 2018.

Domaschka, Jörg, Frank Griesinger, Peter Willis, and Tobias Sundqvist. "Initial Testbed Configuration." RECAP Project Deliverable, 2017.

Domingos, P. *The Master Algorithm.* Basic Books, 2015.

Forsman, Johan, et al. „Iniital Quality of Service Models.“ RECAP Project Deliverable, 2018.

Hinton, G., Interview geführt von A. Ng. *Heroes of Artificial Intelligence* (2017).

Hu, H. J. *Auto-Keras: Efficient Neural Architecture Search with Network Morphism.* Arxiv, 2018.

Le Duc, Thang, and Per-Olov Östberg. “Initial Workload, Load Propagation and Application Models.” RECAP Project Deliverable, 2018.

Leiva, R. A. Garcia. *The Mathematics of the Unknown, with Applications to Artificial Intelligence.* Author, 2019.

Loomba, Radhika, et al. „Final Infrastructure Modelling and Resource Mapping.“ RECAP Deliverable, 2019.

Loomba, Radhika, et al. „Initial Infrastructure Orchestration and Optimisation.“ RECAP Project Deliverable, 2018.

RECAP Consortium. „Artificial Data Traces and Workload Generator Models.“ RECAP Project Deliverable, 2019.

RECAP Consortium. „Final Infrastructure Orchestration and Optimization.“ RECAP Deliverable, 2019.

RECAP Consortium. „Final Quality of Service Metrics and Models.“ RECAP Deliverable, 2019.

RECAP Consortium. „Final System Architecture and Integration.“ RECAP Project Deliverable, 2019.

RECAP Consortium. „Final Workload, Load Propagraion, and Application Models.“ RECAP Deliverable, 2019.

RECAP Consortium. „Improved Testbed Configuration.“ RECAP Deliverable, 2019.

RECAP Consortium. „Methods and Frameworks for Data Visualisation.“ RECAP Project Deliverable, 2019.

Soare, R. I. *Turing Computability.* Springer, 2016.

Thorton, C. „Auto-WEKA: Combined Selection and Hyperparameter Optimization of Classification Algorithms.“ *KDD13.* 2013.

Vitanyi, M. Li & P. *An Introduction to Kolmogorov Complexity and its Applications.* Springer, 2008.

6 Annex

6.1 Monitoring Metrics per Layer

6.1.1 Metrics on the Physical Layer

Table 12 shows the metrics gathered for the physical layer including a brief description. The table is split into seven metric sets (cpu, diskio, filesystem, memory, switch, vms, and vm). Each of the metric sets contain several detailed metrics. The four metric sets `host.cpu`, `host.diskio`, `host.memory`, `host.filesystem` capture the usage and utilisation of basic system resources (cpu, block devices, memory, file systems). `host.vms` gives information about the virtual machines running on a host and the metrics from the `host.vm` metric set detail the resource consumption per virtual machine.

Table 12 Metrics used on physical layer

metricset	metrics	description
host.cpu	cores	Number of cores available on the host.
	Idle	The percentage of CPU time spent in user space, 100% equals 1 core.
	iowait	The percentage of CPU time spent in wait on disk.
	Irq	The percentage of CPU time spent servicing hardware interrupts.
	nice	The percentage of CPU time spent on low-priority processes.
	softirq	The percentage of CPU time spent on servicing software interrupts.
	steal	The percentage of CPU time spent in involuntary wait by the virtual CPUs while the hypervisor was servicing another processor.
	system	The percentage of CPU time spent in kernel spaces.
	user	The percentage of CPU time spent in user space.
	total	The percentage of CPU time spent in non-idle state
host.diskio	read_bytes read_count	The number of bytes read/read operations completed successfully.
	read_time	The total time spent on by all reads
	write_bytes write_count	The number of bytes written/write operations completed successfully.
	write_time	The total time spent on all writes.
host.filesystem	available space	The disk space available to users in bytes.
	file count	The total number of files in the file system.
	free files	The number of free file nodes in the file system.
	free space	The disk space available in bytes.
	total space	The total disk space in bytes.
	used space	The used disk space in bytes.
host.memory	actual free	The actual free memory
	actual used	The actual used memory (difference between available and used)
	free	Total free memory in bytes.
	total	Total available memory in bytes.
	used	Used memory in bytes.
	swap_free swap_total swap_used	Total, used, and free swap space.
host.network	in_bytes in_packets	Number of bytes/packets received.
	in_dropped	The number of dropped incoming packets.
	in_errors	The number of errors while receiving.
	out_bytes out_packets	Number of bytes/packets sent.
	out_dropped	The number of dropped outgoing packets.
	out_errors	The number of errors when sending.
host.vms	vmcount	The number of virtual machines running on that host.
	vmlist	An array of uuids of virtual machines running on that host.
host.vm	cpu total	The percentage of CPU time spent by the process since the last update.
	disk_blkio_bytes	Total number of bytes transferred to and from all block devices by this vm.

	disk_blkio_ios	Total number of I/O operations to and from all block devices performend by this vm.
	memory_limit	The maximum amount of user memory in bytes (including file cache) the vm is allowed to use.
	memory_rss	The amount of memory the vm uses in the main memory
	memory_share	The shared memory the vm uses
	memory_size	The total virtual memory the vm has.
host.ipmi	power_status	the current state of the host (on/off)
	power_consumption	the current power consumption in watts
host.flow	bytes_send	number of bytes sent for a dedicated flow
	bytes_received	number of bytes received for a dedicated flow
switch.flow	bytes_send	number of bytes sent for a dedicated flow
	bytes_received	number of bytes received for a dedicated flow
switch.network	in_bytes in_packets	Number of bytes/packets received.
	in_dropped	The number of dropped incoming packets.
	in_errors	The number of errors while receiving.
	out_bytes out_packets	Number of bytes/packets sent.
	out_dropped	The number of dropped outgoing packets.
	out_errors	The number of errors when sending.

6.1.2 Metrics on the Virtual Layer

The metrics gathered on virtual machine level (i.e. captured from within a virtual machine) are summarised in Table 13. Metric sets on that layer start with `vm`. Basically, we use the same metrics for the virtual layer as for the physical host. Yet, we ignore those that do not make sense on that layer such as `cpu.steal`. In addition to resource consumption, information about available containers is collected just as resource utilisation per container.

Table 13 Metrics used on virtual layer

Metricset	Metrics	description
vm.cpu	cores, idle, iowait, irq, metric layer, nice, softirq, steal, system, user,	cf. Table 12
vm.diskio	read_bytes, read_count, read_time write_bytes, write_count, write_time	cf. Table 12
vm.filesystem	available_space, free_space, total_space, used_space file count, free files,	cf. Table 12
vm.memory	actual_free, actual_used, free, swap_free, swap_total, swap_used, x	cf. Table 12
vm.network	in_bytes, in_dropped, in_errors, in_packets, out_bytes, out_errors, out_dropped, out_packets	cf. Table 12
vm.containers	container_count, container_list	cf. Table 12
vm.container	cpu total, disk_blkio_bytes, disk_blkio_ios, memory_limit, memory_rss, memory_share, memory_size	cf. Table 12
vm.flow	bytes_send, bytes_received	cf. Table 12
vswitch.flow	bytes_send, bytes_received	cf. Table 12

6.1.3 Metrics on the Container Layer

On the container level, we collect the very same metric sets and metrics as for the virtual layer (cpu, diskio, memory, filesystem, diskio, and network). The names of the metric sets start with `container.` instead of `vm.` The `vswitch` metrics are not used on container level and there is no further information about contained elements available.

6.1.4 Metric Attributes: Tagging

So far, we have presented which metrics exist in the system per layer. Yet, with the information provided so far, it is not possible to distinguish data from different sources. For instance, the total cpu load (`host.cpu.total`) from Host A cannot be distinguished from the total cpu load from Host B. Metric attributes supply this capability. Using them, metric data can be grouped and correlated.

Table 14 Supported metric attributes per metric set

Metricset	metric attribute	description
*	metric layer	physical
	timestamp	2017-09-01 03:19:58
host.*	datacentre location	core
	hostname	computenode01
	infrastructure provider	UULM
host.diskio.*	devicename	sda
host.filesystem.*	devicename	sda1
	filesystem type	ext4
	mountpoint	/mount/vda
host.network.*	devicename	eth0
host.vm.*	vm id	16990e8c-e4fa-41a0-87ac-60f8f9db3440
vm.*	cloud id	UULM
	cloud region	core
	vm id,	16990e8c-e4fa-41a0-87ac-60f8f9db3440
vm.container.*	container uuid	3ecf9396961b5e5c42dc1bd119e7e22ccb b86a50aaf25fd447aeb63612f22261
vm.diskio.*	devicename,	sda
vm.filesystem.*	devicename,	sda1
	filesystem type	zfs
	mountpoint	/
container.*	container uuid	3ecf9396961b5e5c42dc1bd119e7e22ccb b86a50aaf25fd447aeb63612f22261
app.*	containment id	<vm id>
	containment type	virtual machine
	application instance id	12345
	component instance id	67899
.flow.	source ip	10.10.1.1
	source port	11211
	destination ip	134.60.64.111
	destination port	80
	protocol	tcp
switch.*	cf. host.*	

Table 14 list the metric sets that are currently used and supported by the RECAP monitoring sub system. All metrics in all metric sets are tagged with the timestamp and the metric layer (physical, virtual, container, application). All metrics on the physical layer are further tagged with the data centre location, the name of the physical host, and the name of the infrastructure provider. Metrics on the virtual layer are enriched with information about the cloud they are running in, the current region they reside in, and their respective identifier. Similarly, container metrics contain information about the container identifier. On all levels, specific attributes are added if required by the metric. For instance, `devicename` helps distinguish network interfaces on physical hosts when it comes to network traffic.

For application metrics tagging needs to fulfil two orthogonal tasks. First, the tags put on a measurement need to enable that different instances of the same application can be distinguished from each other (e.g. Wordpress installation for customer A and customer B) and further that different instances of an application component, e.g. a scaled out application server, can be

distinguished from each other. Hence, all application metrics are tagged with an application instance identifier and a component instance identifier. These are automatically assigned by the platform and added by the RECAP data dispatcher (Domaschka, Griesinger and Leznik, et al. 2018). If needed, application owners can provide further tags for their metrics.

Second, tagging needs to enable that the value of an application metric can be mapped to the metrics representing the utilisation of the hardware resource. Hence, for each component instance, it needs to be known on what physical resource it was running. Therefore, all application metrics are tagged with the type and identifier of the containing entity (e.g. virtual machine or container).

6.2 Use Case Coverage

6.2.1 System Metrics per Use Case

	UULM testbed	Use Case A	Use Case B	Use Case C	Use Case D.1	Use Case D.2
host.cpu.cores	✓	✓		✓	✓	✓
host.cpu.idle	✓	✓		✓	✓	✓
host.cpu.iowait	✓	✓		✓	✓	✓
host.cpu.irq	✓	✓		✓	✓	
host.cpu.nice	✓	✓		✓	✓	
host.cpu.softirq	✓	✓		✓	✓	✓
host.cpu.steal	✓	✓		✓	✓	
host.cpu.system	✓	✓		✓	✓	
host.cpu.user	✓	✓		✓	✓	
host.diskio.io	✓	✓		✓	✓	
host.diskio.read_bytes	✓	✓		✓	✓	
host.diskio.read_count	✓	✓		✓	✓	
host.diskio.read_time	✓	✓		✓	✓	
host.diskio.write_bytes	✓	✓		✓	✓	
host.diskio.write_count	✓	✓		✓	✓	
host.diskio.write_time	✓	✓		✓	✓	
host.filesystem.available_space	✓	✓		✓	✓	
host.filesystem.file_count	✓	✓		✓	✓	
host.filesystem.free_files	✓	✓		✓	✓	
host.filesystem.free_space	✓	✓		✓	✓	
host.filesystem.total_space	✓	✓		✓	✓	
host.filesystem.used_space	✓	✓		✓	✓	
host.memory.actual_free	✓	✓		✓	✓	✓
host.memory.actual_used	✓	✓		✓	✓	✓
host.memory.free	✓	✓		✓	✓	
host.memory.swap_free	✓	✓		✓	✓	
host.memory.swap_total	✓	✓		✓	✓	
host.memory.swap_used	✓	✓		✓	✓	✓
host.memory.total	✓	✓		✓	✓	✓
host.memory.used	✓	✓		✓	✓	

host.network.in_bytes	✓	✓		✓	✓	✓
host.network.in_dropped	✓	✓		✓	✓	
host.network.in_errors	✓	✓		✓	✓	
host.network.in_packets	✓	✓		✓	✓	✓
host.network.out_bytes	✓	✓		✓	✓	✓
host.network.out_dropped	✓	✓		✓	✓	
host.network.out_errors	✓	✓		✓	✓	
host.network.out_packets	✓	✓		✓	✓	✓
host.vm.cpu_total	✓	✓			✓	
host.vm.disk_blkio_bytes	✓				✓	
host.vm.disk_blkio_ios	✓				✓	
host.vm.memory_limit	✓				✓	
host.vm.memory_rss	✓				✓	
host.vm.memory_share	✓				✓	
host.vm.memory_size	✓				✓	
host.vm.metric_layer	✓		✓		✓	
host.vms.vmcount	✓		✓			
host.vms.vmlist	✓		✓			
vm.cpu.cores	✓		✓			
vm.cpu.idle	✓		✓			
vm.cpu.iowait	✓		✓			
vm.cpu.irq	✓		✓			
vm.cpu.nice	✓		✓			
vm.cpu.softirq	✓		✓			
vm.cpu.steal	✓		✓			
vm.cpu.system	✓		✓			
vm.cpu.user	✓		✓			
vm.diskio.io	✓		✓			
vm.diskio.read_bytes	✓		✓			
vm.diskio.read_count	✓		✓			
vm.diskio.read_time	✓		✓			
vm.diskio.write_bytes	✓		✓			
vm.diskio.write_count	✓		✓			
vm.diskio.write_time	✓		✓			
vm.filesystem.available_space	✓		✓			
vm.filesystem.file_count	✓		✓			
vm.filesystem.free_files	✓		✓			
vm.filesystem.free_space	✓		✓			
vm.filesystem.total_space	✓		✓			
vm.filesystem.used_space	✓		✓			
vm.memory.actual_free	✓		✓			
vm.memory.actual_used	✓		✓			
vm.memory.free	✓		✓			
vm.memory.swap_free	✓		✓			
vm.memory.swap_total	✓		✓			

vm.memory.swap_used	✓		✓			
vm.memory.total	✓		✓			
vm.memory.used	✓		✓			
vm.network.in_bytes	✓	✓	✓			
vm.network.in_dropped	✓	✓				
vm.network.in_errors	✓	✓				
vm.network.in_packets	✓	✓				
vm.network.metric_layer	✓	✓				
vm.network.out_bytes	✓	✓				
vm.network.out_dropped	✓	✓				
vm.network.out_errors	✓	✓				
vm.network.out_packets	✓	✓				

6.2.2 Application Metrics per Use Case

Use Case A	Use Case B: Load Balancer Metrics	Use Case B: Elastic Search Metrics
app.cell_counter.cell_id	app.lkn.apache.bytes_per_req	app.lkn.es.cloud_id
app.cell_counter.counter	app.lkn.apache.bytes_per_sec	app.lkn.es.cloud_region
app.cell_counter.value	app.lkn.apache.req_per_sec	app.lkn.es.clustername
app.cell_timer.high_usec	app.lkn.apache.score_closing	app.lkn.es.fs_available
app.cell_timer.low_usec	app.lkn.apache.score_dns	app.lkn.es.fs_free
app.cell_timer.mean_usec	app.lkn.apache.score_finishing	app.lkn.es.fs_total
app.cell_timer.samples	app.lkn.apache.score_idleclean	app.lkn.es.indices_store_docs_count
app.cell_timer.timer	app.lkn.apache.score_keepalive	app.lkn.es.indices_store_docs_deleted
app.ctrl_meas.llsim_bytes_rec	app.lkn.apache.score_logging	app.lkn.es.indices_store_segments_bytes
app.ctrl_meas.llsim_bytes_sent	app.lkn.apache.score_openslot	app.lkn.es.indices_store_segments_count
app.ctrl_meas.llsim_pkt_rec	app.lkn.apache.score_reading	app.lkn.es.indices_store_size
app.ctrl_meas.llsim_pkt_sent	app.lkn.apache.score_sending	app.lkn.es.jvm_collectors_old_count
app.ctrl_meas.mme_bytes_rec	app.lkn.apache.score_starting	app.lkn.es.jvm_collectors_old_ms
app.ctrl_meas.mme_bytes_sent	app.lkn.apache.score_totoal	app.lkn.es.jvm_collectors_young_count
app.ctrl_meas.mme_pkt_rec	app.lkn.apache.score_waiting	app.lkn.es.jvm_collectors_young_ms
app.ctrl_meas.mme_pkt_sent	app.lkn.apache.server_uptime	app.lkn.es.jvm_memory_old_max
app.ctrlplane_traffic.rxBytes	app.lkn.apache.server_uptime_seconds	app.lkn.es.jvm_memory_old_peak
app.ctrlplane_traffic.rxPackets	app.lkn.apache.total_accesses	app.lkn.es.jvm_memory_old_peakmax
app.ctrlplane_traffic.test_id	app.lkn.apache.total_connections	app.lkn.es.jvm_memory_old_used
app.ctrlplane_traffic.txBytes	app.lkn.apache.total_kbytes	app.lkn.es.jvm_memory_survivor_max
app.ctrlplane_traffic.txPackets	app.lkn.apache.workers_busy	app.lkn.es.jvm_memory_survivor_peak
app.global_counter.counter	app.lkn.apache.workers_idle	app.lkn.es.jvm_memory_survivor_peakmax
app.global_counter.value	app.lkn.apache.log.access.agent	app.lkn.es.jvm_memory_survivor_used

app.latency.avg_latency_microseconds	app.lkn.apache.log.access.auth	app.lkn.es.jvm_memory_young_max
app.latency.max_latency_microseconds	app.lkn.apache.log.access.bytes	app.lkn.es.jvm_memory_young_peak
app.latency.min_latency_microseconds	app.lkn.apache.log.access.clientip	app.lkn.es.jvm_memory_young_peakmax
app.latency.samples	app.lkn.apache.log.access.ident	app.lkn.es.jvm_memory_young_used
app.global_counter.value	app.lkn.apache.log.access.referrer	app.lkn.es.project
app.uesim_user_stats.machine_uuid	app.lkn.apache.log.access.remote_ip	app.lkn.process.application
app.uesim_user_stats.pending_ues	app.lkn.apache.log.access.request	app.lkn.process.cpu_total
app.uesim_user_stats.setup_ues	app.lkn.apache.log.access.response	app.lkn.process.disk_blkio_bytes
app.userplane_traffic.machine_uuid	app.lkn.apache.log.access.source	app.lkn.process.disk_blkio_ios
app.userplane_traffic.rxBytes	app.lkn.apache.log.access.timetakenrequest	app.lkn.process.memory_limit
app.userplane_traffic.rxPackets	app.lkn.apache.log.access.verb	app.lkn.process.memory_rss
app.userplane_traffic.test_id	app.lkn.apache.log.error.clientip	app.lkn.process.memory_share
app.userplane_traffic.txBytes	app.lkn.apache.log.error.clientport	app.lkn.process.memory_size
app.userplane_traffic.txDrops	app.lkn.apache.log.error.errorcode	
app.userplane_traffic.txPackets	app.lkn.apache.log.error.module	
app.citysim.cell_datarate_change.rate_dl	app.lkn.apache.log.error.offset	
app.citysim.cell_datarate_change.rate_ul	app.lkn.apache.log.error.pid	
app.citysim.service_initialized.dl_packet_size	app.lkn.apache.log.error.remote_ip	
app.citysim.service_initialized.dl_pps	app.lkn.apache.log.error.source	
app.citysim.service_initialized.duration	app.lkn.apache.log.error.tid	
app.citysim.service_initialized.ul_packet_size		
app.citysim.service_initialized.ul_pps		