CHAPTER 2

RECAP Data Acquisition and Analytics Methodology

Paolo Casari, Jörg Domaschka, Rafael García Leiva, Thang Le Duc, Mark Leznik, and Linus Närvä

Abstract The collection, analysis, and processing of infrastructure information and telemetry data lie at the very heart of RECAP. This chapter describes the infrastructure for the acquisition and processing of data from applications and systems, and explains the methodology used to derive

P. Casari (✉) • R. García Leiva
IMDEA Networks Institute, Madrid, Spain
e-mail: paolo.casari@imdea.org; rafael.garcia@imdea.org

J. Domaschka • M. Leznik
Institute of Information Resource Management, Ulm University, Ulm, Germany
e-mail: joerg.domaschka@uni-ulm.de; mark.leznik@uni-ulm.de

T. Le Duc
Tieto Product Development Services, Umeå, Sweden
e-mail: thang.leduc@tieto.com

L. Närvä
Tieto Sweden Support Services AB, Karlstad, Sweden
e-mail: linus.narva@tieto.com

T. Lynn et al. (eds.), *Managing Distributed Cloud Applications and Infrastructure*, Palgrave Studies in Digital Business & Enabling Technologies, https://doi.org/10.1007/978-3-030-39863-7_2

statistical and machine learning models from this data. These models are then used to identify relevant features and forecast future values, and thus inform run-time planning, decision making, and optimisation support at both the infrastructure and application levels. We conclude the chapter with an overview of RECAP data visualisation approaches.

Keywords Data analytics • Data acquisition • Machine learning • Application modelling • Infrastructure modelling • Distributed cloud computing • Edge computing

2.1 Introduction

The collection, analysis, and processing of data (e.g., infrastructure information and telemetry) lie at the very heart of RECAP and constitute a crucial part of the entire RECAP system. Data make it possible to train machine learning and data analytics algorithms in the analytics mode; moreover, data provide the basis for run-time planning, decision making, and optimisation support at both the infrastructure and the application levels; finally, they can be used as calibration mechanisms for the RECAP simulators. As such, the data acquisition and analytics methodology comprises (1) data acquisition, defining how to collect data from the RECAP infrastructure and the applications running on top of it, how to store that data, and how to provision it to the various parts of the RECAP ecosystem; and (2) data analytics, defining how to access the data and create usable models from it.

Accordingly, this chapter is structured as follows: Section 2.2 describes the infrastructure for the acquisition and processing of data (both from applications and from systems). This is followed by an overview of the data analytics methodology in Sect. 2.3, including the development of mathematical models to identify relevant features and forecast future values. Section 2.4 provides an overview of visualisation in RECAP.

2.2 Data Acquisition and Storage

Data collection in RECAP serves three purposes: (i) to derive information about the flow of messages (hence, the load in the application layer) and use it to create workload and load transition models; (ii) to derive the impact of the application layer behaviour on resource consumption on the

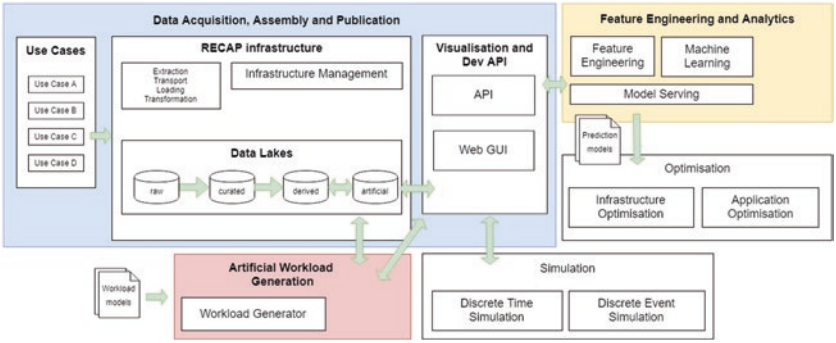

Fig. 2.1 Conceptual overview of data handling in RECAP

physical layer; and (iii) to provide input to simulation and visualisation components.

As shown in Fig. 2.1, RECAP makes use of a central data repository, which serves as the single integration point for all elements of the RECAP ecosystem, and as the primary source of data for other parts of the RECAP platform.

In its databases, the repository stores information about: (i) time series of load metrics, (ii) information about the configuration of the data centre and virtual infrastructure, and (iii) information about the applications running on top of this infrastructure. While (i) is the primary focus of the repository, (ii) and (iii) are additional metadata that enrich the time series data and that help correlate time series of various metrics from different layers of the system. As an example, metadata could help correlate infrastructure metrics, such as CPU usage, with application performance metrics from the application layer, such as worker queue length.

Technically, the data repository cannot be realised as a single entity, as it has to satisfy different requirements from various components. While the data analytics and machine learning functionality in RECAP require access to large chunks of CSV-formatted data, the visualisation component requires the capability to flexibly query for data upon a user request. Finally, other RECAP components require access to a live stream of data: for instance, the optimisers constantly need to look up the current state of the system. In consequence, a polyglot approach to persistence is required, as will be presented in later in this chapter.

2.2.1 Terminology

We now briefly cover the terminology that applies to the RECAP Monitoring Architecture.

2.2.1.1 Metrics and Monitoring

Formally, a **metric** is a function that takes a system as input and yields a scalar as a result. The application of a metric on a particular system is called a **measurement** and the result of the application is called the **value** of that metric. The **unit** of the value depends on the metric.

The **monitoring** process continuously (or periodically) applies metrics to systems and generates a series of timestamped values. This is called a **time series** (of a metric).

In order to distinguish values and time series that belong to the same metric, but come from different systems, we allow values to be further enhanced by **metric properties** (or **tags**). This enables values to be grouped, leading to a time series for that tag.

As an example, the cpu_load metric, when applied to a server, yields the current load of the central processing unit on that server. In order to be able to distinguish values measured from server A from those measured from server B, the value may be tagged with the tag origin that in this example can take the values A and B. In total, this creates three time series: one for A, one for B, and one for both servers.

2.2.1.2 Actors

Based on the context of RECAP and the requirements defined by the project's use case providers, the monitoring infrastructure assumes a cloud-like environment where virtual resources (cloud resources) are made available through a Web-based API.

A **(cloud) operator** or **infrastructure provider** provides the physical resources on which virtual resources run. Physical resources may be geographically distributed, leading to a cloud-edge scenario. This actor is responsible for maintaining the physical set-up and for running the software stack that enables access to the virtual resources. The infrastructure provider is also the actor that operates the RECAP infrastructure. Note that communications service providers, such as telecommunications companies, can also be cloud operators and infrastructure providers.

(Cloud) users access the virtual resources offered by the cloud provider. In Infrastructure-as-a-service clouds, they acquire virtual machines

and virtual networks to operate their applications. This makes them **(application) operators** and therefore also users of RECAP.

Finally, **end users** access the applications provided by the application operator. Usually, they do not care where the application runs, as long as it provides an acceptable quality of service and experience.

2.2.2 Monitoring Layers

Figure 2.2 illustrates the four layers that can be monitored in order to derive insights on application behaviour and load propagation. Not all layers are required for all installations, so the set-up presented here is a superset of the possible set-ups.

The **physical layer** is provided by the infrastructure provider, and contains the hardware used to run all higher layers. Here, monitoring metrics mainly include CPU, RAM, disk, and network consumption at specific points in time. The layout of the physical infrastructure is also important, e.g. which servers share the same network storage or uplink to the Internet. Figure 2.3 shows two data centre locations on the left and right hand sides, each with a router. Both are connected through the Internet. The infrastructure provides RECAP-aware monitoring support for the physical layer and report measurements for the metrics.

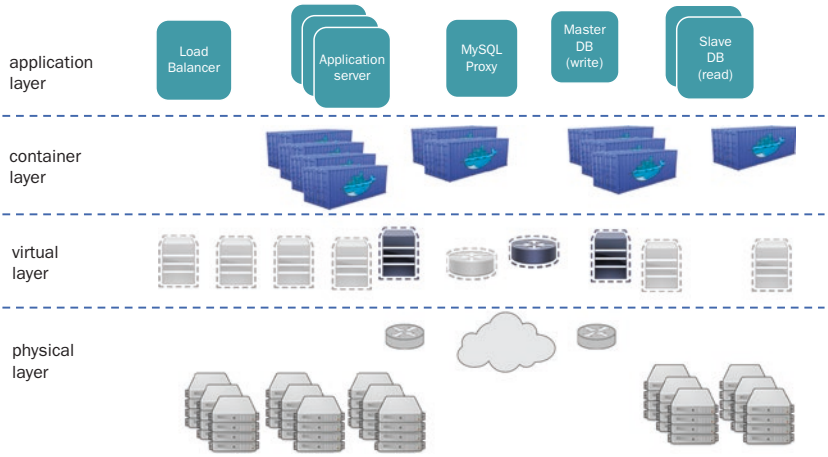

Fig. 2.2 RECAP monitoring layers

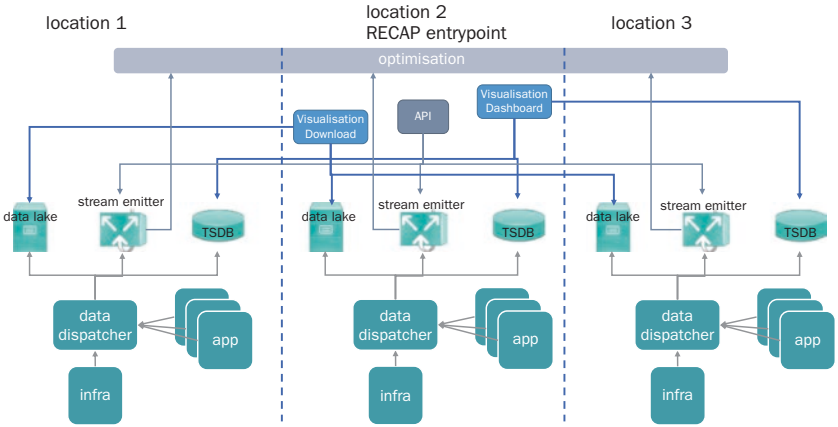

Fig. 2.3 RECAP's distributed monitoring architecture

The **virtual layer** constitutes virtual infrastructure as realised for instance by Infrastructure-as-a-Service (IaaS) clouds. This infrastructure is composed of virtual machines, virtual storage, and virtual networks. It is run by the application operator, which is responsible for the measurements on that layer as well. Similar to the physical layer, metrics mainly include CPU, RAM, disk, and network consumption, but several instances of virtual infrastructure (to be monitored separately) can exist in the virtual layer (cf. different colours in Fig. 2.2). Also, the number of virtual components per virtual infrastructure as well as the number of virtual infrastructures is not fixed, but can dynamically grow and shrink.

On top of the virtual (or physical) layer resides an optional **container layer** such as a Kubernetes[1] cluster or a Rancher Cattle[2] cluster. Basically, the same restrictions and considerations hold for this layer as for the virtual layer. Yet, in contrast to both, the container layer can provide a seamless abstraction and hide the location of different data centres. Whether containers are used is a design choice by application owners or cloud providers: containers can be offered by a cloud provider or be deployed by a user on top of virtual machines.

[1] https://kubernetes.io/
[2] https://github.com/rancher/cattle

At the top resides the **application layer** where application- and component-specific metrics can be applied. These include, for instance, the queue length of load balancers, detailed statistics on the use of databases, and the message throughput of a publish-subscribe system. Application-specific metrics are important conveyors of KPIs or QoS. Although the RECAP monitoring platform cannot define all possible application-level metrics to be captured, it provides a structure to measure and store application-level metrics.

2.2.3 Monitoring Architecture

The RECAP Monitoring Architecture collects and provides the monitoring data from the four layers described earlier to the RECAP simulator, run-time system, and users.

RECAP operators may manage infrastructure spread over several, geographically distributed locations. In each of these sites an edge or core/cloud data centre resides. In order to limit data hauling across data centres, collected data are stored as close to their origin as possible. RECAP's acquisition and retrieval strategy takes these circumstances into account. In the following, we first describe the acquisition and storage architecture per site and then the overall architecture spanning different sites.

2.2.3.1 Single Site Monitoring Set-up

This section describes the monitoring set-up for each site in a RECAP managed infrastructure. Each of the sites can run in isolation and is not affected by traffic and load on other sites. The per-site architecture consists of monitoring probes on the physical layer and on the higher layers of the software stack. It also involves a data dispatcher that filters the incoming data and relays it to three different data sinks: a data lake, a time series database, and a data stream emitter.

Components

Probe: Probes convey monitoring data out of monitored systems. Different probes may be necessary for different metrics, even though most probes will perform measurements for multiple metrics. Each probe may emit data in a different format and at different time intervals. In addition to a timestamp and value, an emitted data item contains metric properties to identify the source and scope of the data point. While it is the responsibility of the infrastructure operator to provide probes for the physical

layer, the cloud user shall provide the necessary metrics for higher layers of the stack. All probes directly or indirectly send their data to the data dispatcher.

Data dispatcher: The data dispatcher is a site's central data monitoring integration point. It receives and normalises all probe data and sends them to the data sinks. Normalisation depends on the probes: an individual transformation is needed per metric and per probe type. The dispatcher adds site information and similar attributes to collected data.

The data normalised by the dispatcher are then put in the sinks. In RECAP, different data post-processing demands exist with regard to the monitoring subsystem. Feature engineering and data analytics in RECAP operate on large data sets which need to be processed offline in dedicated servers. Visualisation works on smaller datasets, but requires high flexibility in data provisioning. Finally, optimisers require a snapshot that represents the most recent state of the managed infrastructure. Therefore, RECAP applies three different types of sinks:

- **Data lake sink**: accumulates large amounts of monitoring data in a durable storage for a long time using a compact representation. This data is the basis for data analytics and machine learning.
- **Time series database sink (TSDB)**: stores monitoring data in time series. Through an underlying indexed search engine, it supports live queries of current and past data. It is the primary data source for the visualisation components.
- **Stream emitter sink**: relays a configurable subset of live monitoring data to other parts of the RECAP infrastructure. It is the primary data source for the application optimisation and infrastructure optimisation engines, which decide on the metrics of interest and any pre-processing (e.g. smoothing) to be applied.

A distinctive trait of RECAP is the "separation of concerns" between application and infrastructure optimisation procedures. This makes it possible to accommodate the (often contrasting) objectives, costs and constraints of both application and infrastructure providers, and to harmonise them as far as possible within the RECAP framework.

Practical Considerations

The dispatcher and all sinks are stateless and can be scaled to serve large hardware installations, large amounts of users, and high volumes of data.

Neither the architecture nor its implementation puts any restrictions on where dispatcher and sinks can be run. Yet, in order to ensure a correct interplay of the acquisition and storage components with other parts of the RECAP infrastructure, the following constraints have to be considered:

1. All kinds of probes at a site need to be able to connect to the dispatcher of that site either directly or indirectly.
2. Both the TSDB and data lake sink need to be accessible from the API component described later in Sect. 2.3.2.
3. The stream emitter sink needs to communicate to other data centres, and particularly to the optimisation subsystem.

2.2.3.2 Cross-site Monitoring Set-up

As RECAP provides cross data centre resource and application management, individual sites must be spanned to achieve a holistic view of the system. The Monitoring Architecture achieves this by introducing a RECAP entrypoint that may also be bound to a DNS name in order to ease access, and includes a load balancer to point to the various sites managed by this instance of RECAP.

Figure 2.3 provides an overview of the overall architecture of the monitoring infrastructure spanning sites. It shows three locations, one of which functions as the RECAP entrypoint. Besides the local entities from Sect. 2.3.1, it shows the visualisation endpoints that offer a dashboard with usage graphs as well as a GUI for bulk download of data from the data lakes. The more generic API entity component serves as an integration point for other RECAP components. In particular, the optimiser can use it to configure the stream emitter sink which provides input to the optimisation cycle or in order to access time series data from the TSDB.

As detailed earlier, the data lake sink is instantiated per site and can be a distributed component that compresses and stores raw monitoring data. Its primary purpose is to serve files for bulk download. As this storage form is resource hungry, the monitoring infrastructure (1) switches off persisting raw metrics on a per-site basis (this is beneficial if the site cannot store larger amounts of data or no later data analysis shall be performed), and (2) deletes or moves away data older than a certain age. While this creates cross-site load, the fact that data is sent filtered and compressed requires much less bandwidth than uncompressed probe data.

2.2.4 Data Structure for Storage

This section introduces the actual data that is collected on the four layers. We do not discuss the data sent by the various probes as RECAP does not enforce the use of specific probes. Instead, it assumes that the dispatcher performs probe-specific normalisation.

2.2.4.1 Metrics on the Physical Layer

The metrics gathered for the physical layer are split into seven metric sets (cpu, diskio, filesystem, memory, vms, and vm). Each of the metric sets contain several detailed metrics. The four metric sets host.cpu, host.diskio, host.memory, and host.filesystem capture the detailed usage and utilisation of basic system resources (cpu, block devices, memory, file systems). Instead, 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.

We measure the resource consumption of a virtual machine from the host to avoid the misinterpretation of numbers seen from inside the virtual infrastructure. For example, a 100% CPU load seen inside the virtual machine may not mean that the machine uses a full physical core. The mapping of how many physical cores are represented by one virtual core for this particular virtual machine is subject to the CPU scheduler on the host/hypervisor and is heavily influenced by the overbooking factor of the physical server. Hence, the physical layer needs to report on the physical resource usage per virtual machine.

2.2.4.2 Metrics on the Virtual Layer

The metrics gathered at the virtual machine level (i.e. captured from within a virtual machine) start with vm. and are basically the same as the physical host except, for example, cpu.steal. In addition to resource consumption, information about available containers is collected in the same way as resource utilisation per container.

2.2.4.3 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, and network). The names of the metric sets start with container. instead of vm.

2.2.4.4 Metrics on the Application Layer

Applications differ and so do the metrics that can and need to be collected from them. In particular, the measurement gathering methods depend on the application and its software components. Hence, the data format and content for application metrics cannot be fixed in advance, and metric collection must be part of the application lifecycle management.

A generic naming convention for application-level metrics is adopted in RECAP with the format app.<app name>.<comp name>.<metric name>, which includes the (system-wide unique) application name, the component name (unique per application), and the metric name.

2.2.4.5 Metric Attributes: Tagging

So far, we have presented metrics per layer. Yet, with the information provided so far, it is not possible to distinguish data from different sources. This is achieved via metric attributes that also enable data grouping and correlation. For example, all metrics are tagged with the timestamp and the layer (physical, virtual, container, application). All physical layer metrics 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.

For application metrics, tagging needs to fulfil two orthogonal tasks: to distinguish different instances of the same application (e.g. WordPress installation for customer A and customer B), and to distinguish different instances of an application component, e.g. a scaled out application server. Hence, all application metrics are tagged with an application instance identifier and a component instance identifier, both automatically assigned by the platform and added by the RECAP data dispatcher system. If needed, application owners can provide further tags.

Second, tagging needs to convey on what physical resource an application or component was running. Therefore, all application metrics are tagged with the type and identifier of the containing entity (e.g. virtual machine or container).

2.2.5 *Implementation Technology*

The implementation technology for the monitoring system chosen for RECAP is largely based on experience gained from the FP7 CACTOS project (Groenda et al. 2016) using an OpenStack testbed and production system (bwCloud[3]) and from the Horizon 2020 Melodic project (Melodic 2019). Where possible, all technical building blocks were components where technology was available under an open source and/or a commercial licence. Finally, no chosen component makes any assumptions on the technology of the other components, facilitating replacements and upgrades.

The **data dispatcher** is realised through Elastic Logstash[4] which offers pipelines for receiving, processing, and dispatching a wide range of monitoring data. It comes with an extensive list of input plugins, including software to accept TCP/UDP network traffic with JSON payload. Output plugins range from time series databases and overwrites to the file system to sending message streams through publish-subscribe platforms such as Apache Kafka. Filters are provided for data curation and transformation.

The **time series database sink** is realised via an InfluxDB instance, which supports both groups of metrics and metric attributes/tags. It also supports continuous queries and data aggregation, and integrates well with Grafana, an open source metric analytics and visualisation suite commonly used for visualising time series data for infrastructure and application analytics.[5]

The **stream emitter sink** is realised by the Apache Kafka[6] publish-subscribe system, due to its wide adoption and well-known scalability.

The **data lake sink** is based on CSV files stored in a compressed format.

Probes: The entire monitoring subsystem is independent from the specific monitoring technology. This allows RECAP to integrate into existing installations. Consequently, running RECAP does not require operators to perform major updates on their infrastructure. Therefore, the mapping from the data collected by the probes to the metrics schema must be implemented for the dispatcher per probe type. Based on the RECAP testbeds, a set of mapping rules have been implemented for specific probes. In particular, this is the case for the Elastic Metric Beat metric collector to

[3] https://www.bw-cloud.org/
[4] https://www.elastic.co/products/logstash
[5] https://grafana.com
[6] https://kafka.apache.org/

collect metrics on the physical and virtual layer, for Intel's SNAP collector to collect metrics on the virtual container and application layer, and for a VMware vSphere[7] collector.[8]

2.3 Data Analytics and Modelling

2.3.1 Data Analytics Methodology

In this section, we describe the RECAP methodology for the analysis of datasets and the development of machine learning algorithms to support the application of RECAP's results to new problems related to optimal resource allocation and capacity planning. The methodology is composed of five main steps as outlined in Fig. 2.4.

2.3.1.1 Step 1: Problem Definition and Data Assembling

The initial steps are to *identify the problem* to be solved and *the available data* that can help solve the problem through machine learning. In RECAP we merged these steps into a single task due to their high interdependence. If the available datasets are insufficient, we have to change our expectations about the problem or find additional data. As an alternative, we later explain how to enrich existing datasets with synthetic datasets mimicking the same workload data collected from RECAP Use Cases.

2.3.1.2 Step 2: Metric for the Evaluation of the Results

Selecting the metric to evaluate the results of our model is critical, since that metric is exactly what the training algorithm will optimise. If the output of the model is a continuous variable, the Root Mean-Square Error (RMSE) is a typical choice. In the case of a categorical response, typical metrics are accuracy, or the area under the receiver operating characteristic (ROC) curve (AUC).

There are multiple standard techniques to evaluate the performance of a machine learning model and detect issues, such as overfitting, early. These include train/test splits of the dataset, N-fold cross-validation, and bootstrapping. In RECAP, we use train/test splits for the early model prototyping, and apply a cross-validation to the final models before

[7] http://www.virten.net/2015/05/vsphere-6-0-performance-counter-description/
[8] https://github.com/Oxalide/vsphere-influxdb-go

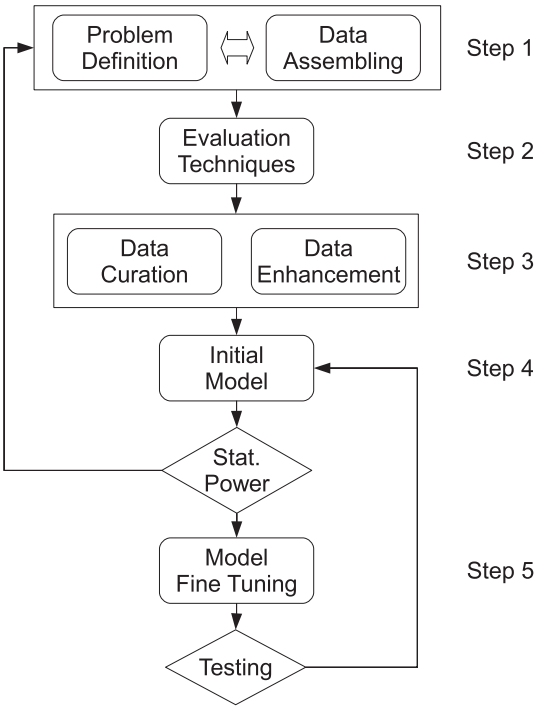

Fig. 2.4 A summary of the main steps of the methodology for exploratory data analysis of new datasets

production. Techniques to avoid cross-validation altogether have also been investigated as a promising research direction (García Leiva et al. 2019).

2.3.1.3 Step 3: Data Curation and Enhancement

A data curation process to remove errors and anomalies and fix missing data is an important preparatory step before training a model. A visualisation of the dataset and a descriptive analysis provides valuable information about the quality of the data being used in the project. Outlier detection or the identification of 'Not Available' values could be applied as well. It might also be necessary to enhance the data by deriving new features based on those that already exist. This data enrichment could significantly improve the predictive capabilities of models.

2.3.1.4 Step 4: Model Development

Identifying the best model is often a daunting task in the presence of all possible alternatives. 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. An approach to speed up the selection of the right family of models is to test the statistical power of the machine learning techniques. This test consists of performing fast training of the model, perhaps with a data subset, and in checking if the model has better predictive capabilities than random guessing. Any family of models with no predictive power should be discarded.

2.3.1.5 Step 5: Regularisation and Hyperparameter Selection

The final step of the methodology is to tune the model's hyperparameters, whose values must be set before the learning process begins. Hyperparameter optimisation makes it possible to obtain the best predictive capabilities from a machine learning model, at the price of a higher risk of overfitting. Once hyperparameters have been optimised, the model can be applied to test data never used during training and validation. A clear sign of overfitting is then a divergence between test performance and validation performance.

2.3.2 Exploratory Data Analysis

Descriptive statistics are metrics that quantitatively describe, characterise, and summarise the features of a data set. Even when data analysis draws its main conclusions using inferential statistics and predictive analytics, descriptive statistics can be used to provide a summary of the types of data involved in the use cases, and inform future inference and prediction steps.

Exploratory data analysis (EDA) is used to understand data beyond formal modelling or hypothesis testing. EDA is useful to check assumptions required for model fitting, to handle missing values, and understand the required variable transformations. Figure 2.5 shows an example of a decomposition of a time series in order to visually identify trends and possible cycles. The top panel visualises the original time series. From this data, we extract a trend (second panel), a seasonal component showing clear cyclic behaviour (third panel), and a residual behaviour not explained by trend and seasonal components (bottom panel). These exploratory steps are helpful to inform the choice of time series prediction techniques.

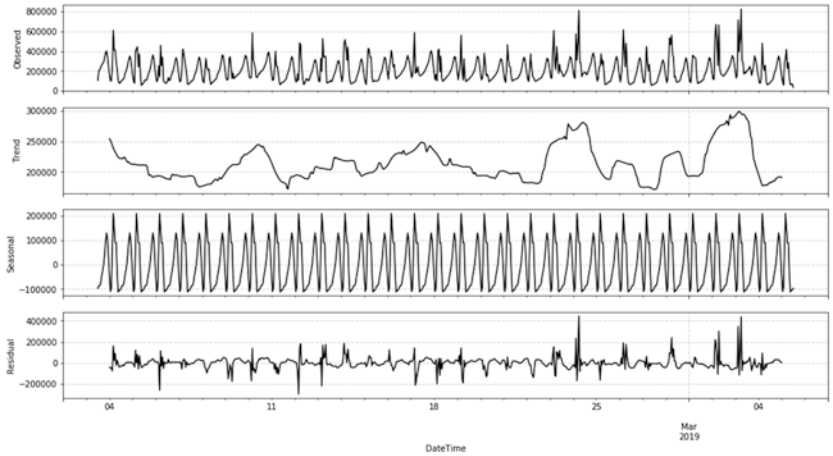

Fig. 2.5 Decomposition of received traffic at a cache

2.3.3 Workload Prediction

After a careful survey of the available literature in the field (Le Duc et al. 2019), three techniques were considered for the specific task of workload prediction—probabilistic models, regression-based models, and machine learning models.

2.3.3.1 Probabilistic Models

Probabilistic models are powerful tools to explain datasets, and are widely used in statistics, traffic engineering, simulations, etc. To facilitate workload prediction in RECAP, we attempt to fit several probability density functions to our datasets on a per-use-case basis. Parameter fitting is obtained through Maximum-Likelihood Estimation, and the resulting models are compared through the Kolmogorov-Smirnov test. The best fitting model is finally chosen (an example for cache content pulling is provided in Fig. 2.6).

2.3.3.2 Regression-based Models

Regression-based models are often simple and robust in generating predictions, and thus particularly suitable for offline modelling and prediction tasks. In RECAP, we consider autoregressive integrated moving average (ARIMA) models, which are composed of three parts. The AR part relies

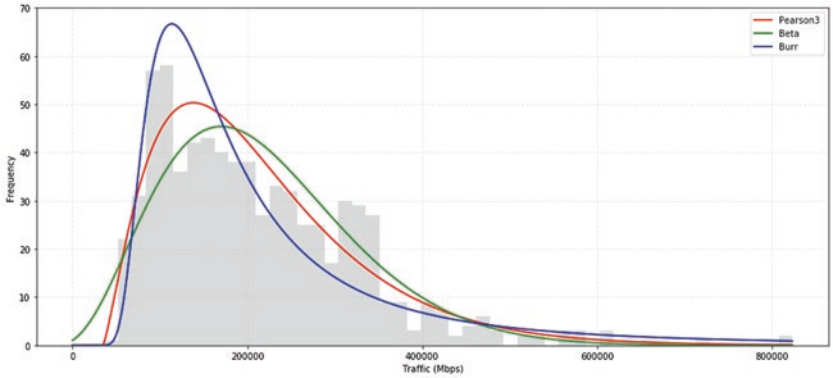

Fig. 2.6 Statistical distributions fitted to records of data sizes of pulled cache content

on the lagged values of the variable of interest; the MA part is actually a linear combination of error terms whose values occurred in the past; and the I part (for "integrated") indicates that the data values have been replaced with the difference between their values and previous values. We also extend ARIMA models with seasonal components (SARIMA).

2.3.3.3 Machine-Learning-based Models

In order to facilitate fast online workload predictions in RECAP, we consider the Online Sequential Extreme Learning Machine (OS-ELM), which enables the generation of workload models and predictions online, and can flexibly handle workload changes. OS-ELM is an efficient technique for online time series modelling and prediction due to its accuracy comparable to batch training methods and to its extremely fast generation of predictions (Huang et al. 2005; Liang and Huang 2006). It accepts input data either sample-by-sample or through varying- or fixed-size data chunks.

Different from other learning methods (e.g. single hidden layer feed-forward neural networks), OS-ELM randomly initialises input weights and updates output weights using the recursive least squares method. This makes OS-ELM adapt quickly to new input patterns, and results into a better prediction performance than other online learning algorithms (Park and Kim 2017).

2.3.4 Artificial Workload Generation

RECAP is interested in the availability of datasets describing the evolution of workloads of servers and services (applications), so that stochastic models can be trained to forecast future workloads. However, for reasons including commercial sensitivity and privacy, such datasets may be insufficient for research tasks. This issue can be circumvented by generating synthetic datasets that preserve the statistical properties of the datasets collected from real infrastructures.

Here, we briefly introduce the mathematical models used to generate artificial workload traces in RECAP. Relevant references are provided for the interested reader.

2.3.4.1 Structural Models-based Workload Generation

Structural time series models are a family of stochastic models for time series that includes and generalises modelling techniques, including ARIMA or SARIMA models (Harvey 1989). A structural time series model expresses an observed time series as the sum of simpler components:

$$f(t) = f_1(t) + f_2(t) + \ldots + f_n(t) + \epsilon$$

where ϵ is a white error term following a normal distribution of mean 0 and variance σ^2.

For example, one component might encode a linear trend, a cycle, or a dependence of previous values. Structural time series models identify and encode assumptions about the processes that have generated the original data. In this way, they make it possible to generate artificial data traces that have the same statistical properties as the original datasets. The application of a structural time series model to requests coming to a search engine web server is shown in Fig. 2.7. We observe that predicted data (light grey) mimic well the general characteristics of ground truth (dark grey).

2.3.4.2 GAN-based Workload Generation

Synthetic data generation using Generative Adversarial Networks (GAN) has recently gained popularity. A GAN is based on a combination of two neural networks, a discriminator (D) and a competing generator network (G). In the training phase, D is trained to distinguish real data from generated data. In parallel, G is trained to fool D by producing better and better fake data that D will eventually accept.

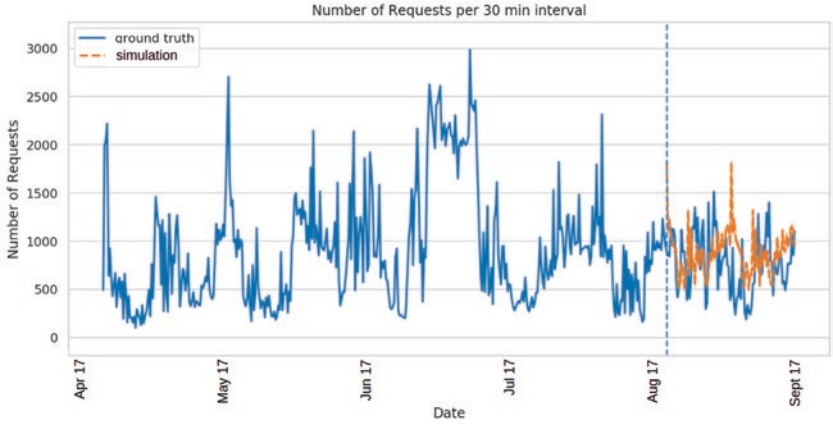

Fig. 2.7 Simulated workload for a search engine

In RECAP, the overall idea behind the use of GANs is twofold. Firstly, using this approach provides a "what-if" analysis on a dataset, answering such questions as "how would this workload look for a larger number of nodes?" Secondly, the inherent training goal of a GAN is to estimate the probability distribution of the training data and to generate synthetic samples drawn from that distribution. Hence, when applied to a real dataset, the GAN learns to mimic its statistical properties.

2.3.4.3 Traffic-Propagation-based Workload Generation

RECAP implements five diffusion algorithms for workload generation. These algorithms can be divided into two groups: non-hierarchical and hierarchical workload diffusion. The former includes population-based, location-based, and bandwidth-based algorithm; the latter includes hierarchy-based and network-routing-based algorithm.

Diffusion algorithms can be applied in different use cases and under different assumptions related to the network topology, network links' capacity, and the distribution of users throughout the network. Given these models, and real workload data traces collected as time series at a limited number of locations, it is possible to produce workload traces for any or all network locations.

2.3.4.4 Simulation System Model Data Sets

The role of simulation in the RECAP project is to go beyond the limitations of an available testbed in terms of scale and complexity of experimentations. Based on simulation, it is possible to generate synthetic datasets consisting of two parts: large-scale models of a system that is being simulated, and simulated behaviour measurements of the modelled system. This is discussed further in Chap. 5.

2.4 Data Visualisation

Data visualisation empowers end users and data scientists to analyse and reason about data and its features. With data visualisation, data sets produced by RECAP or collected from production systems of use cases are transformed to be more accessible, understandable and consumable. RECAP uses a range of visualisation tools which we will now discuss.

2.4.1 Visualisation for Data Analysis

To facilitate data analysis and reasoning, RECAP has adopted various visualisation tools for data presentation, for instance the histogram, box plot, and scatter plot. Upon dealing with heterogeneous data sets, the selected tools enable both univariate and multivariate data visualisation, facilitating corresponding data analysis methods applied to different data sets. To illustrate the use of the visualisation tools as well as their facilitation of data analysis, different visualisations of features extracted from a real data set from a search engine are provided along with explanations of how each visualisation helps retrieving insights into the data.

Figure 2.8 visualises univariate data (specifically, the serving time of user requests in the given workload data set) in different forms. The histogram provides the insight that the majority of user requests are served within very short time periods. The observable data distribution suggests a potential application of probabilistic modelling techniques is needed to construct models of the feature for further analysis or workload generation. The box plot of this serving time feature shows a large number of outliers exist in the data set. Further investigation is thus required for hints on the construction of predictive models. Figure 2.9 visualises multivariate data and shows a relationship between the response size and response time of the user requests. This visualisation suggests a correlation analysis on the data set is needed when addressing workload analysis and modelling.

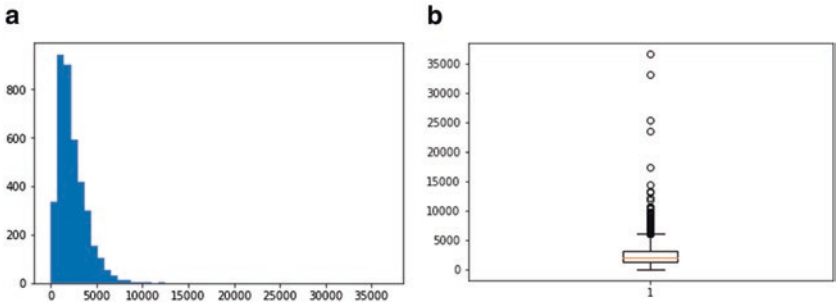

Fig. 2.8 An exemplary presentation of serving time of requests in a workload data set. (**a**) Histogram of serving time of user requests. (**b**) Box plot of serving time of user requests

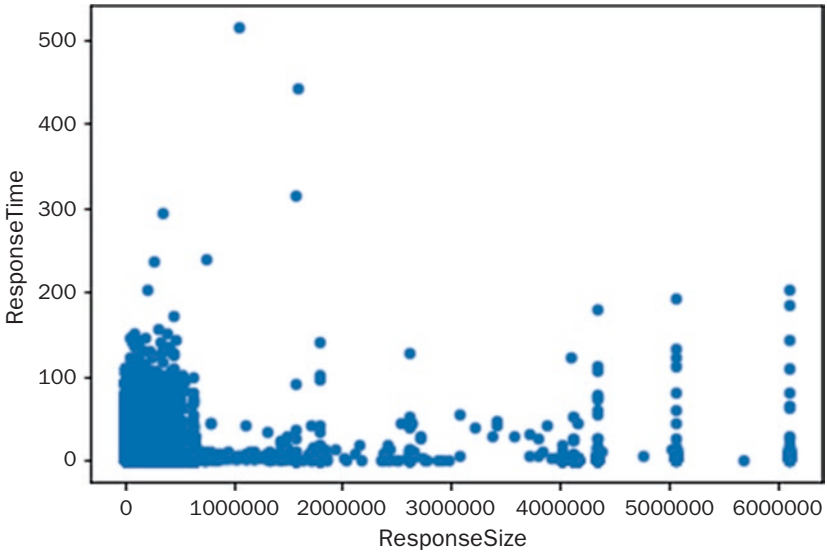

Fig. 2.9 An exemplary presentation of a correlation of features in a workload data set

2.4.2 Visualisation of RECAP Telemetry Data

The visualisation of telemetry data makes the status of the infrastructure and applications operating on the infrastructure more comprehensible for the operators at both application and infrastructure level. This becomes

crucial for the automation of system (application and infrastructure) management, in which trust is required and can be established based on visualisations illustrating the response of the system to the triggered and ongoing management actions. In RECAP, telemetry data acquired from use case testbeds and production systems need to be visualised in order to aid the analysis of the workload and application behaviours as well as the mutual dependencies between metrics or features of both the infrastructure and applications.

As discussed, to facilitate the visualisation, Grafana was used as a visualisation tool. This is an open source tool with a large community and a wide selection of plugins and pre-configured dashboards which accelerates visualisation. Grafana has an easy-to-use interface with various graph visualisation techniques including line graphs, bars, heat maps, maps, and architecture. It enables grouping various graphs into a single-view dashboard and supports multiple dashboards to provide different perspectives of a given data set. Figures 2.10 and 2.11 illustrate the snapshots of two dashboards. The first includes multiple graphs showing resource utilisation of the core of a testbed deployed at Ulm University (UULM), Germany, and the second illustrates the mobility and behaviour of users emulated in a testbed deployed at Tieto, Sweden, in a study of Infrastructure and Network Management.

Fig. 2.10 Snapshot of the dashboard for the testbed at UULM

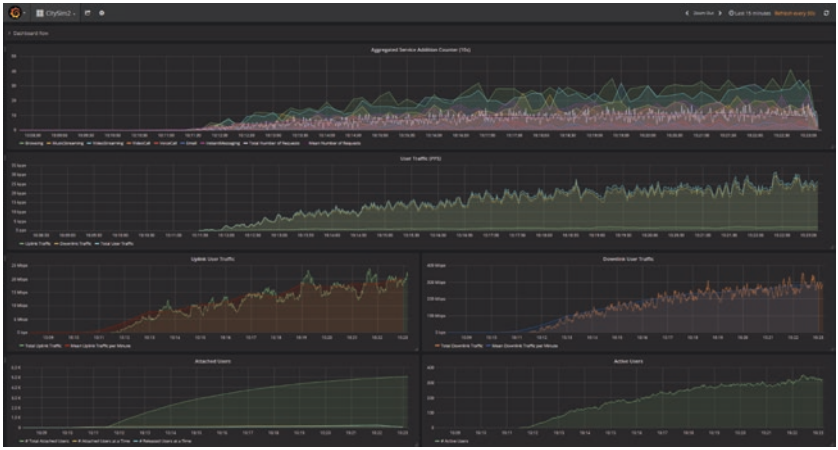

Fig. 2.11 Snapshot of the dashboard for the testbed at Tieto

2.5 Open Data

The RECAP project adheres to the Open Data Pilot of the European Commission. This means that the project committed to providing the datasets required to reproduce the results in the project, unless this would result in, for example, a breach of confidentiality for the dataset provider or in the loss of intellectual property. Several datasets have been derived and provided in the context of RECAP. These datasets are described in RECAP's Deliverable D5.3 and are available, where appropriate, at RECAP's website—https://recap-project.eu.

References

Le Duc, Thang, Rafael García Leiva, Paolo Casari, and Per-Olov Östberg. 2019. Machine Learning Methods for Reliable Resource Provisioning in Edge-Cloud Computing: A Survey. *ACM Computing Surveys* 52 (5): 94:1–94:39.

García Leiva, Rafael, Antonio Fernández Anta, Vincenzo Mancuso, and Paolo Casari. 2019. A Novel Hyperparameter-Free Approach to Decision Tree Construction That Avoids Overfitting by Design. *IEEE Access* 7: 99978–99987.

Groenda, Henning, et al. 2016. CACTOS Toolkit Version 2. CACTOS Project Deliverable.

Harvey, A.C. 1989. *Forecasting, Structural Time Series Models and the Kalman Filter*. Cambridge University Press.

Huang, Liang, et al. 2005. *On-Line Sequential Extreme Learning Machine*. The International Conference on Computational Intelligence.

Liang, N.-Y., and G.-B. Huang. 2006. A Fast and Accurate Online Sequential Learning Algorithm for Feedforward Networks. *IEEE Transactions on Neural Networks* 17 (6): 1411–1423.

MELODIC. 2019. *Multi-cloud Management Platform*. http://www.melodic.cloud/.

Park, Jin-Man, and Jong-Hwan Kim. 2017. *Online Recurrent Extreme Learning Machine and Its Application to Time-Series Prediction*. Proceedings of the International Joint Conference on Neural Networks (IJCNN), Anchorage, AK.