An Elastic Software Architecture for Extreme-Scale Big Data Analytics

Maria A. Serrano, César A. Marín, Anna Queralt, Cristovao Cordeiro, Marco Gonzalez, Luis Miguel Pinho, and Eduardo Quiñones

Abstract This chapter describes a software architecture for processing big-data analytics considering the complete compute continuum, from the edge to the cloud. The new generation of smart systems requires processing a vast amount of diverse information from distributed data sources. The software architecture presented in this chapter addresses two main challenges. On the one hand, a new elasticity concept enables smart systems to satisfy the performance requirements of extreme-scale analytics workloads. By extending the elasticity concept (known at cloud side) across the compute continuum in a fog computing environment, combined with the usage of advanced heterogeneous hardware architectures at the edge side, the capabilities of the extreme-scale analytics can significantly increase, integrating both responsive data-in-motion and latent data-at-rest analytics into

M. A. Serrano · E. Quiñones (✉)
Barcelona Supercomputing Center (BSC), Barcelona, Spain
e-mail: eduardo.quinones@bsc.es

C. A. Marín
Information Catalyst for Enterprise Ltd, Crewe, UK
e-mail: cesar.marin@informationcatalyst.com

A. Queralt
Universitat Politècnica de Catalunya (UPC), Barcelona, Spain

Barcelona Supercomputing Center (BSC), Barcelona, Spain
e-mail: anna.queralt@bsc.es

C. Cordeiro
SIXSQ, Meyrin, Switzerland
e-mail: cristovao.cordeiro@sixsq.com

M. Gonzalez
Ikerlan Technology Research Centre, Basque Research Technology Alliance (BRTA), Arrasate/Mondragón, Spain
e-mail: marco.gonzalez@ikerlan.es

L. M. Pinho
Instituto Superior De Engenharia Do Porto (ISEP), Porto, Portugal
e-mail: lmp@isep.ipp.pt

a single solution. On the other hand, the software architecture also focuses on the fulfilment of the non-functional properties inherited from smart systems, such as real-time, energy-efficiency, communication quality and security, that are of paramount importance for many application domains such as smart cities, smart mobility and smart manufacturing.

Keywords Smart mobility · Software architecture · Distributed big data analytics · Compute continuum · Fog computing · Edge computing · Cloud computing · Non-functional requirements · Cyber-security · Energy-efficiency · Communications

1 Introduction

The extreme-scale big data analytics challenge refers not only to the heterogeneity and huge amount of data to be processed both on the fly and at rest but also to the geographical dispersion of data sources and the necessity of fulfilling the non-functional properties inherited from the system, such as real-time, energy efficiency, communication quality or security. Examples of smart systems that can exploit the benefits of extreme-scale analytics include production lines, fleets of public transportation and even whole cities. Providing the required computing capacity for absorbing extreme (and geographically dispersed) amounts of collected complex data, while respecting system properties, is of paramount importance to allow converting the data into few concise and relevant facts that can be then consumed and be decided or acted upon.

In a typical smart system (e.g., a smart city), data is collected from (affordable) sensors to gather large volumes of data from distributed sources using Internet of Things (IoT) protocols. The data is then transformed, processed and analysed through a range of hardware and software stages conforming the so-called *compute continuum*, that is from the physical world sensors close to the source of data (commonly referred to as edge computing) to the analytics backbone in the data centres (commonly located in the cloud and therefore referred to as cloud computing). Due to the computing complexity of executing analytics and the limited computing capabilities of the edge side, current approaches forward most of the collected data to the cloud side. There, big data analytics are applied upon large datasets using high-performance computing (HPC) technologies. This complex and heterogeneous layout presents two main challenges when facing extreme-scale big data analytics.

The first challenge refers to the non-functional properties inherited from the application domain:

- *Real-time* big data analytics is becoming a main pillar in industrial and societal ecosystems. The combination of different data sources and prediction models within real-time control loops will have an unprecedented impact in domains such as smart city. Unfortunately, the use of remote cloud technologies makes

infeasible to provide real-time guarantees due to the large and unpredictable communication costs on cloud environments.
- *Mobility* shows increased trade-offs and technological difficulties. Mobile devices are largely constrained by the access of *energy*, as well as suffering from *unstable communication*, which may increase random communication delays, unstable data throughput, loss of data and temporal unavailability.
- *Security* is a continuously growing priority for organization of all sizes, as it affects data integrity, confidentiality and potentially impacting on safety. However, strict security policy management may hinder the communication among services and applications, shrinking overall performance and real-time guarantees.

Overall, while processing time and energetic cost of computation is reduced as data analytics is moved to the cloud, the end-to-end communication delay and the performance of the system (in terms of latency) increases and becomes unpredictable, making not possible to derive real-time guarantees. Moreover, as computation is moved to the cloud, the required level of security increases to minimize potential attacks, which may end up affecting the safety assurance levels, hindering the execution and data exchange among edge and cloud resources.

The second challenge refers to the elasticity concept. In recent years, the dramatic growth in both data generation and usage has resulted in the so-called three V's challenges of big data: volume (in terms of data size), variety (in terms of different structure of data, or lack of structure), and velocity (in terms of the time at which data need to be processed). These factors have contributed to the development of the elasticity concept, in which cloud computing resources are orchestrated to provide the right level of service (in terms of system throughput) to big data workloads. The elasticity concept, however, does not match the computing requirements when considering extreme-scale analytics workloads. On the one side, elasticity does not take into account the computing resources located on the edge. The advent of new highly parallel and energy-efficient embedded hardware architectures featuring graphical processing units (GPUs), many-core fabrics or FPGAs, have significantly increased the computing capabilities on the edge side. On the other side, elasticity mainly focuses on system throughput, without taking into account the non-functional properties inherited from the domain.

Addressing together these two important challenges along the compute continuum, that is from the edge to the cloud, is of paramount importance to take full benefit of extreme-scale big data analytics in industrial and societal environments such as smart cities. This chapter describes an end-to-end solution applied along the complete compute continuum to overcome these challenges. Concretely, the ELASTIC project [1], funded by the European Union's Horizon 2020 Programme, faces these challenges and proposes a novel software platform that aims to satisfy the performance requirements of extreme-scale big data analytics through a novel elasticity concept that distributes workloads across the compute continuum. The proposed software framework also considers the non-functional requirements of the system, that is operation with real-time guarantees, enhanced energy efficiency, high communication quality and security against vulnerabilities.

The chapter relates to the technical priority "Data Processing Architectures" of the European Big Data Value Strategic Research & Innovation Agenda [13]. Moreover, the chapter relates to the "Systems, Methodologies, Hardware and Tools" cross-sectorial technology enablers of the AI, Data and Robotics Strategic Research, Innovation & Deployment Agenda [14]. The rest of the chapter is organized as follows: Sect. 2 describes the ELASTIC software architecture. Concretely, Sect. 2.1 motivates the use of such framework in the smart city domain, Sect. 2.2 provides an overview of the layered software framework, and Sects. 2.3–2.6 describe each layer in detail. Finally, Sect. 3 concludes the chapter.

2 Elastic Software Architecture

2.1 Applicability to the Smart City Domain

One of the domains in which extreme-scale big data analytics can have a significant impact on people's day-to-day life is Smart Cities. Big data is increasingly seen as an effective technology capable of controlling the available (and distributed) city resources in a safely, sustainably, and efficiently way to improve the economical and societal outcomes. Cities generate a massive amount of data from heterogeneous and geographically dispersed sources including citizens, public and private vehicles, infrastructures, buildings, etc.

Smart cities can clearly benefit from the proposed software architecture, capable of deploying federated/distributed, powerful and scalable big data systems to extract valuable knowledge, while fulfilling the non-functional properties inherit from the smart cities. This opens the door to a wide range of advanced urban mobility services, including public transportation and traffic management. Therefore, the proposed software architecture is being tested in the city of Florence (Italy), to enhance the tramway public transportation services, as well as its interaction with the private vehicle transportation. The new elasticity concept will enable the efficient processing of multiple and heterogeneous streams of data collected from an extensive deployment of Internet of Things (IoT) sensors, located on board the tram vehicles, along the tramway lines, as well as on specific urban spots around the tram stations (e.g. traffic lights).

Concretely, three specific applications have been carefully identified to assess and highlight the benefits of ELASTIC technology for newly conceived mobility solutions (more details can be found in the ELASTIC project website [1]):

- Next Generation Autonomous Positioning (NGAP) and Advanced Driving Assistant System (ADAS): NGAP enables the accurate and real-time detection of the tram position through data collected and processed from on-board sensors. The positioning information is then sent through a reliable connection to the tram operation control system on the ground. This information also enables the development of ADAS, for obstacle detection and collision avoidance functionalities based on an innovative data fusion algorithm combining the output of multiple sensors (radars, cameras and LIDARs). Data from additional sources, such as fixed sensors placed at strategic positions in the streets (e.g., road crossings), are also integrated to increase the reliability of the system.

- Predictive maintenance: It monitors and profiles the rail track status in real time, enabling the identification of changes in equipment behaviour that foreshadow failure. Furthermore, through offline analytics, potential correlations between unexpected detected obstacles (obtained through the NGAP/ADAS application) and rail track damages are examined. The application also provides recommendations, enabling maintenance teams to carry out remedial work before the asset starts to fail. Finally, the power consumption profile is also monitored and processed in real time, in order to potentially minimize consumption and have an environmentally positive impact.
- Interaction between the public and private transport in the City of Florence: ELASTIC uses the information from the city network of IoT sensors to enhance the quality of the city traffic management, providing valuable outputs for both users and operators that will enable them to: (1) Identify critical situations (e.g. vehicles crossing the intersection with the tram line despite having a red traffic light) (2) Optimize the local traffic regulation strategies (e.g. reduce the waiting time of cars at tram crossings through improved light priority management, or slow down trams to reduce a queue of waiting vehicles, etc.)

2.2 ELASTIC Layered Software Architecture: Overview

In any smart system, large volumes of data are collected from distributed sensors, transformed, processed and analysed, through a range of hardware and software stages conforming the so-called compute continuum, that is from the physical world sensors (commonly referred to as edge computing), to the analytics back-bone in the data centres (commonly referred to as cloud computing). The proposed software architecture to efficiently manage and process this complex data processing scenario is shown in Fig. 1, and it is composed of the following layers:

- *Distributed Data Analytics Platform (DDAP)*: It provides the data accessibility and storage solutions, and the APIs. The data solutions provide the set of mechanisms needed to cope with all data-type variants: formats, syntax, at-rest and in-motion, 3V's (volume, velocity and variety), edge, cloud, etc. The APIs allow to extract valuable knowledge from the connected data sources using distributed and parallel programming models.
- *Computation Orchestrator*: It implements the elasticity concept in which the computing resources will be properly orchestrated across the compute continuum to provide the right level of service to big data analytics workloads. To do so, the orchestrator does not only consider the overall system throughput but also the fulfilment of non-functional properties inherited from the application domain.

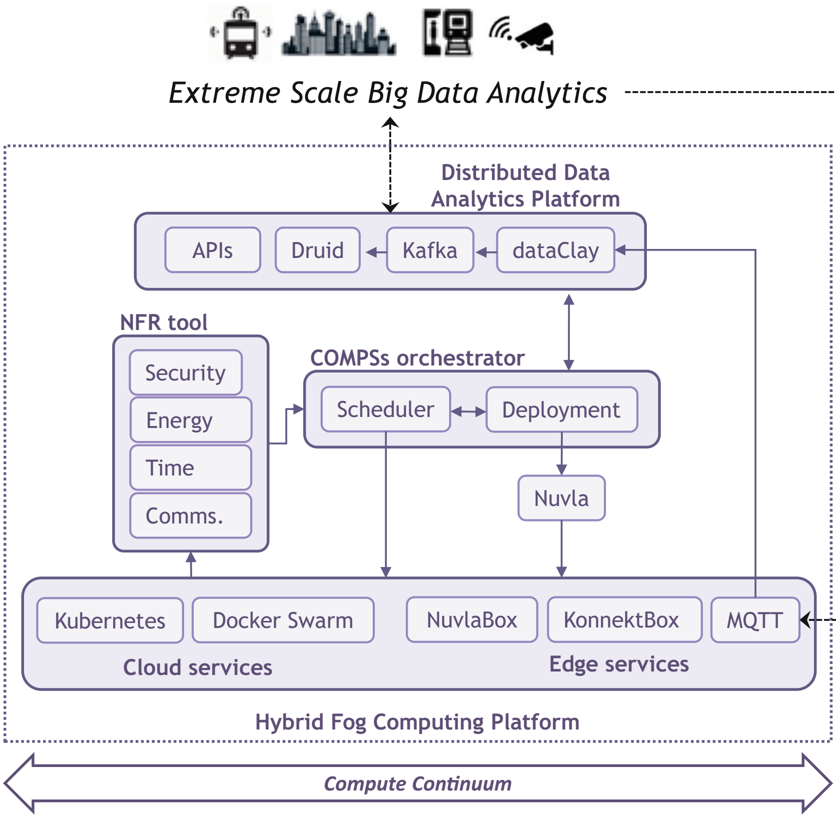

Fig. 1 Overview of the elastic software architecture

This layer supports the APIs exposed to the programmer to efficiently distribute the execution of the analytics in a transparent way, while exploiting the inherent parallelism of the system and abstracting the application from the underlying distributed fog computing architecture.

- *Non-functional Requirements (NFR) Tool*: It provides the required support to monitor and manage the behaviour of the system, in order to guarantee some level of fulfilment of the non-functional requirements of the supported applications, that is real-time guarantees, energy efficiency, communication quality and security properties.
- *Hybrid Fog Computing Platform*: It abstracts the multiple edge and cloud computing resources spread across the compute continuum. To do so, this layer deploys the application components, that is the computational units distributed by the above layer, to virtual resources using container technologies, and considering configuration and infrastructural requirements.

Overall, the aim of the elastic software architecture is to enable the design, implementation and efficient execution of extreme-scale big data analytics. To do so, it incorporates a novel elasticity concept across the compute continuum, with the objective of providing the level of performance needed to process the envisioned volume and velocity of data from geographically dispersed sources at an affordable development cost, while guaranteeing the fulfilment of the non-functional properties inherited from the system domain. The following subsections provide a detail description of each software architecture component.

2.3 Distributed Data Analytics Platform

The distributed data analytics platform is developed to cater for domain specific as well as generic needs of analysing data across the compute continuum. Concretely, this layer takes care of two important matters: (1) the actual development of data analytics, providing APIs support (Sect. 2.3.1), and (2) the management, storage and retrieval of data at the time it is needed and at the location where it is needed (Sect. 2.3.2).

2.3.1 Application Programming Interfaces (APIs)

The software architecture provides support for the development of big data analytics methods, capable of analysing all the data collected by IoT sensors and distributed devices. As an example, Deep Neural Networks (DNNs) are used for image processing and predictive modelling, and aggregation and learning methods (based on unsupervised-learning strategies) are used for automatically detecting data patterns, including ant-based clustering, formal concept analysis and frequent pattern mining.

This layer also provides an API to support distributed and parallel computation. This enables the simultaneous use of multiple compute resources to execute software applications. Concretely, the COMPSs [4] task-based programming model is supported to allow developers to simply specify the functions to be executed as asynchronous parallel tasks. At runtime, the system exploits the concurrency of the code, automatically detecting and enforcing the data dependencies between tasks and deploying these tasks to the available resources, which can be edge devices or nodes in a cluster. More details of this component are provided in Sect. 2.4.

One of the key competitive advantages of the DDAP is that these methods are offered to the software developer in a unique development environment. Moreover, big data analytics methods can be optimized to be executed at both, the edge and the cloud side, providing the required flexibility needed to distribute the computation of complex big data analytics workflows across the compute continuum.

2.3.2 Data Accessibility and Storage

One of the main goals of the distributed data analytics platform (DDAP) is to ensure data accessibility across the compute continuum, covering aspects such as data-in-motion and data-at-rest, for data analytic applications. To do so, the DDAP is currently composed of the following main components:

- *dataClay* [8], distributed at the edge/fog side, is responsible for managing the information generated in real time, covering the data-in-motion needs. dataClay is an active object store that can handle arbitrary data structures in the form of objects and collections, as in object-oriented programming, which allows the application programmer to manage data as if it was just in memory. It is highly optimized for accessing and manipulating data at a fine granularity, and it can run in heterogeneous devices, from the edge to the cloud.
- *Druid* [3], distributed across the fog/cloud side, is responsible for collecting all information generated and shared across DDAP; it is a column-based data warehouse tuned to ingest large amounts of time series data such as that generated by a transport infrastructure. Druid is distributed by design and optimized for visual analytics. It contains mechanisms for a fast and easy access to data regardless of its location. This functionality makes Druid a complementing element in DDAP suitable for covering data-at-rest needs.
- *Kafka* [2] is a well-known message queue for streaming data; it functions as a transient message queue to transfer data from dataClay to Druid at each station. In DDAP, Kafka can be seen as the boundary between data-in-motion and data-at-rest.

The combination of dataClay, Druid, and Kafka makes DDAP suitable for real-time and historical big data analytics at the same time, as these solutions complement each other. In particular, Kafka helps enforce a unidirectional data flow from dataClay to Druid, effectively making DDAP operate as a well-known Content Delivery Network. In the latter, data is generated and processed at the edge for real-time needs, then it is collected at the cloud from distributed locations, and finally the content, that is historical big data analytics results, is delivered to interested users.

An example of the DDAP functionality can be seen in Fig. 2. The applications executed in the tram provide different kinds of data, such as the objects detected by its cameras and sensors, or the tram position. In the meantime, the applications executed in the context of a tram stop, which includes the stop itself as well as cabinets with cameras in surrounding intersections, also detect objects that may fall out of the visibility scope of the tram. To provide real-time performance, these applications use COMPSs to distribute the work between the different devices. Both data sources are merged in the tram stop in order to predict possible collisions between the objects detected according to their current trajectories. Simultaneously, the objects detected and their positions at each point in time are pushed to Kafka so that they can be ingested by Druid as they are created, thus immediately enabling them to take part of historical analytics triggered from the cloud.

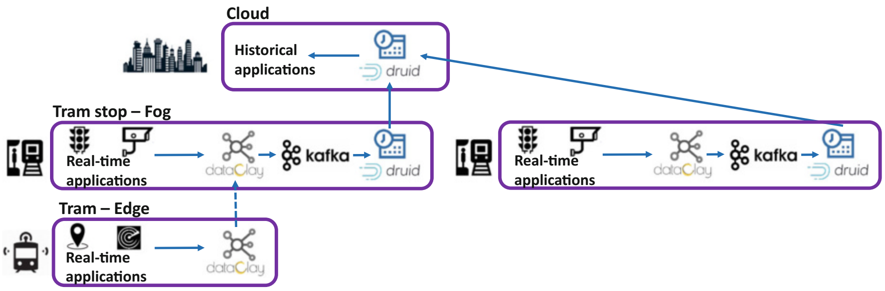

Fig. 2 Example application of DDAP in a transport infrastructure

2.4 *Computation Orchestrator*

This layer provides the software component in charge of distributing the computation across available computing resources in the hybrid for computing platform. Specifically, it implements the *elasticity* concept to properly orchestrate the computing resources across the compute continuum to provide the right level of service to analytics workloads. Moreover, elasticity will not only consider the overall system throughput but also the fulfilment of non-functional properties inherited from the application domain.

The software component in charge of implementing these features is COMPSs [4]. COMPSs is a distributed framework developed at the Barcelona Supercomputing Center (BSC) mainly composed of a *task-based programming model*, which aims to ease the development of parallel applications for distributed infrastructures, such as Clusters, Clouds and containerized platforms, and a *runtime system* that distributes workloads transparently across multiple computing nodes with regard to the underlying infrastructure. In cloud and big data environments, COMPSs provides scalability and elasticity features allowing the dynamic provision of resources. More specifically, the COMPSs task-based model is offered in the DDAP layer to implement big data analytics methods, and the COMPSs runtime implements the scheduling techniques and deployment capabilities to interact with hybrid resources in a transparent way for the programmer.

2.4.1 Task-based Programming Model

COMPSs offers a portable programming environment based on a task execution model, whose main objective is to facilitate the parallelization of sequential source code (written in Java, C/C++ or Python programming languages) in a distributed and heterogeneous computing environment. One of the main benefits of COMPSs is that the application is agnostic from the underlying distributed infrastructure. Hence, the COMPSs programmer is only responsible for identifying the portions of code, named COMPSs *tasks*, that can be distributed by simply annotating the sequential

Fig. 3 Video analytics COMPSs example

```
1  @task (camera = IN, returns = numpy.ndarray)
2  def get_video_frame(cameraID):
3    return get_next_frame(cameraID)

5  @task (frame = IN, returns = list)
6  def video_analytics(frame):
7    return process(frame)

9  @task (list_results = IN)
10 def collect_and_display(list_results):
11   update_dashboard(list_results)

13 ### Main function ###
14 while (true):
15   for i,cam in enumerate(cameras_set)
16     frame[i] = get_video_frame(cam)
17     results[i] = video_analytics(frame[i])
18   collect_and_display(results)
```

source code. Data dependencies and their directionality (i.e. `in`, `out` or `inout`) are also identified. Upon them, the COMPSs runtime determines the order in which COMPSs tasks are executed and also the data transfers across the distributed system. A COMPSs task with an `in` or `inout` data dependency cannot start its execution until the COMPSs task with an `out` or `inout` dependency over the same data element is completed. At run-time, COMPSs tasks are spawned asynchronously and executed in parallel (as soon as all its data dependencies are honoured) on a set of distributed and interconnected computing resources. Moreover, the data elements marked as `in` and `inout` are transferred to the compute resource in which the task will execute if needed.

Figure 3 shows a basic example of a Python COMPSs application (PyCOMPSs [12]) that performs video analytics. COMPSs tasks are identified with a standard Python decorator `@task`, at lines 1, 5 and 9. The `returns` argument specifies the data type of the value returned by the function (if any), and the *IN* argument defines the data directionality of function parameters. The main code starts at line 14, where the application iterates to process video frames over the time. Then, at line 15 a loop iterates over the available camera video feeds, and first it gets the next frame by instantiating the COMPSs task defined at line 1. At line 17, the COMPSs task that process the video frame (defined at line 5) is instantiated. Finally, all the results are collected at line 18, instantiating the COMPSs task defined at line 9.

2.4.2 Runtime System

The task-based programming model of COMPSs is supported by its runtime system, which manages several aspects of the application execution, keeping the underlying infrastructure transparent to it. The two main aspects are the deployment on the available infrastructure and the scheduling of tasks to available computing resources.

Deployment
One of the main features of COMPSs is that the model abstracts the application from the underlying distributed infrastructure; hence, COMPSs programs do not include any detail that could tie them to a particular platform boosting portability among diverse infrastructures and enabling execution in a fog environment. Instead, it is the COMPSs runtime that features the capabilities to set up the execution environment. The COMPSs runtime is organized as a master-worker structure. The Master is responsible for steering the distribution of the application, as well as for implementing most of the features for initialising the execution environment, processing tasks or data management. The Worker(s) are in charge of responding to task execution requests coming from the Master.

The COMPSs runtime support various scenarios regarding deployment strategy and interoperability between edge/cloud resources. Three different scenarios, compatible between them, are supported:

- Native Linux, monolithic: The big data analytics workload is natively executed in a Linux-like environment. In this configuration, all the nodes available for the execution of a COMPSs workflow require the native installation of COMPSs, and the application.
- Containerized, Docker: A Docker COMPSs application image contains the needed dependencies to launch a COMPSs worker and the user application. In this case, there is no need for setting up the execution environment in advance in all the nodes, but only Docker must be available. Docker image repositories, e.g. Docker Hub, can make the image, and hence the application, available anytime and anywhere. In this deployment, COMPSs takes care of making the image available at the nodes and launching the containers.
- Cloud provider: A cloud infrastructure, in this context, refers to a data centre or cluster with great computing capacity that can be accessed through an API and that can lend some of that computational power, for example in the form of a container. This is the case of a Docker Swarm or Kubernetes cluster. COMPSs also supports the deployment of workers in these infrastructures, using the Nuvla API (see Sect. 2.6).

Scheduling
One key aspect of the COMPSs runtime scheduler is that it maintains the internal representation of a COMPSs application as a Direct Acyclic Graph (DAG) to express the parallelism. Each node corresponds to a COMPSs task instance and edges represent data dependencies. As an example, Fig. 4 shows the DAG representation for three iterations of the COMPSs application presented in Fig. 3, when three camera video feeds are processed. Based on this DAG, the runtime can automatically detect data dependencies between COMPSs tasks.

The COMPSs scheduler is in charge of distributing tasks among the available computing resources and transferring the input parameters before starting the execution, based on different properties of the system such as the non-functional

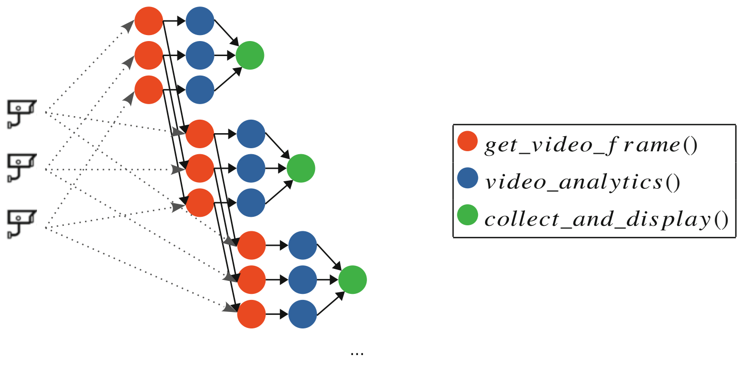

Fig. 4 DAG representation of the COMPSs example in Fig. 3

requirements (real-time, energy, security and communications quality) analysed by the NFR tool (see Sect. 2.5).

2.5 Non-functional Requirements Tool

The software architecture presented in this chapter addresses the challenge of processing extreme-scale analytics, considering the necessity of fulfilling the non-functional properties inherited from the system and its application domain (e.g. smart manufacturing, automotive, smart cities, avionics), such as real time, energy efficiency, communication quality or security.

This task is led by the Non-functional Requirements (NFR) Tool layer (see Fig. 1), in collaboration with the Orchestrator layer, and the Hybrid Fog Computing Platform. The NFR tool continuously monitors and evaluates the extent to which non-functional properties' required levels are guaranteed in the fog computing platform. Moreover, this tool identifies and implements the appropriate mechanisms to deal with the NFRs, monitoring system behaviour and helping taking decisions (such as offloading or reducing performance). *Runtime monitoring* of system status is used to detect NFR violations, while a *Global Resource Manager* guides the evolution towards configurations that are guaranteed to satisfy the system's NFRs.

The NFR monitoring is conceptually constituted by *probes*, i.e. the system tools that provide monitoring data. The probes are in charge of interfacing with the underlying fog platform (OS and/or hardware), to collect the required information, which is used to detect NFR violations. The NFR Monitors are per-property-specific components, which, based on the information from the probes, and the application information, determines if some requirement is not being met. This information is shared with the Orchestrator that may (re)configure the scheduling of a given application to meet its requirements. The Global Resource Manager is the component that considers a holistic approach, providing decisions based on a global

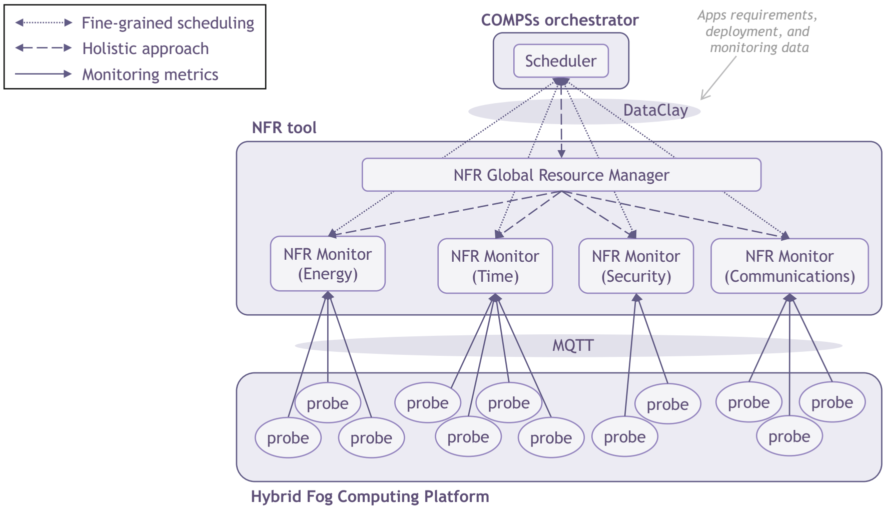

Fig. 5 NFR tool architecture and synergies within the elastic software architecture

view of the system (composed of distributed computing nodes), and considering simultaneously all non-functional properties. This decision is also shared with the Orchestrator to (re)configure the applications accordingly. Figure 5 shows the NFR tool internal structure and the synergies with the Fog platform (see Sect. 2.6) and the Orchestrator layer (see Sect. 2.4).

Next subsections describe the concrete NFR metrics analysis.

2.5.1 Real Time

Coping with real-time computing across the compute continuum requires the ability to specify and manage different timing perspectives. Two main challenges arise: tasks deployed at the edge (e.g. on board the connected car) need to guarantee "hard real-time" responses (e.g. very low latency), while those deployed at the cloud need to guarantee certain QoS levels regarding time: right-time or "soft real-time" guarantees. Closer to the environment, at the edge, tight timing mapping and scheduling approaches can be used, while at the cloud, time is measured in terms of average statistical performance with Quality of Service (QoS) constraints. These perspectives complement each other, and the elastic software architecture provides solutions that try to guarantee the required response time to applications while optimizing energy and communication costs.

To do so, it is necessary to monitor different timing properties, in all nodes of the distributed fog infrastructure. This ranges from monitoring actual CPU utilization and execution time of applications to detection of deadline violations or memory accesses.[1] This monitoring allows to dynamically adjust the system resources to which the application is mapped, depending on the actual load of the system.

2.5.2 Energy

The NFR tool augments the system "introspection" capabilities in terms of power consumption, by means of energy-aware execution models, from the hardware platform to the holistic system. This allows to propagate workload-specific monitoring information from the run-time to the decision-making module, which can be exploited to better adapt to the requirements, as well as to the time predictability and security optimization levels. Furthermore, a richer knowledge of applications' requirements and concurrency structure, coupled with precise energy models for the underlying hardware, combined with the possibility of dynamically switching between edge and cloud deployments, constitutes an enabling factor towards larger energy savings.

Concretely, the NFR tool monitors the power consumption of the different hardware components on edge devices (e.g. System on Chip (SoC), CPU, GPU, etc.). This allows to develop energy-aware execution models and efficiency tune power consumption over the complete continuum.

2.5.3 Security

Verifying that applications correctly comply with security mechanisms and do not contain vulnerabilities is essential. This implies much more than an online analysis and monitoring, e.g. GDPR [6] regulation compliance, secure communication protocols, the use of device certificates and mutual authentication (server and client), etc. Besides these design decisions, in order to guard against security threats, the NFR tool continuously monitors the systems and applications deployed and incorporates security upgrades to software and deploy updates to existing configurations.

The security monitoring component is based on OpenSCAP [9], an open-source tool that simply implements the Security Content Automation Protocol (SCAP), as a vulnerability scanner. OpenSCAP can easily handle the SCAP standards and generate neat, HTML-based reports. The NFR monitor tool and the global resource manager take simple decisions concerning security: The security status of the computing nodes is monitored, providing a security score for each of them. Then, the list of available (secure) nodes is updated for each application, based on its particular requirements.

[1] Memory accesses can be used to provide information on contention accessing shared memory, providing a more accurate timing analysis for hard real-time applications.

2.5.4 Communications Quality

In the context of wireless communications, and especially LTE networks, several performance parameters need to be considered to characterize system behaviour. Different types of service prioritize different figures of merit due to the nature of the information to be transmitted and/or received. For instance, packet loss rate plays a paramount role in VoIP services, whereas high throughput is not strictly required, given that VoIP does not generate high volumes of data. On the contrary, video streaming and file transfer services demand a much higher throughput.

The NFR tool considers the following communication monitoring metrics to evaluate the communications quality of the system: active network interfaces, transmitted/received data volume, average throughput, roundtrip time (RTT), packet loss rate (PLR). These metrics provide information that is considered both at the orchestrator, to take fine-grained scheduling decisions (see Sect. 2.4), and at the global resource manager to consider communications quality in the holistic approach.

2.6 Hybrid Fog Computing Platform

Fog computing encompasses the benefits of edge and cloud computing: on the one hand, devices have increased computer capability, on the other hand, Cloud Computing has matured strongly. The main focus of the elastic software architecture is to obtain the best from both approaches (edge and cloud) into fog architecture. While fog computing has recently led to great interest by the research community and industry, it is still a conceptual approach [2]. The elastic software architecture presented in this chapter considers two main concepts for its hybrid fog architecture: (1) a software stack that can be run in (almost) any computing device and (2) the coordination between edge and cloud components to efficiently support elasticity across the compute continuum.

The proposed hybrid fog-computing platform is based on standard open-source reusable components. It follows a microservice-based design, thus decoupling the composing components, which makes the overall solution generic and capable of coping with a wide range of smart edge devices and clouds. The hybrid architecture (see Fig. 6) allows the use of dynamic applications in the form of microservices (containers) or native applications (monolithic). Predictability (native) and flexibility (microservices) can be achieved with this approach. The platform is compatible with both systems offering a high level of flexibility for the use case.

Next subsections provide an overview of each component.

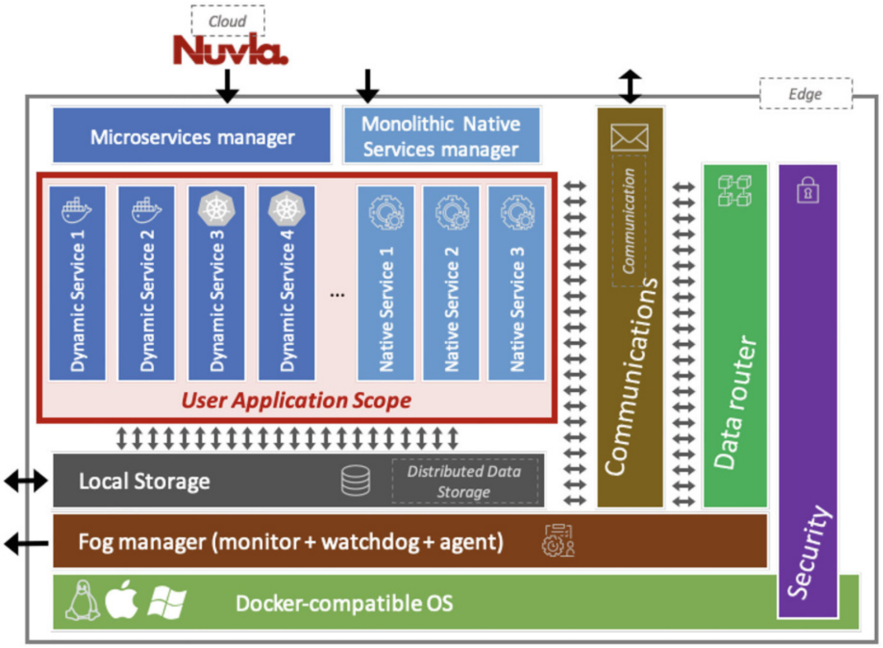

Fig. 6 Hybrid Fog Architecture

2.6.1 Cloud: Nuvla

Nuvla [10] acts both as the orchestration support and deployment engine for all micro-service-based workloads being submitted into both cloud infrastructures and edge devices. As an open-source software stack, Nuvla can be run anywhere. In particular, the elastic software architecture profits from its existing SaaS offering running in the Exoscale cloud,[2] at https://nuvla.io/. Nuvla offers the following services:

- *Application Registration*: users can register Docker Swarm and Kubernetes applications in Nuvla.
- *Infrastructure Registration*: users can register new Docker Swarm and Kubernetes infrastructures in Nuvla (be those at the cloud or at the edge).
- *NuvlaBox Registration*: users can create new NuvlaBoxes via Nuvla. Nuvla will provide users with a "plug-and-play" installation mechanism that can be executed on any Docker compatible device.
- *Resource Sharing*: users can share their Nuvla resources (applications, infrastructures, etc.) with other users.

[2] https://www.exoscale.com/.

- *Application deployment*: users can launch their applications into any of their Nuvla infrastructures.
- *Application Monitoring*: all the deployed applications are monitored from Nuvla, giving users an overall view of the deployment status.
- *Edge Monitoring*: all NuvlaBoxes can be monitored and managed from Nuvla. Resource consumption, external peripheral, and lifecycle management options are provided to users from Nuvla.
- *RESTful API*: a standardized and language-agnostic API is available to all Nuvla users, providing full resource management capabilities, plus a comprehensive querying and filtering grammar.

2.6.2 Edge: KonnektBox and NuvlaBox

Two commercial edge solutions are being used as the ground foundation for the edge infrastructure in the elastic software architecture: the KonnektBox [7] and the NuvlaBox [11].

The IKERLAN *KonnektBox* is an industry-oriented digitization solution built over EdgeXFoundry [5], an open-source project backed by Linux Foundation which provides basic edge building blocks. KonnektBox uses a mix between vanilla EdgeXFoundry components and custom services tailored for Industry 4.0 use cases.

The *NuvlaBox* is a secured plug-and-play edge to cloud solution, capable of transforming any Docker compatible device into an edge device. This software solution has been developed by SixSq and is tightly coupled with the application management platform, Nuvla.

2.6.3 Fog Components

As the officially adopted edge software appliances for ELASTIC, both the KonnektBox and NuvlaBox provide their own implementation for each of ELASTIC's Fog Architecture building blocks:

Docker-Compatible OS

In order to comply with the reconfiguration and dynamic fog-cloud service execution requirements of the project, a micro services architecture is required. Docker is the standard open-source micro services software. Docker allows the execution of micro services in the form of Docker containers. Each one of the containers runs in an isolated environment and interfaces with other services via network communications (REST APIs, message brokers, etc.). Docker allows the configuration of priorities and limits for each container. For example, the maximum CPU usage by each container, number of CPUs to use, CPU quota, maximum RAM, etc. If a Linux kernel with the real-time extension is used, the priority of each container can also be defined.

Microservices Manager

The dynamic services refer to the different applications that the platform will run. The applications range from industrial protocol drivers to AI inference, DB manager, etc. These applications shall be independent from one another, and the communication between them should be established via some predefined APIs defined in the data router. The NuvlaBox self-generates TLS credentials to be used on a secure and dedicated endpoint which relays the Docker API via HTTPS, for external orchestration platforms, like Nuvla, to speak with.

Monolithic Native Services Manager

The native services manager is the module in charge of controlling the monolithic native applications run in the system. The NuvlaBox provides the execution of remotely issued operations, via a secured and dedicated HTTPS endpoint, exposing a RESTful API. Such operations include the generation of user-specific SSH keypair for executing Native workflows via SSH. The KonnektBox supports the remote deployment of services via a secured MQTT-over-TLS cloud connection.

Local Storage

The local/distributed storage of the system will be implemented as a software middleware. BSC component dataClay platform will be used as the base component for structured data. The KonnektBox provides an additional local database using Redis and Consul. Apart from the local system storage (in the form of Docker volumes), the NuvlaBox does not provide any dedicated storage for user applications. Such functionality is left entirely to the user's preferences. The selected storage element for ELASTIC (dataClay) is supported, as an additional module, by the NuvlaBox.

Fog Manager

The system manager is the application in charge of starting up the whole fog platform, monitor it and manage it. The system manager will run as a standalone Linux application. This service will send telemetry and statistics data to the cloud in order to update the NFR analysis (QoS, security, etc.). It will implement a watchdog service in order to control that all the microservices are running correctly and the health of the system is good. The manager can stop, start and update each micro service. A local configuration UI can be deployed to allow the local configuration of the platform. Both the NuvlaBox and KonnektBox include several microservices which are responsible for discovering external peripherals, collecting telemetry data, categorizing the host environment and performing regular performance and security scans. All of this information is periodically sent (on a configurable frequency) both to Nuvla and to a local edge dashboard running on the hosting edge device.

Communications

The communications service offers an abstraction layer between platform services and multiple communication protocols. This service allows the platform user to define rules to select automatically the appropriate communication protocol to be used in different use cases or environments. All the protocols would be secured with TLS1.2 (or DTLS1.2 for UDP-based protocols). The KonnektBox uses MQTT-over-TLS for all the connections with the cloud. The NuvlaBox exposes secure and dedicated HTTPS endpoints for configuration application management (separately).

Data Router

The data router abstracts the communication between micro services and serves as a central point for all data communication. A decision algorithm can be implemented to decide where to send the data (other local micro service, the cloud, the edge, etc.). The NuvlaBox together with the Fog Manager's peripheral discovery functionality provides an MQTT-based messaging system, which not only brokers internal application messages but also automatically consumes and serves sensor data from the existing peripherals, to any subscribing user applications.

Security

The security module handles the security credentials of the platform and it checks the device data and binaries for unintended manipulation. All the critical applications of the system should be signed to only allow the execution of trusted and original applications. If the security module detects some anomalies, the device will be restored to factory defaults. The KonnektBox is integrated with OpenSCAP vulnerability scanner. The NuvlaBox on top of the security scans within the Fog Manager has the ability to automatically update its own database of common vulnerabilities. Upon every scan, and for a configurable set of vulnerabilities found, it can proactively take action, halting certain sensitive internal services or even moving the whole edge device into a quarantine state.

2.6.4 Distributed Storage

The distributed storage component in ELASTIC is implemented by dataClay (see Sect. 2.3). Since dataClay runs on different kinds of devices, it can be integrated at any level throughout the edge to cloud continuum. Its function within the elastic architecture is twofold. On the one hand, it is in charge of storing data gathered by the Data Router and making it accessible in other devices. The embedded computing capabilities of dataClay enable the association of a given behaviour to each type of data, such as synchronization policies or filters before handling it to other devices. On the other hand, dataClay is used by other components, such as the NFR tool, or the DDAP.

2.6.5 Communication Middleware

The communication middleware is the software component in charge of the exchange of information between services and other devices. This component offers an abstraction over the communication protocols and physical devices used.

Inter-Service Communication

For communication between services, the middleware offers a standard MQTT broker. MQTT is an IoT-oriented pub-sub communication protocol built over TCP/IP. For example, a service which is getting readings from a temperature sensor can publish data to a topic (e.g. /sensor/temperature/data). Other services can subscribe to that same topic in order to get updates of the temperature in real time.

External Communication

The communication between services and other devices (other edge nodes, cloud, etc.) can be separated in different data streams depending on the desired QoS. Different levels of QoS can be defined with different requirements in order to choose between communication interfaces or protocols. For example, a high priority data stream can be mapped to the 4G/LTE modem. A bulk-data data stream (e.g. high volume of data for offline analysis, etc.) can be transferred when Wi-Fi connectivity is available, because this is not a critical data with real-time constraints.

3 Conclusions

Big data analytics have become a key enabling technology across multiple application domains, to address societal, economic and industrial challenges for safe mobility, well-being and health, sustainable production and smart manufacturing, energy management, etc. A particular challenge for big data analytics in the near future (or even today) is managing large and complex real-world systems, such as production lines, fleets of public transportation and even whole cities, which continuously produce large amounts of data that need to be processed on the fly. Providing the required computational capacity level for absorbing extreme amounts of complex data, while considering non-functional properties, is of paramount importance to allow converting the data into few concise and relevant facts that can be then consumed by humans and be decided or acted upon. The ELASTIC project [1] is facing this challenge by proposing the end-to-end software framework described in this chapter. The final goal is to efficiently distribute extreme-scale big data analytic methods across the compute continuum, to match performance delivered by the different computing resources with the required precision and accuracy.

Acknowledgments The research leading to these results has received funding from the European Union's Horizon 2020 Programme under the ELASTIC Project (www.elastic-project.eu), grant agreement No 825473.

References

1. A software architecture for extreme-scale big-data analytics in fog computing ecosystems (ELASTIC) (2020). https://elastic-project.eu/ [Online; accessed October 2020].
2. Apache Foundation (2017). Apache kafka. https://kafka.apache.org/ [Online; accessed October 2020].
3. Apache Foundation (2020). Apache druid. https://druid.apache.org/ [Online; accessed October 2020].
4. Barcelona Supercomputing Center (BSC). COMP Superscalar (COMPSs). http://compss.bsc.es/ [Online; accessed October 2020].
5. EdgeXFoundry (2020). https://www.edgexfoundry.org/ [Online; accessed November 2020].
6. EU General Data Protection Regulation. https://ec.europa.eu/info/law/law-topic/data-protection/eu-data-protection-rules_en [Online; accessed November 2020].
7. IKERLAN (2020). IKERLAN KONNEKT. https://www.ikerlan.es/en/ikerlankonnekt [Online; accessed October 2020].
8. Martí, J., Queralt, A., Gasull, D., Barceló, A., Costa, J. J., & Cortes, T. (2017). Dataclay: A distributed data store for effective inter-player data sharing. *Journal of Systems and Software, 131*, 129–145.
9. OpenScap (2020). https://www.open-scap.org/. [Online; accessed November 2020].
10. SixSq (2020). Edge and container management software. https://sixsq.com/products-and-services/nuvla/overview. [Online; accessed October 2020].
11. SixSq (2020). Secure and intelligent edge computing software. https://sixsq.com/products-and-services/nuvlabox/overview. [Online; accessed October 2020].

12. Tejedor, E., Becerra, Y., Alomar, G., Queralt, A., Badia, R. M., Torres, J., Cortes, T., & Labarta, J. (2017). Pycompss: Parallel computational workflows in Python. *The International Journal of High Performance Computing Applications, 31*(1), 66–82.
13. Zillner, S., Curry, E., Metzger, A., & Seidl, R. (Eds.) (2017). European big data value strategic research and innovation agenda. https://bdva.eu/sites/default/files/BDVA_SRIA_v4_Ed1.1.pdf. [Online; accessed October 2020].
14. Zillner, S., Bisset, D., Milano, M., Curry, E., Robles, A. G., Hahn, T., Irgens, M., Lafrenz, R., Liepert, B., O'Sullivan, B., & Smeulders, A. (Eds.) (2020). Strategic research and innovation agenda—AI, Data and Robotics Partnership. Third Release. https://ai-data-robotics-partnership.eu/wp-content/uploads/2020/09/AI-Data-Robotics-Partnership-SRIDA-V3.0.pdf. [Online; accessed February 2021].