Guaranteed Latency Applications in Edge-Cloud Environment

Petr Hnetynka
Charles University,
Faculty of Mathematics and Physics
hnetynka@d3s.mf.cuni.cz

Petr Kubat
Charles University,
Faculty of Mathematics and Physics
kubat@d3s.mf.cuni.cz

Rima Al-Ali
Charles University,
Faculty of Mathematics and Physics
alali@d3s.mf.cuni.cz

Ilias Gerostathopoulos
Fakultät für Informatik,
Technische Universität München
gerostat@in.tum.de

Danylo Khalyeyev
Charles University,
Faculty of Mathematics and Physics
khalyeyev@d3s.mf.cuni.cz

ABSTRACT

Modern Cyber-Physical Systems combine distributed embedded devices with computation in cloud. The inclusion of cloud increases the smartness of the systems by allowing computationally- and data-intensive tasks such as complex data analytics, optimization and decision making, learning and predictions. However, an important implication of interacting with the physical world is the presence of real-time requirements. It puts the cloud in the loop and requires the cloud to participate in the overall real-time guarantees, which poses a difficult problem. In this paper, we address the problem of providing real-time guarantees (in particular statistical guarantees on response time) by combining edge-cloud processing with runtime performance awareness and adaptation.[1]

CCS CONCEPTS

• Computer systems organization → Architectures → Distributed architectures → Cloud computing

KEYWORDS

CPS, edge-cloud, guaranteed latency, adaptation

ACM Reference format:

Petr Hnetynka, Petr Kubat, Rima Al-Ali, Ilias Gerostathopoulos, Danylo Khalyeyev. 2018. Guaranteed Latency Applications in Edge-Cloud Environment. In *12th European Conference on Software Architecture: Companion Proceedings (ECSA '18), September 24--28, 2018, Madrid, Spain, 4 pages.* https://doi.org/10.1145/3241403.3241448

ECSA '18, September 24–28, 2018, Madrid, Spain

ACM ISBN 978-1-4503-6483-6/18/09…$15.00
https://doi.org/10.1145/3241403.3241448

1 INTRODUCTION

Modern Cyber-Physical Systems (CPS) combine distributed embedded devices with computation in cloud. This creates an integrated computation, sensing and control fabric that crosses the boundary between physical and virtual worlds. Such systems include applications of smart traffic, smart agriculture, smart power grid, etc. The extent of these systems typically spans large areas (countries, continents) and includes thousands or millions of devices (sensors, smartphones, machines, etc.). The inclusion of cloud increases the systems' smartness by allowing computationally- and data-intensive tasks such as complex data analytics, optimization and decision making, learning and predictions.

An important implication of interacting with the physical world is the presence of real-time requirements. These come in the form of end-to-end response time requirements for control, decision making and visualization. The increasingly strong pressure on providing smart behavior means that even the real-time tasks, which have been traditionally performed only locally on embedded devices, need to include computation on the cloud. This, however, puts the cloud in the loop and requires the cloud to participate in the overall real-time guarantees, which poses a difficult problem.

Related work in this area addresses the general unsuitability of a cloud to give any real-time guarantees by reducing communication latencies by bringing the cloud closer to user. These techniques come in the form of cloudlets [12], fog-cloud [3], edge-cloud [13], and cyber-foraging [9]. The approaches have been further elaborated in multiple directions. Cachier [4] is a system that inserts a specialized cache on the edge-cloud and thus minimizes the communication latency with the cloud. While the approach is functional (3 times increased responsiveness), it is suitable only for applications where data can be cached. The PreCog [5] system further extends the Cachier idea and uses prediction (based on Markov chains) to prefetch data onto the devices and thus further reducing latency (up to 5 times). Both approaches, however, work on best effort, i.e., they do not provide any real-time guarantees. An approach to reduce latency in cloud-enabled IoT systems (that are commonly considered as a special case of CPS) is Fog Computing Architecture Network (FOCAN) [10]. It reduces latency primarily by splitting

the system into multiple regions and tiers and introducing multiple types of communications in order to avoid unnecessary data transfer. Primary intent of the system is to optimize energy consumption of mobile services. While FOCAN targets primarily smart-city scenarios, the authors extended the approach into Fog of Everything (FoE) [1], targeting energy optimizations.

A completely different approach to reduce latency is described in [6] and employs traffic offloading from cellular to other types of networks, e.g., WiFi. The core element of the approach is Mobile Cloud Offloading Helper that resides in mobile operator network and uses feedback from the network, application requirements and user behavior to provide decisions for traffic offloading. The approach targets guarantees in communication (as our approach), however it needs multiple available networks. A similar approach is described in [11], where based on analyses of context, parts of the application are offloaded. However, the approach does not provide real-time guarantees.

Goal: In this position paper, we address the problem of providing response time guarantees by combining edge-cloud processing with runtime performance awareness and adaptation. The key idea is that by automatically pre-assessing an application, continuously monitoring its performance and combining that with prediction based on historical performance observations, we can control service (re-)deployment in an edge-cloud such that we can give statistical guarantees on end-to-end response time.

Contrary to cyber-foraging and related approaches, which typically work in a rather decentralized and client-driven manner, we specifically emphasize the cloud-centric control and deployment of services. This allows us to provide a solution that remains as close as possible to the traditional declarative way of deploying and orchestrating services in the cloud, which is already familiar to cloud developers. Essentially, we only extend the cloud deployment specification by timing requirements and description of how to validate the timing.

We showcase our approach on an augmented reality application and describe its integration with Kubernetes (K8S) PaaS. We provide indicative evaluation based on the individual pieces of the solution we have created so far.

2 RUNNING EXAMPLE

As a real-life example of an application benefiting from the guaranteed communication latencies with a cloud, we consider a simple yet realistic augmented reality application, which is one of the use-cases in the ECSEL JU project FItOptiVis[2] in which we participate and whose main objective is to develop a distributed image- and video-processing pipeline for CPS.

The user-interface part of the application runs on a mobile phone and captures a video stream (via the phone camera), which is not only displayed on the screen but also sent to the video processing part running in the cloud. The processing part analyzes the video and sends back to the client "augmentation" information that is drawn by the client to the displayed stream. For such a scenario, the guaranteed communication latency is crucial as it is necessary to augment the displayed stream without any significant delay. Thus, in ideal case, the processing part should be as close to the client as possible, i.e., in the edge cloud.

Namely, our application analyzes the video stream for human faces and tries to identify particular persons using a trained face recognizer. From the user point of view, the application draws rectangles around the faces and displays a name of the particular person—the rectangles and names are computed by the processing part in the cloud.

[2] https://www.ecsel.eu/projects/fitoptivis

3 STRUCTURE AND REQUIREMENTS

We structure our approach, which provides probabilistic real-time guarantees for edge-cloud applications, in three main parts. These parts also respectively address the three principal research questions tackled by our approach:

1. *How to specify the probabilistic real-time guarantees for cloud applications?* The main concerns here are that: (a) such specification should be in line with the existing way of specifying deployment of microservice architectures in cloud settings; (b) such specification should establish an automatically verifiable and guaranteeable contract between the application developer and cloud provider.
2. *How to assess that the cloud can guarantee the real-time response?* Before deploying the application to the cloud, we assess if the requested real-time guarantees can be provided. If so, we admit the application to the cloud. The main concern here is that this assessment procedure should be performed automatically by the cloud without human intervention. Ideally, this should also build knowledge of how the application behaves from timing perspective and establish strategies for providing the guarantees.
3. *How to guarantee the end-to-end response time at runtime?* The main concerns here are that (i) we need to pool resources among tenants such that the cloud is well utilized and that (ii) the load in the cloud changes over the time depending on the applications deployed in the cloud and mobility of users. (In edge-cloud settings we assume communication with an edge-cloud datacenter close by.)

3.1 Specifying the real-time guarantees

In our approach, we use declarative specification of an auto-scaling microservice architecture. We extend this specification by the specification of measurement probes and real-time requirements over the probes.

The measurement *probes* are specific functions provided by the application developer that perform a performance test. In case of our running example, this test consists of recognizing faces in a pre-defined image. The important features of the probe are that: (a) it does not take any inputs, (b) it can be measured anytime (even at runtime), and (c) it measures an operation which is in strong correlation to what needs to be guaranteed. Thanks to (a) and (b), the cloud can execute the performance measurements by itself at any time it needs to assess the application or check if the guarantee is still upheld. Thanks to (c) the

application developer can reliably express the real-time requirements over the probes rather than the real operations.

The fact that we express the real-time requirements over the probes instead over the real operations constitutes one of the key novelties of our work. This allows us to form a contract between the developer and the cloud that can be assessed before runtime and then proactively and speculatively checked at runtime without affecting the functionality of the application. In this way, we move the responsibility of understanding the application to the developer and avoid having a human inspect the application on the side of the cloud.

To invoke the probes, we expect each microservice to integrate a special library (called *application agent*) that is a part of our approach. The application agent is intended to execute the probes locally and provide the measured data to our infrastructure. The application agent exposes a remote interface (REST or gRPC[3]), which allows the rest of our orchestration infrastructure request and collect measurements when required.

To give technical grounding to our approach, we use K8S PaaS. Figure 1 shows a fragment of the deployment descriptor in

```
kind: Deployment
metadata:
  name: recognizer-deployment
  labels:
    app: recognizer
spec:    # micoservices specification
  template:
    metadata:
      labels:
        app: recognizer
    spec:
      containers:
      - name: recog
        image: d3srepo/recog
        ports:
        - containerPort: 7777
probes:  # probes
- name: recognize
timingRequirements:  # timing require-
ments
- name: recognize limit
  probe: recognize
  limits:
  - probability: 0.999
    time: 50 # Max. 50ms in 99.99% cases
  - probability: 0.99
    time: 30 # Max. 30ms in 99.9% cases
```

Figure 1 Deployment descriptor.

YAML data format of the running example. The specification defines the architecture in terms of (a) microservices, which are further specified using K8S DeploymentSpec[4], (b) probes that can be executed inside a microservice, and (c) timing requirements.

3.2 Assessment of a cloud application

Compared to the traditional deployment procedure in the cloud, we insert an extra step before the actual deployment as typically done in real-time scenarios. This step assesses the application to check the feasibility of real-time requirements specified in its deployment descriptor. The deployment procedure thus looks as follows: (1) the application developer packages the application (in our case creates Docker images of individual microservices and writes the deployment descriptor as described in Section 3.1), (2) the developer submits the application to the cloud, (3) the cloud assesses the application (as described below), (4) the cloud informs the developer whether the application can be admitted and optionally informs the developer about the price of the deployment (we assume that the price depends on how stringent the real-time requirements are and hence on how much resources have to be reserved as opposed to shared), and (5) the developer accepts the price and confirms the deployment.

The assessment phase consists of running the application in a staging environment and repeatedly measuring the probes. Here we exploit the performance benchmarking methodology developed in our department [2], in which we execute multiple runs, each consisting of performing warmup and then collecting a number of observations (i.e., invocation of a probe). Each run starts with a restart of the computing node. This addresses the variability caused by probabilistic decisions in the underlying operating system and system libraries.

We measure the application in isolation and with pre-defined typified background workload (i.e., IO-bound, CPU-bound workload). This gives us rough estimate of how the application should work when sharing resources with other applications. We further refine these results by collecting system counters (e.g., instruction count, cache miss count, input/output operations per second, VM utilization) during the run of each probe. This allows us to categorize what kind of computation (from resource utilization perspective) a particular probe does.

We use all the measured data to build a statistical regression model similar to our previous work [7], where we exploited factorial ANOVA to give statistical guarantees in controlling system experimentation. In this context, the execution time we measure for a probe constitutes a dependent variable. The presence of a typified workload constitutes qualitative independent variables. The observation of system counters constitutes quantitative independent variables.

The regression model allows us to gradually build knowledge about the performance of applications from data collected in the assessment phase and at runtime—Section 3.3. The regression over the collected data then allows performing what-if analysis on how an application is likely to behave in presence of other applications. This is important (a) for making the decision about admitting the application, and (b) for controlling redeployment/scaling at runtime. Thanks to use of the regression model, we can gradually improve precision of the what-if analysis as more observations in different situations are collected.

3.3 Providing guarantees at runtime

The principal problem of providing the real-time guarantees is that all the software stack we use provides no real-time guarantees at all. Consequently, our guarantees are probabilistic—e.g., "below 50ms in 99.9% and below 30ms in 99%". This allows us to tolerate uncontrollable lags in software stack as long as we in the long-term stay within the limits. To do so, we exploit self-adaptation architecture and the MAPE-K loop [8] for controlling the deployment and redeployment in the cloud.

From the architectural perspective, we treat the existing cloud infrastructure (K8S in our case) as the system under control and we provide a *controller* which controls the redeployment in the cloud using the REST API it provides. Technically, we assign a distinct label to each node in the cloud to have a relatively fine-grained control over the placement of microservices to nodes. We place a dedicated *node agent* on each node to collect

[3] https://grpc.io/
[4] https://kubernetes.io/docs/concepts/workloads/controllers/deployment/

system counters and to set scheduling parameters and control CPU reservation, which are all actions we cannot observe and control directly via K8S API. Additionally, as already mentioned in Section 3.1, we place an *application agent*, in each microservice to collect data from the probes. Figure 2 shows the high-level architecture of our approach.

From the MAPE-K perspective, we perform the following steps:

1. *Monitoring*—The controller collects data about the current deployment (from K8S) and about the node utilization (via node agents). Further, it regularly collects measurements from probes (via application agents) and follows the mobility of users, which determines which microservices will have to be deployed at which cloud edges.
2. *Analysis*—The controller builds a constraint optimization problem describing on which particular nodes microservices could be placed and if and which resources should be reserved for them (e.g., dedicated CPU core or dedicated IO bandwidth). The what-if analysis over the regression model (described in Section 3.2) is used to assign costs to respective alternatives. The model also incorporates cost of reallocation to provide a certain level of affinity in decisions. Technically, we are experimenting with a Constraint Satisfaction Problem solver for this step, however the scalability issues suggest that some variant of linear programming might be a better choice.
3. *Planning*—Application of the new configuration selected in analysis is transformed to a flow of tasks similar to UML activity diagram. Each of these tasks describes a particular action on K8S or node agent API and captures synchronization among these tasks.
4. *Execution*—This step executes the tasks planned in the Planning phase. The result of this is a redeployment of microservices in the cloud to satisfy the probabilistic real-time constraints.

4 EVALUATION AND CONCLUSION

In this position paper we overviewed our approach to providing statistical guarantees on response time of (edge-)cloud applications. Our solution extends the typical cloud deployment specification with real-time constraints. Contrary to the practice nowadays, this allows a developer to express real-time constraints directly instead of having to figure out how much CPU and IOPS they have to reserve (as it is common for instance in Amazon AWS). Further, it gives the cloud more leeway in deciding where to place the workload and which resources to reserve (e.g., CPU core, IO/network bandwidth, etc.). In our approach, we combine the performance awareness with statistical methods to give guarantees and with adaptation methods to ensure that guarantees are kept in face of changing conditions. The key idea of our approach is automatic pre-assessment of an application, which allows automatic decision if guarantees can be given, and builds a knowledge model that is used by the adaptation loop at runtime to make adaptation decisions.

Being this a position paper, we do not have a complete framework yet. However, we build our indicative evaluation of feasibility of the approach on our previous work and on several

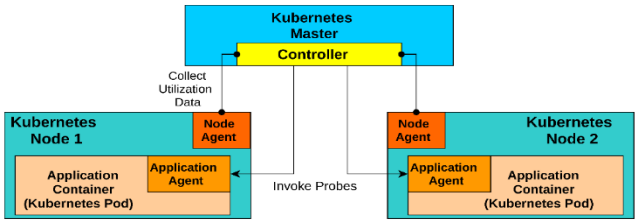

Figure 2 High level architecture.

experiments and prototypes that we have already built for our approach. In particular, in the statistical area and performance awareness we build on our previous experiments that suggest that an application can be at least to a certain extent categorized based on how it utilizes resources and that factorial ANOVA can be successfully used to provide such categorization. We also did experiments with real-time augmented reality computed in application containers, where we observed that such computation is performance-wise stable enough to easily provide 99.9% statistical guarantees on response time even with background load. Though our approach is general, we grounded it technologically in K8S (a container-based cloud). We have already created a control architecture over K8S and prototypes of modules for orchestrating and performing the performance assessment.

ACKNOWLEDGMENTS

The research leading to these results has received funding from the ECSEL Joint Undertaking (JU) under grant agreement No 783162. Also, this work has been partly funded by the Bavarian Ministry of Economic Affairs, Energy and Technology as part of the TUM Living Lab Connected Mobility project.

REFERENCES

[1] Baccarelli, E. et al. 2017. Fog of Everything: Energy-Efficient Networked Computing Architectures, Research Challenges, and a Case Study. *IEEE Access*. 5, (2017), 9882–9910.

[2] Bulej, L. et al. 2017. Unit testing performance with Stochastic Performance Logic. *Automated Software Engineering*. 24, 1 (Mar. 2017), 139–187.

[3] Dastjerdi, A.V. and Buyya, R. 2016. Fog Computing: Helping the Internet of Things Realize Its Potential. *Computer*. 49, 8 (Aug. 2016), 112–116.

[4] Drolia, U. et al. 2017. Cachier: Edge-Caching for Recognition Applications. *Proceedings of ICDCS 2017, Atlanta, USA* (Jun. 2017), 276–286.

[5] Drolia, U. et al. 2017. Precog: PRefetching for Image RecogNition Applications at the Edge. *Proc. of SEC '17, San Jose, USA* (2017), 17:1–17:13.

[6] Fiandrino, C. et al. 2015. Network-assisted offloading for mobile cloud applications. *Proc. of ICC 2015, London, UK* (2015), 5833–5838.

[7] Gerostathopoulos, I. et al. 2018. Adapting a System with Noisy Outputs with Statistical Guarantees. *Proceedings of SEAMS 2018, Gothenburg, Sweden* (May 2018).

[8] Kephart, J. and Chess, D. 2003. The Vision of Autonomic Computing. *Computer*. 36, 1 (2003), 41–50.

[9] Lewis, G.A. and Lago, P. 2015. A Catalog of Architectural Tactics for Cyber-Foraging. *Proc. of QoSA 2015, Montreal, Canada* (2015), 53–62.

[10] Naranjo, P.G.V. et al. 2017. FOCAN: A Fog-supported Smart City Network Architecture for Management of Applications in the Internet of Everything Environments. *arXiv preprint arXiv:1710.01801*. (Oct. 2017).

[11] Orsini, G. et al. 2018. Generic context adaptation for mobile cloud computing environments. *J. of Ambient Intelligence and Humanized Computing*. 9, 1 (2018), 61–71.

[12] Satyanarayanan, M. et al. 2009. The Case for VM-Based Cloudlets in Mobile Computing. *IEEE Pervasive Computing*. 8, 4 (Oct. 2009), 14–23.

[13] Satyanarayanan, M. 2017. The Emergence of Edge Computing. *Computer*. 50, 1 (Jan. 2017), 30–39.