Edge-to-Cloud Solutions for Self-Adaptive Machine Learning-based IoT Applications

A Cost Comparison

Marco Emilio Poleggi, Nabil Abdennadher, Raoul Dupuis, and Francisco Mendonça

ISC department
University of Applied Sciences and Arts, Western Switzerland
Geneva, Switzerland
{marco-emilio.poleggi, nabil.abdennadher, raoul.dupuis, francisco.mendonça}@hesge.ch

Abstract. Large-scale IoT applications based on machine learning (ML) demand both edge and cloud processing for, respectively, AI inference and ML training tasks. Context-aware applications also need self-adaptive intelligence which makes their architecture even more complex. Estimating the costs of operating such edge-to-cloud deployments is challenging. To this purpose, we propose a reference service-oriented event-driven system architecture for IoT/edge applications comprising a minimal set of components, mapped on available cloud services. We then propose a resource consumption model for estimating the cost of deploying and running self-adaptive AI-assisted IoT applications on selected edge-to-cloud platforms. The model is evaluated in two scenarios: Road Traffic Management and Smart Grid. We finally provide some estimates showing how the expenditure breakdown varies significantly depending on the adopted platform: storage costs are dominant in Road Traffic Management for all providers, whereas either messaging or edge management costs may dominate the Smart Grid scenario, and, surprisingly, computing costs are almost negligible in all cases.

Keywords: Edge · Cloud · IoT · Cost Model · PaaS.

1 Introduction

Nowadays, the Cloud is being massively adopted for a plethora of applications, many of which also need some components deployed at the Edge, in charge of governing large-scale IoT sensor networks: we refer to these as *edge-to-cloud* IoT deployments. A particular subclass of these IoT applications is based on machine learning (ML) to accomplish inference tasks on data coming from IoT sensors: traffic monitoring systems, environmental sensing, fleet management and asset tracking, as well as smart grid appliances. Because of their distributed nature over edge devices with constrained resources, these applications leverage the

Cloud for (heavy, long-term) learning tasks, while exploiting edge devices for (light, low-latency) inference tasks on data coming from nearby IoT sensors.

Many IoT applications also exhibit high context sensitivity, as their intelligence has to cope with different physical settings, complex usage patterns as well as varying meteorological conditions. All this demand "context-aware" solutions, which, for optimized performance, would follow a "self-adaptive" paradigm, as illustrated in Figure 1: an end-user application consumes the output of an edge device that processes data coming from some IoT sensors. An artificial intelligence-based (AI) inference module is deployed on this edge device, which makes "predictions" on the sensing data: e.g., classify environmental sounds, recognize car license plates, etc. The AI inference module is endowed with a machine learning model (MLM) specifically tailored to the application at hand and optimized for the edge device. This MLM is built and trained in the Cloud, where an AI learning module is deployed.

Such applications operate in the following way. During a bootstrap phase, a raw dataset is fed to the Cloud where it is first labeled and then processed for AI learning. That results in a first MLM which is deployed to the edge device. Then, the system enters operation mode: a feedback loop enabling continuous intelligence adaptation. The edge device autonomously processes IoT sensors' data; two cases may occur:

a. The prediction is satisfying: application output is provided.
b. The prediction is not satisfying: no application output is provided; the related sensing input is uploaded to the Cloud as "low-performance" data. These are labeled and fed for training to the AI learning module: a new MLM is generated which is then redeployed to the edge device. The original dataset is extended with the new labeled data.

Several commercial actors already offer edge-to-cloud solutions (platforms) suitable for our scenario. Given the complexity of such three-tiered (IoT, Edge and Cloud) architectures, estimating their operating costs is not trivial. In a previous work of ours [3], we presented a cloud application placement tool: a decision-support system that optimally selects a cloud provider for an application, based on current prices fed to a resource consumption model (RCM). With this paper, we go beyond the *pure* cloud application paradigm and provide a comparative study of edge-to-cloud platforms for self-adaptive machine learning-based IoT applications. Indeed, we are interested in answering the specific question: *how much does it cost to deploy and operate a generic ML-based IoT application on a given platform?*

For comparison purposes, we consider five platforms: three well-known that employ proprietary solutions (Amazon AWS [1], Google Cloud [4] and Microsoft Azure [8]) and two newcomers that are based on (mostly) open-source software (SixSq Nuvla [11] and Balena [2]).

The rest of the paper is structured as follows. We review some related work in Section 2. A detailed description of our use case is provided in Section 3 where a reference system architecture is also proposed. Then, Section 4 presents our

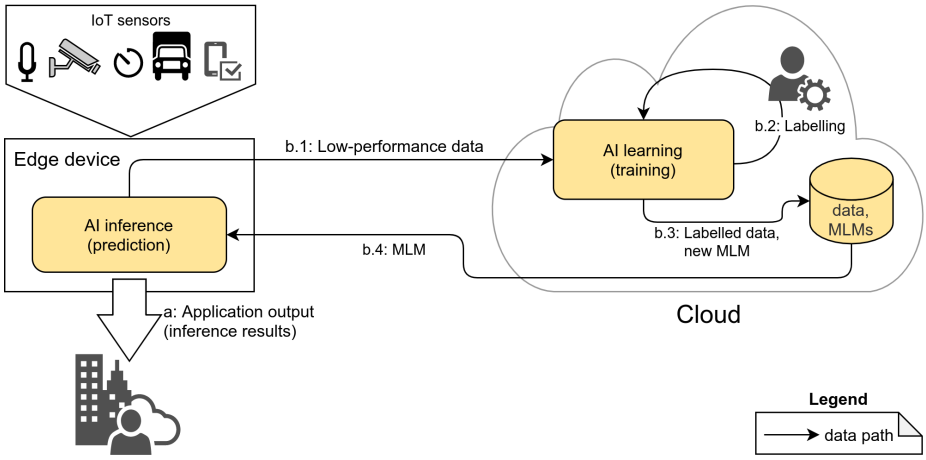

Fig. 1. A self-adaptive ML-based edge-to-cloud application scenario (operation feedback loop).

RCM based on which we compare the cost of the different edge-to-cloud solutions under study (Section 5). Finally, we draw our conclusions in Section 6.

2 Related work

Several research works aim to estimate the placement or operating costs of end-user applications on different cloud service providers. To the best of our knowledge, none has yet proposed an all-comprehensive cost model for complex IoT applications.

Martens et al. [7] tackle the case of customers who own their IT infrastructure and want to compare public cloud offerings. They propose a mathematical method to compute the Total Cost of Ownership (TCO) of cloud computing services. This work set the basis for our previous RCM [3], which is extended here to especially deal with ML-based IoT/edge applications.

Laatikainen et al. [6] study the different pricing models of many IaaS, PaaS, and SaaS cloud providers. They realize that, despite the heterogeneity and complexity of the models, common offering patterns exist, however no established normalized price strategy is available. Thus, the authors propose an extended SBIFT-based pricing model customized for generic cloud services.

Nguyen et al. [9] study the optimal placement of IoT/edge applications in cloud systems employing virtual network functions. Based on a system architecture with focus on the IoT network topology, they define an analytical cost model in terms of computation resources and network bandwidth, as well as algorithms for small and large-scale network settings. Our analysis is instead focused to the cost of edge-to-cloud service architectures.

With the goal of simulating an infrastructure-agnostic IoT/fog application placement method, Goudarzi et al. [5] propose a novel Memetic algorithm which minimizes the operating costs in terms of execution time and energy consumption. Local IoT computation is compared to both Edge and Cloud offloading scenarios. Conversely, we do not consider IoT as computing devices, but propose a more refined Ege/Cloud resource consumption model that encompasses storage and device management services.

3 A reference service-oriented architecture

We consider a generic IoT use case in which an MLM is first trained in the Cloud on a labeled dataset and then deployed to the edge devices performing the AI inference. In order to estimate the operating costs of such applications, we propose, as a reference, the service-oriented architecture depicted in Figure 2: a platform-agnostic edge-to-cloud deployment whose implementation technology may vary across different service providers.

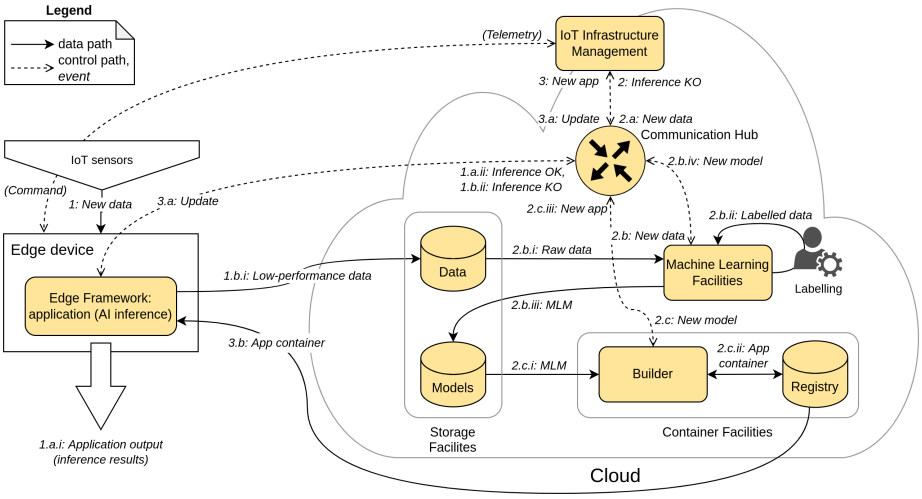

Fig. 2. Reference architecture for a generic self-adaptive ML-based IoT application (edge-to-cloud deployment). Event labels are numbered according to the application workflow described below.

The minimal set of necessary service components is:

IoT Infrastructure Management to compose, provision and monitor Edge / IoT networks. Edge devices and their companion IoT devices are mostly autonomous; their Edge Framework modules are packaged into some form and provisioned to them from a Cloud repository. As a best practice, a

dedicated control path would be used for IoT Infrastructure Management operations–especially for monitoring and telemetry tasks.

Edge Framework to enable edge modules programming and execution as micro services. We assume that Edge Framework artifacts (OS and application modules) are provisioned as Docker containers–possibly the most popular way of deploying micro services without resorting to provider-specific services like those based on Functions-as-a-Service (FaaS).

Container Facilities to build a Docker container with a trained MLM and possibly other Edge Framework artifacts, and to store it in a Cloud registry. Edge devices would then pull containers directly from the registry. A full-fledged *builder* component would enable automatic (event-driven) (re)building of containers.

Communication Hub to create an event-based messaging service among the different application's modules: the reception of an event triggers a workflow operation.

Storage Facilities to store labeled training data and several MLM versions in Cloud data warehouses.

Machine Learning Facilities: to build and train a MLM in the Cloud. In the case of a supervised learning strategy, we assume that the associated labeling task is performed by humans.

The service-oriented architecture depicted in Figure 2 is event-based and data-driven. Once the system bootstrap phase has been performed, the application workflow follows the logic described in Algorithm 1 (Edge) and Algorithm 2 (Cloud).

See [10] for a detailed discussion of how the different platforms cover the above components, and how their respective service-level offerings compare to each other.

4 Resource consumption model

In order to estimate the operating cost of a self-adaptive ML-based IoT application, we need to further specify the edge application. We are interested in large-scale scenarios characterized by different and varying operating conditions that cannot be tackled with a unique, static MLM configuration. Thus, we need an edge AI that is 1. tailored to different contexts by leveraging several MLMs (context-awareness), and 2. continuously improved over time through recurrent retraining of its MLMs (self-adaptation). For the sake of clarity, we consider two example scenarios: Road Traffic Management and Smart Grid. In the former case, camera-less sound sensors perform vehicle classification according to the noise they generate; this application is context-aware since the noise generated by a vehicle depends on several aspects: wet/dry weather, pavement type, street configuration, surroundings, etc. In the latter case, edge devices are deployed in households to predict electrical energy consumption and production; this application is context-aware because the prediction depends on different settings: urban/rural, season, weather, weekdays/weekend, etc.

Algorithm 1: Edge Processing

```
task Application:
    Data: infThreshold
    forever do
1.      newData ← edge.IoTSensor.read()
        (appResult, confidence) ← edge.AI.doInference(newData)
        if confidence > infThreshold then
            // The inference is satisfying
1.a.i       edge.sendUserOutput(appResult)
1.a.ii      edge.event.send('Inference_OK')
        else
            // The inference is *not* satisfying
1.b.i       edge.cloudStorage.put(newData)
1.b.ii      edge.event.send('Inference_KO')
task Provisioning:
    // Triggered by 'Update' events (3.a.) from the Cloud Processing
       Algorithm 2
3.b.    newAppContainer ← edge.cloudStorage.get()
    edge.AI.doUpdate(newAppContainer)
```

From an IoT perspective, groups (or clusters) of sensors are expected to belong to different spatio-temporal contexts, each needing its own MLM configuration. A context is defined in terms of some features, such as location, time of the day, season, weather conditions, surroundings, etc.–anything that may affect the MLM performance. Our scenarios are based on the following assumptions:

- Each context has exactly one specialized MLM–this is the simplest configuration.
- Each sensor belongs to one context at any time, but can *shift* from one context to another over time. Sensors belonging to the same context are expected to exhibit a similar behavior. We deal with context shift by simulating periodical retraining of a fixed fraction of the MLMs.
- Each sensor is connected to only one edge device to which it reports measurement events. Any edge device support as many MLMs as are the contexts of the sensors connected to it. When receiving an event from a given sensor, the corresponding MLM is triggered in the edge device, as explained in Figure 1.

The model considers the fees for registering the edge devices and is composed of sub-models for, respectively, exchanged control messages (application and telemetry, excluding data), storage (data at rest and operations), data transfers (network usage for data moving between the Edge and the Cloud) and computing. The model's parameters are listed in Table 1: the actual values depend on the application; they are detailed in [10]. For each sub-model, we compute the application demand (workload) for one edge device. Then, for each platform under study, we chose the service resources (storage, messaging, computing, etc.)

Algorithm 2: Cloud Processing

```
   Data: maxNewDataCount
   forever do
       newEvent ← cloud.event.receive()
       switch newEvent do
2.         case 'Inference_KO' do
               // In IoT Infrastructure Managament
2.a.           cloud.event.send('New_Data')
               cloud.dataSet.newDataCount += 1
           case 'New_Data' do
               // In Machine Learning Facilities
2.b.           if cloud.dataSet.newDataCount ≥ maxNewDataCount then
2.b.i              rawDataBatch ← cloud.cloudStorage.get()
2.b.ii             labeledDataBatch ← cloud.labelling(rawDataBatch)
2.b.iii            newMLM ← cloud.MLTraining(labeledDataBatch)
2.b.iv             cloud.cloudStorage.put(newMLM)
           case 'New_Model' do
               // In Container Facilities
2.c.i          newMLM ← cloud.cloudStorage.get()
2.c.ii         newAppContainer ← cloud.containerBuilder(newMLM)
               cloud.containerRegistry.put(newAppContainer)
2.c.iii        cloud.event.send('New_App')
3.         case 'New_App' do
               // In IoT Infrastructure Management, triggers Edge's
                  'Provisioning' task in Algorithm 1
3.a.           cloud.event.send(Update')
```

that satisfy the demand at the cheapest offering, so as to obtain a normalized price list. Based on these prices, we compute the cost of using the chosen resources on a large-scale deployment.

With reference to Figure 2, the edge intelligence workflow is simulated as follows:

1. **Bootstrap.** Sensors are clustered on a given feature set, resulting in a set of MLMs (`ml_contexts`). The corresponding MLMs are provisioned to the edge devices. The costs associated with the bootstrap phase are not considered because they are negligible ($< 1\%$) compared to those of the operation phase.
2. **Operation.** The sensor network is started: the process is now event-driven. At a frequency of `event_rate`, each edge device is fed a "raw" sensing data item, each of size `raw_data_size`, and outputs one "application" data item, each of size `app_output_size`.
   a. **ML inference.** Inference operations are unsuccessful with probability `ml_error_rate`. Accordingly, a certain amount of raw data (in addition to the application data) is uploaded to the Cloud and stored there.

Table 1. Cost model's parameters.

Parameter	Description
`event_rate`	Rate at which the MLM is triggered
`raw_data_size`	Size of a raw data item fed to the MLM
`app_output_size`	Size of an application output item produced by the MLM
`ml_error_rate`	Fraction of events which the MLM is unable to classify or predict
`ml_model_size`	Size of the MLM
`ml_contexts`	Number of contexts: each context needs a specific MLM
`ml_point_size`	Size of a data point used to train the MLM
`ml_train_size`	Number of data points used to train the MLM
`ml_train_time`	Computing time needed for training the MLM on 1 vCPU
`ml_train_rate`	Rate of MLM training rounds in the Cloud
`ml_underperf`	Fraction (i.e., the "nonperforming") of all MLMs that must be retrained at each round
`ml_train_deadline`	Maximum allowed time to train all nonperforming MLMs at each round. This sets the minimal number of needed computation resources.
`edge_img_size`	Size of the system package (OS, containers, MLM and libraries) deployed to any edge device
`daily_connect_time`	Number of minutes per day during which an edge device is connected to the Cloud
`deployment_size`	Number of edge devices deployed
`tmetry_metrics`	Number of telemetry metrics collected at the edge devices
`tmetry_msg_rate`	Rate at which telemetry messages are sent form the edge application to the Cloud
`tmetry_msg_size`	Size of a telemetry message

b. **Continuous learning**. MLMs whose prediction scores fall persistently below a given threshold (which we do not specify) are said "nonperforming": they are a fraction `ml_underperf` of the total. We assume that the related MLM is retrained in the Cloud at frequency `ml_train_rate`.

4.1 Messaging model

As noted above, a well-designed edge application would use separate channels for application messages and infrastructure control messages. Hence, we consider two classes of messages: application and telemetry (which includes monitoring

and logging) that are respectively handled by two different services. Notice that network usage for data transfers is excluded here.

Application messages are routed through the Communication Hub component. Telemetry messages may be conveyed through a dedicated service subcomponent of the IoT Infrastructure Management, where available; else the same Communication Hub is used. Application messages are exchanged for any operations that have to be performed in the Cloud.

4.2 Storage operation model

We consider the used space (data at rest) and the rate of operations performed in the Cloud storage service. Here we only attempt a rough estimate of the workload–a detailed analysis is beyond the scope of this paper. We assume that:

- Application data are uploaded to the Cloud for each MLM execution. Moreover, raw data are uploaded to the Cloud on each unsuccessful inference.
- Each upload of raw and application data to the Cloud triggers one write-like operation.
- To generate a new MLM, a set of data points, each of size `ml_point_size`, is
  1. first, transferred (read-like) from Cloud storage to the Machine Learning facilities service for labeling,
  2. then, transferred (write-like) from the Machine Learning facilities service back to Cloud storage,
  3. again, transferred (read-like) from Cloud storage to the Machine Learning facilities for training,
  4. finally, stored (write-like) for later usage (retraining).

4.3 Data transfer model

Data transfers may incur costs related to network usage metering, excluding application messages and telemetry. We make the following assumptions:

- Any edge device is provisioned once at bootstrap with a payload of `edge_img_size` (OS) and `ml_model_size` (MLM).
- Application output data (i.e., inference results) are always uploaded to the Cloud. Raw data are uploaded to the cloud only upon failed inference.
- We assume that the average number of learning algorithm executions is set by `ml_train_rate`. Accordingly, any affected edge device is re-provisioned with a payload of `ml_model_size+edge_img_size`.

4.4 Computing model

When the prediction fails at the Edge, some raw (low-performance) data are uploaded to the Cloud, with the following assumptions:

- Data labeling time is not taken into account.

– If a platform does not provide an optimized ML training service, we use the pricing figures for computing resources based on equivalent or similar technology.

Our computing model:

1. Sets a target maximum training time for the whole deployment (`ml_train_deadline`), irrespective of its size and of the number of contexts.
2. Computes the sequential virtual CPU (vCPU) time (`monthly_cloud_computing_rate`) needed to train one MLM.
3. Computes the minimum number of vCPUs (`cloud_computing_vcpus`) needed to meet the training deadline for one MLM, assuming that all vCPUs run in parallel and are fully utilized, and that the training process is embarrassingly parallel.

4.5 PaaS pricing

Edge-to-cloud platforms may charge for their services on either a pay-as-you-go or a long-term commitment basis, possibly with discounts. We consider here only platform-as-a-service (PaaS) hosting costs for long-term (1 year), large-scale deployment (1K active devices). Costs are incurred at six different service layers:

– **Edge** device management. It may incur the following costs:
  • A yearly **Subscription**.
  • A **Registered** edge device with any associated fixed costs, such as specific telemetry metrics.
  • A minute of **Connectivity**, assuming that any device is connected 24/7.
– **Messaging**. We only consider edge-cloud communication.
  • A **Telemetry** message exchanged with the Cloud.
  • An exchanged **Application** message (irrespective of its type) and any subsequent operation triggered in the Cloud: we assume that each message triggers exactly one device status update plus another Cloud operation (statistic aggregation, dashboard visualization, etc.). For platforms metering by volume, we do the appropriate conversion.
– **Data transfer**: each byte transferred from the Edge to the Cloud and vice versa, as well as between different Cloud services may incur a cost. Thus, we break this down into: **Cloud-to-Edge**, **Edge-to-Cloud** and **Intra-Cloud**. For this latter, we assume that all transfers occur within the same "availability" zone, that is, roughly speaking inside the same data center.
– **Storage**: each unit of **Space** stored incurs a cost; each **Read** and **Write** operation may incur a cost.
– **Computing**: each hour spent by any application's Cloud component incurs a cost. Additional costs for a minimum amount of attached storage are also considered.
– Technical support: **Helpdesk**.

Details about the pricing that accommodates the two scenarios for the different platforms are available elsewhere [10].

5 Operating cost comparison

The cost estimates are made with the assumption that each "solution" covers all the edge application's needs in terms of messaging, data transfer, storage and computation. The only fully-integrated solutions considered in this work are AWS and Azure, whereas the others need partnering with foreign service providers: we consider official partnerships when they exist, and otherwise the cheapest foreign Cloud option fulfilling the mission; foreign partnership is supposed to incur the extra provider's helpdesk costs. Specifically:

- Google Cloud does not provide native Edge infrastructure management services.
- Google has partnered with Balena, hence the complete solution "Google Cloud + Balena".
- SixSq Nuvla does not provide native telemetry, event, storage and computing services. SixSq has partnered with Exoscale for storage and computing services. The cheapest telemetry and event services are provided by AWS. Hence the complete solution "SixSq Nuvla + Exoscale + AWS".
- Balena can be used as a stand-alone Edge management platform, though it needs extra services exactly as for Nuvla. Hence the complete solution "Balena + Exoscale + AWS".
- Since AWS support fees are proportional to the total AWS charges, and AWS telemetry/event service fees account for ~28% of the whole AWS solution, the same corresponding extra hepldesk cost has to be charged to both SixSq Nuvla and Balena.

Estimates are provided for two scenarios: Road Traffic Management and Smart Grid. The values of the cost model's parameters (c.f. Table 1) of these two scenarios are detailed in [10].

Scenario: Road Traffic Management In this scenario, the edge devices classify transiting vehicles by sampling environment noise via high-definition microphones. By our estimates and preliminary knowledge, the application is characterized by:

- High event rate (~$1Kevent/hour$), because of heavy traffic in dense urban areas.
- Big raw data footprint (~$2MiB/sample$), because noise sensors must sample some seconds of high-resolution stereophonic audio signal.
- High ML error rate (~35%), because of the involved bleeding-edge technology based on neural networks.
- Long ML training times (~$24h$ @ $1vCPU$), because of the involved complex models.

The cost breakdown for a deployment of $1K$ edge devices over 1 year is shown in Figure 3. The main variable cost drivers are especially storage operations and then messaging. Computing and data transfer have a very small impact ($< 1\%$) which is somewhat unexpected and needs further investigations: a possible reason could be our overly optimistic computing model.

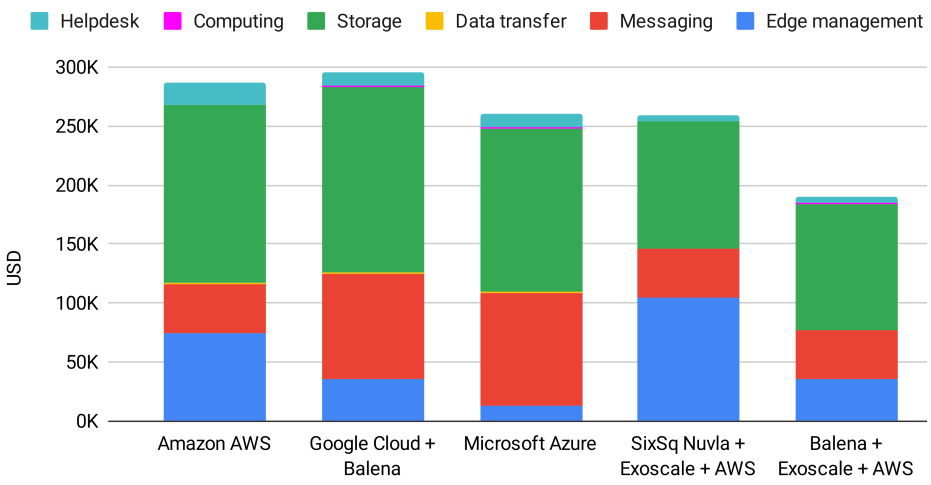

Fig. 3. Road Traffic Management: Cost breakdown for 1K-deployment over 1 year.

Scenario: Smart Grid In this scenario, the edge devices forecast energy production and consumption by sampling electric power at several sensing stations in the same context—household, office, school, plant, etc. By our estimates and preliminary knowledge, the application is characterized by:

- Low event rate (~60 $event/hour$), because power measurements are normally averaged over a rather long time frames (minutes).
- Small raw data footprint (~0.1$MiB/sample$), because only power measurements are involved.
- Low ML error rate (~5%), because of proven and stable forecasting methods.
- Short ML training times (~4h @ 1$vCPU$), because forecasting is mainly based on simple models, such as XGBoost and LSTM.
- High ML training rate (~8 $round/month$), because the application needs to react quickly to context variations–changing weather, reduced consumption during unplanned absences, etc.

The cost breakdown for a deployment of 1K edge devices over 1 year is shown in Figure 4. The main variable cost drivers are messaging and storage operations for two of the well-known providers; conversely, the edge management cost is dominating with AWS and the two newcomers. Again, computing and data transfer have a very small impact ($< 1\%$). Storage costs are negligible with the newcomer solutions because the adopted storage provider (Exoscale) does not charge for operations.

6 Conclusion

Large-scale IoT applications based on ML and employing self-adaptive algorithms call for cost-effective edge-to-cloud solutions. Indeed, this kind of systems

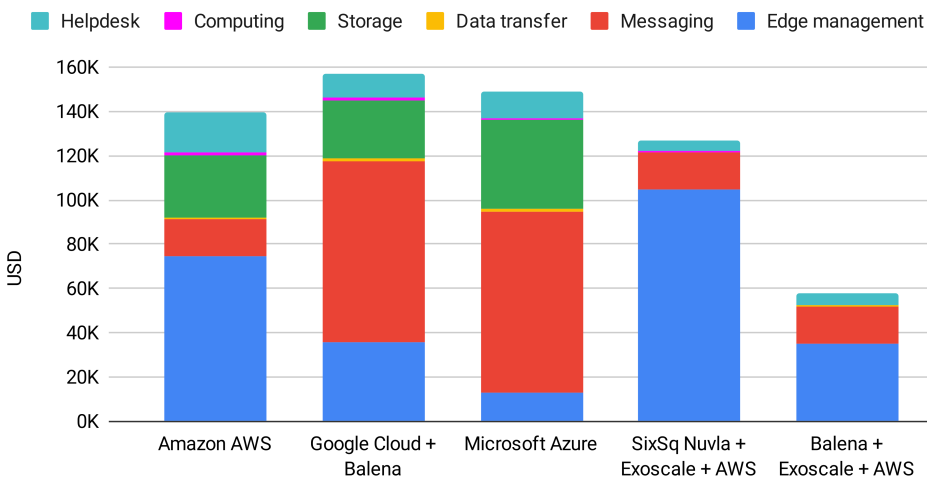

Fig. 4. Smart Grid: Cost breakdown for 1K-deployment over 1 year.

poses challenges both at the Edge, because ML inference has to be performed efficiently on resource-constrained devices, and in the Cloud, because vast amounts of data have to be transferred and stored. Also, managing thousands of IoT and edge devices needs streamlined solutions for system monitoring and recurrent software updates.

Stakeholders wish to make informed decisions about the best PaaS Cloud platform for such deployments; thus, an analysis of the application requirements has been done to isolate the needed service components. Based on that, we proposed, as a first contribution, a generic reference service-oriented architecture as well as an event-driven application workflow, which we then mapped on selected Cloud platforms. Our second contribution is a detailed resource consumption model, based on PaaS pricing, which considers edge management, messaging, data transfer, storage space and operations, computing and helpdesk. We drew some cost estimates in two scenarios (Road Traffic Management, Smart Grid). The results show that the expenditure breakdown may vary significantly across the considered platform; among the variable costs, computing and data transfer have a very low impact compared to messaging and storage operations (space fees are negligible); the fixed costs (edge management and helpdesk) may dominate in scenarios characterized by a compact data footprint. The surprisingly low computing cost, especially in computationally-heavy scenarios like Road Traffic Management, might induce thinking that, contrarily to our expectations, dynamic AI (context-awareness and self-adaptation) has no significant incidence on overall expenditures. This demands more research: in fact, our simple computing model might need refinements to consider resource contentions such as cache/main memory scarcity and its consequent CPU stalling effects.

For future developments, we plan to integrate the edge-to-cloud cost model discussed in this paper into our placement tool [3]. We are also interested in

exploring cost models based on Functions-as-a-Service (FaaS) instead of pure Docker containers.

References

1. Amazon Web Services, Inc.: AWS IoT Greengrass, https://aws.amazon.com/greengrass/. Last accessed: 2022-06-01.
2. Balena: Balena, https://www.balena.io/. Last accessed: 2022-06-01.
3. Belli, O., Loomis, C., and Abdennadher, N.: Towards a Cost-Optimized Cloud Application Placement Tool. In: 2016 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), pp. 43–50 (2016). DOI: 10.1109/CloudCom.2016.0022
4. Google: Goolge Clouf IoT Core, https://cloud.google.com/iot-core. Last accessed: 2022-06-01.
5. Goudarzi, M., Wu, H., Palaniswami, M., and Buyya, R.: An Application Placement Technique for Concurrent IoT Applications in Edge and Fog Computing Environments. IEEE Transactions on Mobile Computing 20(4), 1298–1311 (2021). DOI: 10.1109/TMC.2020.2967041
6. Laatikainen, G., Ojala, A., and Mazhelis, O.: Cloud Services Pricing Models. In: Herzwurm, G., and Margaria, T. (eds.) Software Business. From Physical Products to Software Services and Solutions, pp. 117–129. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
7. Martens, B., Walterbusch, M., and Teuteberg, F.: Costing of Cloud Computing Services: A Total Cost of Ownership Approach. In: 2012 45th Hawaii International Conference on System Sciences, pp. 1563–1572 (2012). DOI: 10.1109/HICSS.2012.186
8. Microsoft: Azure IoT Edge, https://azure.microsoft.com/en-us/services/iot-edge/. Last accessed: 2022-06-01.
9. Nguyen, D.T., Pham, C., Nguyen, K.K., and Cheriet, M.: Placement and Chaining for Run-Time IoT Service Deployment in Edge-Cloud. IEEE Transactions on Network and Service Management 17(1), 459–472 (2020). DOI: 10.1109/TNSM.2019.2948137
10. Poleggi, M.E., Abdennadher, N., Dupuis, R., and Mendonça, F.: Edge-to-cloud solutions for self-adaptive machine learning-based applications. Tech. rep., HEPIA - Haute école du paysage, d'ingénierie et d'architecture (2022)
11. SiqSq SA: Nuvla, https://sixsq.com/products-and-services/nuvla/overview. Last accessed: 2022-06-01.