Migrating IoT Processing to Fog Gateways

Daniel Happ, Sanjeet Raj Pandey, Vlado Handziski

Technische Universität Berlin, Telecommunication Networks Group (TKN)
{happ, pandey, handziski}@tkn.tu-berlin.de

***Abstract*—Internet-connected sensor devices usually send their data to cloud-based servers for storage, data distribution and processing, although the data is often mainly consumed locally to the source. This creates unnecessary network traffic, increases latency and raises privacy concerns. Fog and edge computing instead propose to migrate some of those functions to the edge of the network. In particular, on premise gateways have the potential to offer more privacy preserving and low-latency local storage and processing capabilities. In this study, we outline our ongoing efforts to combine the benefits of fog and cloud sensor data processing. We present our work-in-progress towards a system that automatically selects the most suitable execution location for processing tasks between cloud and fog. We present a protocol for migration of processing tasks from one system to another without service interruption, and propose a reference architecture. We additionally introduce an analytical cost model that serves as basis for the placement selection and give advice on its parametrization. Finally, we show initial performance results, gathered with an early prototype of the proposed architecture.**

I. Introduction

The Internet of Things (IoT) enables applications that interact with the physical world around us in real-time by embedding sensors, software and network connectivity into physical objects. Additional software components for sensor data distribution, storage, and analytics, are often offered as services by cloud providers [1]. However, cloud computing is not always the most suitable option for offering those services in the IoT context, especially for latency and privacy sensitive applications [2]. Instead of forcing all data through cloud servers that are possibly located far away, fog computing proposes to move some of those services to the edge of the network [3], in our case to existing gateways devices.

We envision the architecture of future IoT systems to have 3 layers, as shown in Figure 1. The device layer consists of sensor and actuator devices, which are severely constrained in terms of processing power, memory and energy. Those "things" are connected to a nearby gateway which relays their data to the Internet. Fog and cloud layers provide the means to distribute, store and process sensor data. While fog instances are relatively local and thus closer to the end-user, the centralized remote cloud provides ubiquitous and seemingly infinite access to storage and processing.

We expect that all the components use the publish/subscribe pattern to communicate [2], [4]. In this one-to-many pattern, the matching between consumer and producer of data is done by message brokers, which we expect to run on cloud and fog instances. We mainly consider the topic based naming scheme, where the symbolic channel addresses are strings.

We identify gateway devices as a prime candidate for local, low-latency and privacy preserving storage and processing. While sensor devices themselves are very constrained, today's IoT gateway hardware include powerful embedded single-board computers (Raspberry Pis, BeagleBone Blacks, Intel NUCs, home routers, etc.) and smartphones and are powerful enough to take over some of the services the cloud is providing today. Gateways in industrial and home environments are usually mains-powered and not energy constrained. We argue that already deployed gateway hardware has a significant amount of potential processing power and storage available right on premise that is not yet fully utilized. Through leveraging local processing on gateway devices and migrating heavy computation to resourceful cloud servers on demand, the resource constraints of local gateway hardware regarding CPU-, memory-, or network-intensive tasks can be overcome while still benefiting from the locality gateways provide.

In this work, we present our ongoing effort aimed at developing a flexible framework for IoT process relocation. Although the concepts provided here universally apply to process migration in distributed IoT systems using publish/subscribe, we consider the context of gateway to cloud offloading as an example. Our main contributions in this work are threefold: 1) We outline a migration mechanism and a framework for sensor processing tasks. 2) We provide a cost model for processing task migration and give advice on a suitable parametrization. 3) We show preliminary measurements of a research prototype. Those parts of this paper describing the framework and measurements are largely based on already published work [5]. The in-depth analysis of the cost model for task migration has not been published before.

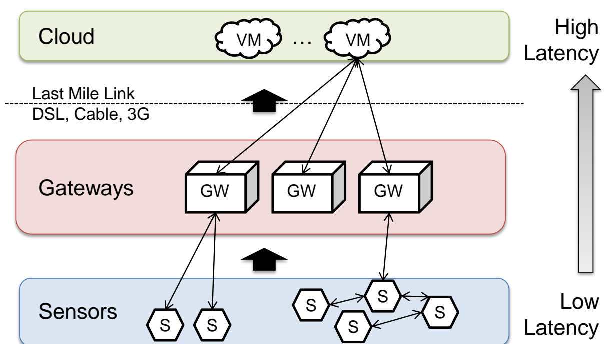

Fig. 1. Overall architecture of a simplified Fog-enabled IoT platform.

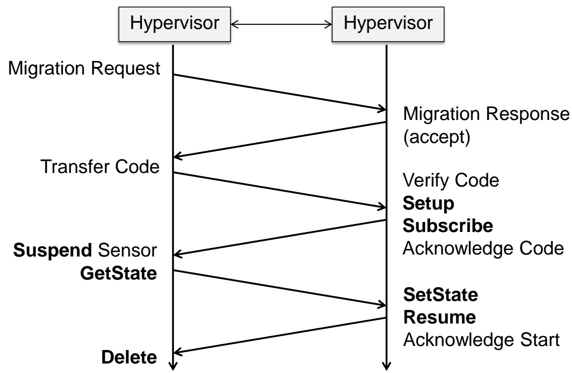

Fig. 2. Migration between two hypervisors.

II. Gateway to Cloud Offloading of Sensor Processing Tasks

In this work, we define a sensor processing task very broadly as an arbitrary computation on a set of input sensor streams that create one or more output streams. Input streams are exclusively obtained by subscribing to a pub/sub system. Likewise, the output is published on one or more output topics. The process can be defined freely by the user; we do not make any assumptions about the function provided.

A process has two parts, the code that defines its function and its state. To enable migration, we need a standard way to snapshot the state and to restore it at the new location. We therefore require each processing task to implement a predefined set of operations, which are given in Table I. The getState method is used to extract the current process state. The setState method is called to set the (initial) state of the task. The state has to be serialize to a byte stream. The specific encoding of the state is up to the task developer, as long as it is serialized to a byte stream.

For starting, suspending and resuming tasks, we use additional operations. Since we do not want to make any assumption on the specific type of task, each one has its own constructor/deconstructor (setup/delete) methods to initialize the process, e.g. with a pub/sub broker. The subscribe method starts to subscribe on every input topic given and fills the relevant queues with sensor data. It does not start processing any data. This is triggered by a call to the resume method, which takes the sequence number of the next value to process. A process is likewise suspended using the suspend call, which returns the next sequence number to process.

The system we envision should dynamically adapt to the current state of each available processing instance and offer migration from cloud to gateway and from gateway to cloud. While not specifically targeted in this work, a gateway to gateway offloading would be possible as well. As a simplified example, we use one gateway and one cloud instance. We envision that the system has a default entry point that stays responsible for the task. In our case, this is the local gateway. Each task is defined by code, initial state and estimates for resource usage. Since different task have different requirements, not all tasks are equally suitable for remote execution. The task issuer may thus give additional constraints, e.g. if a task must not be offloaded. The placement of the processing task must comply with the given constraints, as well as the available resources on the gateway. Resource usage is continuously monitored and the corresponding task meta-data updated.

TABLE I
Overview of specific processing task methods.

Method	Desciption
setup	called before launch to initialize task
subscribe	called to subscribe to input topics and fill buffers
setState	called to set initial state to representation given
resume	called to start processing
restart	called to restart processing
suspend	called to suspend processing
getState	called to get representation of the process state
delete	called for terminating the processing task

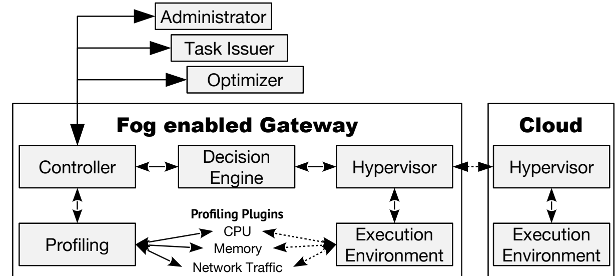

Fig. 3. Simplified gateway to Cloud offloading architecture.

Figure 3 depicts the high-level architecture of the offloading system. Every entity offering execution of processing tasks has a Hypervisor that suspends, resumes and migrates processing tasks. The execution environment is instrumented with profilers that monitor the overall and per task resource usage. The entities that not only support processing of existing tasks but also issuing new processing tasks have a controller that receives the necessary task definitions and relevant metadata. A decision engine decides whether and where to offload the task based on constraints, issued tasks, optimization goal and profiling data.

We further identify three distinct roles to interact with the system: The local administrator defines constraints for the execution environment, e.g. total CPU or memory consumption can be restricted. The task issuer defines the processing function and additional metadata, such as estimates of resource consumption and input and output topics as well as estimated output frequencies. The optimizer sets the optimization goal.

III. Adaptive Offloading Policy

A core part of the presented offloading system is the decision engine, which determines which processes should be offloaded. The problem of assigning processing tasks to processing nodes is related to the knapsack problem and NP-hard. In order to get further insights into the trade-offs of different offloading policies, we formulate the problem of assigning tasks to processing nodes as an optimization problem. We then derive

a greedy strategy and give some pointers for future work in the area.

We assume the simplified case of one fog gateway that can either run processing tasks locally or give them upstream to the next higher layer, which in our case is a cloud backend. We further assume that the cloud can provide infinite resources and the gateway has zero cost for unused resources. We model the system as time discrete with time steps on the creation or destruction of processing tasks and at regular intervals where resource usage is updated and the assignment problem reevaluated. Let us suppose that we have n processing tasks $P_1, P_2, \ldots, P_n$. We further assume a processing task P_i during the time frame t yields an output from the profilers which is characterized as the tuple $\langle cpu_i(t), mem_i(t), rx_{i,loc}(t), rx_{i,rem}(t), tx_{i,loc}(t), tx_{i,rem}(t)\rangle$, where $cpu_i(t)$ is the amount of computing resource required during the time frame (e.g. CPU usage), $mem_i(t)$ is the amount of memory used for the task, $rx_{i,loc}(t)$ and $tx_{i,loc}(t)$ are the number of bytes of traffic received and transmitted locally on the same gateway and $rx_{i,rem}(t)$ and $tx_{i,rem}(t)$ the amount of external traffic to the cloud. When the process is first started, the variables are set to the metadata provided by the task issuer.

We further introduce the decision variable $x_i \in \{0, 1\}$ determining if P_i should be run locally ($x = 0$) or in the remote cloud ($x = 1$). This yields the following function for the monetary cost of the system, which would be one possible optimization goal. The cost function has three parts: The first part describes the bandwidth cost from gateway to cloud. The second and third part are costs for computing power and memory in the cloud respectively. Weight parameters have to be defined according to the cost for CPU, memory and bandwidth of each individual cloud provider. Since we only look at a fixed t, we omit this parameter for clarity:

$$\min_{x_i \in 0,1} (c_{transfer} \cdot \omega_{tr} + c_{memory} \cdot \omega_{mem} + c_{cpu} \cdot \omega_{cpu}) \quad (1)$$

where

$$c_{transfer} = \sum_{i=1}^{n} \quad x_i \quad (rx_{i,loc} + tx_{i,loc}) \\ +(1 - x_i)(rx_{i,rem} + tx_{i,rem}) \quad (2)$$

$$c_{memory} = \sum_{i=1}^{n} mem_i \cdot x_i \quad (3)$$

$$c_{cpu} = \sum_{i=1}^{n} cpu_i \cdot x_i \quad (4)$$

Additionally, we identify several constraints, namely that processes run locally may in total not exceed a predefined fraction of the available resources:

$$\sum_{i=1}^{n} cpu_i \cdot (1 - x_i) \leq avail_{cpu} \cdot f_{cpu} \quad (5)$$

$$\sum_{i=1}^{n} mem_i \cdot (1 - x_i) \leq avail_{mem} \cdot f_{mem} \quad (6)$$

TABLE II
GOOGLE CUSTOM MACHINE PRICES.

Item	Price ($)	ω
vCPU	0.033174 / vCPU hour	0.210463
Memory	0.004446 / GB hour	0.028207
Bandwidth	0.12 / GB	0.761329

$$\sum_{i=1}^{n} rx_{i,rem} \cdot (1 - x_i) + tx_{i,loc} \cdot x_i \leq avail_{bw\downarrow} * f_{rx} \quad (7)$$

$$\sum_{i=1}^{n} tx_{i,rem} \cdot (1 - x_i) + rx_{i,loc} \cdot x_i \leq avail_{bw\uparrow} * f_{tx} \quad (8)$$

To evaluate different offloading policies, we additionally need to find a suitable parametrization of the aforementioned model, i.e. the weights for the cost parameters. Since these parameters are subjective to the system configuration, we give advice on parametrization with a specific but realistic example. We use prices provided by Google for a "custom" virtual machine types as listed in Table II. We can use those prices to determine the weights ω_{tr}, ω_{mem} and ω_{cpu} by dividing the individual price for one traffic, memory and CPU item by the sum of all prices.

As a first example, this cost function can trivially be transformed into a greedy strategy. First, all processes that were determined to never be offloaded due to provided metadata are copied to the "local" set and resources are allocated accordingly. The algorithm sorts the remaining processes by the specific gain w.r.t. the cost function. It iterates over this sorted list of remaining processes and adds the one with the highest gain to the "local" set and allocates local resources if enough resources are available. That means it effectively determines the process that is saving the most amount of monetary cost when running locally in contrast to running it in the cloud. If that does not yield a possible allocation, the algorithm should stop or reject tasks, e.g. the tasks that where created last.

We plan to use Markov decision processes (MDPs) to determine the solution to our offloading problem in future work. MPDs are discrete time stochastic control processes, which can be used to model decision processes with partly random outcome. Since processing task have event-based input data that shows partly random behavior and random external impact on the system, such as background processing tasks, MPDs seem to be a good fit for the problem at hand. If the state transition function P and the reward function R of an MPD are known, a policy can be computed. Because of the random nature of the amount of input data over a specific timeframe, P and R cannot be expected to be static and known in advance. Our idea is to use reinforcement learning to continuously reestimate the current values for both functions. With this approach, the system would in any state with a relatively low probability "try out" assignments that are expected to give a suboptimal reward to rediscover better solutions in case of an outdated estimate of P and R and update the functions accordingly. As mentioned above, this is part of future studies.

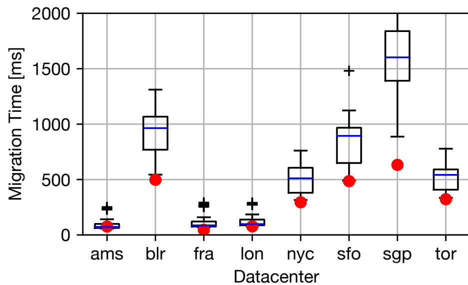

Fig. 4. Migration between two hypervisors.

IV. Evaluating Migration Time

In the previous model we have ignored the cost and time of task migrations. We use a prototype of our system to evaluate this time. The prototype is written in python and uses Message Queue Telemetry Transport (MQTT) as the messaging middleware between all components of the framework and also as the pub/sub system the actual processing tasks uses for input and output data. As a first step, we use the prototype to evaluate the migration time between a gateway node in Berlin, Germany and a cloud-based virtual machine, manifests as a perceived service interruption or lag.

We use a Beaglebone Black as the local gateway and cloud instances in every datacenter of the provider Digital Ocean. We use a processing task that we assume to be realistic to offload, namely a motion detection task that analyses webcam footage. The size of this task is around 3kB. The state of the task is a background image which is used to detect changes and varies in size depending on the image. In our example it is 21kB.

Figure 4 shows the migration times of 64 migrations back and forth along with a red dot marking an estimate calculated from round trip time and throughput measurements, which can be easily done using widely available tools. The migration time is in the worst case under 2.5s even to very remote destinations. Due to the additional overhead and latency introduced by the messaging middleware, the actual average migration time shows to be considerably higher than the prediction, with the estimation being the lower bound of the migration time.

In conclusion, since we do not expect migrations across the globe to happen often, the migration time is small enough to be tolerable by the end-user for most sensor applications; especially for migrations to cloud backends on the same continent, which are well below 200ms. The measurements additionally highlight the need for a prototypical implementation of the system. First, the migration time that is ignored in our model has to be taken into account by future work on the topic. Second, it shows that parameters of an extended model can potentially not be accurately derived from simple estimations but have to be verified by measurements in realistic settings.

V. Related Work

Many mechanisms have been proposed in previous work that address the challenges of seemless offloading, particularly mobile offloading from phones to infrastructure. In the IoT context, existing work is still limited. The authors of [6] describe an integrated fog cloud IoT (IFCIoT) architectural paradigm, including application, analytics, virtualization, reconfiguration, and hardware layer. In [7], the authors propose a virtual machine migration mechanism for fog computing. Aazam and Huh [8] present a resource estimation and pricing model for fog-based IoT, including resource prediction, estimation, reservation, and pricing. We complement this existing work by focusing on the specific context of processing tasks using publish/subscribe. Additionally, previous work often lacks the notion of state, which we deem necessary.

VI. Future Work and Conclusion

In this work, we argue that the full potential of globally interconnected sensor technology will only be fully utilized through increased local sensor data processing. To realize this vision, we present an offloading mechanism and an initial architecture of a generic offloading framework which enables flexible definition of processing tasks as well as constraints and optimization targets. We provide a formal definition of the offloading problem, give advice on the parametrization and give hints on potential solutions. We highlight the need for a prototypical implementation of such a framework and show that processing tasks found in IoT systems can be migrated in a reasonable short time. We further plan to expand our platform to not only take into account the available resources, but to optimize the offloading decision with regard to a wider range of performance targets, such as reduced latency, minimization of energy consumption, or reduced network traffic. We also plan to expand the model and the evaluation in terms of migration time and cost and more realistic use cases.

References

[1] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645–1660, Sep. 2013.

[2] B. Zhang, N. Mor, J. Kolb, D. S. Chan, K. Lutz, E. Allman, J. Wawrzynek, E. Lee, and J. Kubiatowicz, "The cloud is not enough: Saving iot from the cloud," in *7th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud '15)*. Santa Clara, CA: USENIX Association, Jul. 2015.

[3] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, ser. MCC '12. New York, NY, USA: ACM, 2012, pp. 13–16.

[4] D. Happ, N. Karowski, T. Menzel, V. Handziski, and A. Wolisz, "Meeting IoT Platform Requirements with Open Pub/Sub Solutions," *Annals of Telecommunications*, vol. 72, no. 1, pp. 41–52, 2017.

[5] D. Happ and A. Wolisz, "Towards gateway to cloud offloading in iot publish/subscribe systems," in *Second International Conference on Fog and Mobile Edge Computing, FMEC 2017, Valencia, Spain, May 8-11, 2017*, 2017, pp. 101–106.

[6] A. Munir, P. Kansakar, and S. U. Khan, "Ifciot: Integrated fog cloud iot architectural paradigm for future internet of things," *CoRR*, 2017. [Online]. Available: http://arxiv.org/abs/1701.08474

[7] L. F. Bittencourt, M. M. Lopes, I. Petri, and O. F. Rana, "Towards virtual machine migration in fog computing," in *10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC)*, Nov 2015, pp. 1–8.

[8] M. Aazam and E. N. Huh, "Fog computing micro datacenter based dynamic resource estimation and pricing model for iot," in *29th International Conference on Advanced Information Networking and Applications*, March 2015, pp. 687–694.