2014 IEEE 6th International Conference on Cloud Computing Technology and Science

Energy-Efficient QoS-aware Service Allocation for the Cloud of Things

G. Tanganelli, C. Vallati, E. Mingozzi
Dip. Ingegneria dell'Informazione, University of Pisa
L.go L. Lazzarino 1, I-56122
Pisa, Italy
{g.tanganelli, c.vallati, e.mingozzi}@iet.unipi.it

Abstract— **Cloud computing is a key enabler for the development and deployment of large-scale IoT service platforms. The integration into such platforms of different sensing and actuating systems will imply the availability of many similar IoT services with common functionalities though with different QoS and cost. A cloud-based IoT platform can then benefit from implementing QoS-aware service selection algorithms to match application demand to IoT services, whilst guaranteeing to meet the respective QoS requirements. Such algorithms must however take into account that IoT service providers are usually constrained devices with limited computation, storage and energy capabilities. In this work we formulate the QoS-aware service selection problem for IoT cloud platforms as an integer optimization problem, whose solution minimizes the energy consumption so as to maximize the lifetime of battery-powered devices, whilst guaranteeing the fulfillment of real-time QoS requirements. We then propose a computationally efficient heuristic algorithm to solve the problem, and show through extensive numerical analysis that it is able to find solutions very close to the optimal one in all considered scenarios.**

Keywords–Cloud-bases IoT platforms, M2M applications, QoS-aware service selection, Energy efficiency

I. INTRODUCTION

The Internet of Things (IoT) vision foresees a future in which *smart* things or objects are connected to the Internet and exploited as remote sensing and/or actuating components of large distributed computing infrastructures [1]. In this scenario, machine-to-machine (M2M) applications, i.e., IoT applications exploiting direct interactions between things in order to monitor and control themselves and the surrounding environment with no or minimum human intervention, are expected to play a major role, given their huge impact on many real-world application domains including industrial, healthcare, transportation, social and environmental [2].

More recently, the cloud computing paradigm has been considered as a key enabler for the development and deployment of large scale IoT platforms, since it allows for the global sharing of (IoT) resources in a naturally distributed environment, elastic resource provisioning, and flexible interaction with cloud customers [3]. Traditional cloud-based service models are being extended to accommodate for this new class of *IoT service providers* featuring sensing and actuating capabilities, thus introducing the so called Things as a Service model [4].

New cloud-based architectural designs are also being considered to account for the distinctive features and characteristics of IoT components, i.e., support of real-time QoS-aware interactions and collection of fresh and accurate context information. To this aim, such novel hierarchical architectures extend the cloud to the edge through a capillary infrastructure that moves computing and storage services closer to sensors and actuators. A notable example of these new approaches is the so called *fog computing* paradigm [5][6].

Cloud-based hierarchical IoT service platforms will have to face many challenges. On one hand, the integration of different sensing and actuating systems into one single service infrastructure allows applications to benefit from high IoT service availability: different *equivalent* smart things may provide similar services with common functionalities but different QoS and cost (e.g., different smart cameras may provide, if appropriately steered, the same view of a given area from different directions). To this aim, an efficient management of resources will be required to perform a proper selection of things matching application requests, whilst guaranteeing to meet the respective QoS requirements. On the other hand, however, smart things are constrained devices in terms of computation, storage and energy (since they may be battery operated). Moreover, things are more volatile and dynamic: continuous changing context and intermittent availability are the two main factors that differentiate IoT services supported by smart objects, i.e., battery powered devices perhaps in movement, from traditional services running on fixed powerful hosts. Therefore, novel service selection approaches are needed to address these unique features when allocating things to application requests.

In this work we propose a QoS-aware service selection algorithm designed for IoT cloud platforms. The contribution of the paper is twofold: (i) an energy-efficient thing allocation problem is formally defined as an instance in the class of generalized assignment problems; and (ii) a time-efficient heuristic algorithm is proposed that is shown, through numerical analysis, to find a solution close to the optimal one in a time suitable for implementation in a real system.

The rest of the paper is organized as follows. In Section II we describe the conceptual architecture underlying the problem formulation, which is formally defined in Section III. In Section IV we illustrate our proposed solution to the service selection problem, and in Section V we show the numerical results of the performance evaluation. Finally, in Section VI we provide an overview of the related work, while Section VII draws the conclusions and outlines possible future work.

978-1-4799-4093-6/14 $31.00 © 2014 IEEE
DOI 10.1109/CloudCom.2014.148

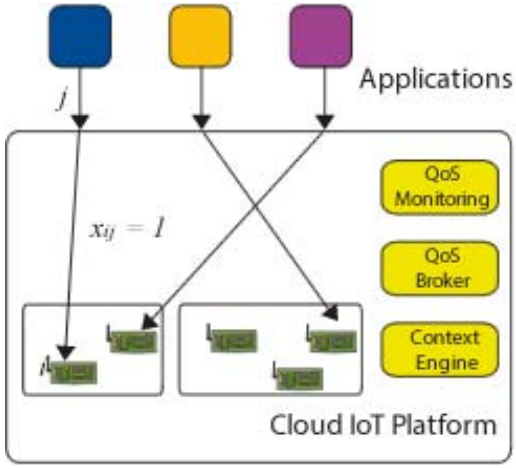

Figure 1. Conceptual model.

II. ASSUMPTIONS

The conceptual model considered in this work is illustrated in Figure 1. A cloud-based service platform integrates different independent IoT systems, each one composed of a set of smart things. Each thing is as a *service provider* that can expose one (or more) service(s) identified by some *context* information. Context, defined as "any information that characterize the situation of an entity" [8], enhances the description of a service, e.g., *acquisition of the temperature*, with any additional relevant information related to it, e.g., its *location*, its *freshness*, etc.

Applications connect to the cloud to request a specific service with a desired level of QoS. It is worth to mention that the cloud paradigm adopted in this work is called *fog* and it is an extension to the classic centralized cloud. The basic idea is to have a geographically distributed (fog) tier, deployed close to the edge, for the execution of real-time applications. In this work, in fact, we focus on real-time periodic service invocations with time-bounded requirements; support for services with sporadic invocations is left for future work. The requested service is also identified through some context information that will be used to lookup available services in the platform.

A *QoS broker* capable of engaging in QoS negotiation with applications is deployed; its goal is to manage and reserve the resources of the platform so that the service level agreed with applications can be met [9]. Application requests are accepted or rejected based on context information and the status of the system. The latter is maintained by a *QoS monitoring* entity, which provides information about the battery level, as well as the computational, communications, and storage capabilities of things. Context information is processed instead by the *Context engine,* the entity within the platform responsible for collecting and managing all the context information related to things. The *Context engine* implements also a context-based service look-up functionality: it determines, for each application service request, the list of *equivalent* things that can provide that service. This is done by analyzing both the current context information associated to things, and the context information associated to the required service. Algorithms for context management and analysis are however outside the scope of this work.

Based on the information about the status of things provided by the *QoS monitoring* entity, and the set of equivalent things per service request determined by the *Context engine*, the *QoS broker* determines if all requests can be satisfied. It then allocates one thing to each service request so as to minimize the energy consumption in case of energy-constrained devices, thus maximizing the lifetime of the system. In this work, we specifically focus on this allocation problem for a fixed set of requests and available things, and develop a heuristic algorithm to find a solution close to the optimal one.

As other requests are negotiated or the system status changes, thing selection and allocation to application requests need to be re-computed (transparently to applications) in order to continue guaranteeing the commitment to meet QoS requirements.

III. PROBLEM FORMULATION

We now formally define the thing allocation problem addressed in this work. We are given a system comprising n things, each exposing a subset of IoT services, and a set of k requests for service invocation. Each service j is assumed to be invoked periodically with period p_j, e.g., in order to retrieve periodic updates from sensors or to send periodic commands to actuators.

Thing i is assumed to be a constrained device capable of satisfying only one service invocation at a time. Moreover, a thing may be battery-powered. Let b_i be the battery capacity on thing i, i.e., the amount of available energy before service allocation (possibly equal to $+\infty$, if not battery-powered). Each invocation of a service j on thing i has a fixed *execution time* t_{ij}, including both communication and computation times, and a fixed *energy cost* c_{ij}, representing the overall amount of energy needed to accomplish the invocation of service j on thing i (including also energy consumption due to communication). The cost of execution of a service on a thing has a different impact depending on the initial battery level of the thing. In order to make a fair comparison among costs, we consider costs of execution over a common (hyper-)period h and normalized with respect to the available energy b_i. The hyper-period h is computed as the least common multiple among all request periods p_j; the *normalized energy cost* f_{ij} of executing service j on thing i is given by

$$f_{ij} = \frac{h}{p_j}\frac{c_{ij}}{b_i} \tag{1}$$

Not all services can be invoked on any thing, but the same service can be invoked on multiple equivalent things. Equivalence among things is based on context information associated to thing services, which we assume is provided by m_{ij} as follows

$$m_{ij} = \begin{cases} 1 & \text{if service } j \text{ can be invoked on thing } i \\ 0 & \text{otherwise} \end{cases}$$

Without losing generality, we assume that each service request can be executed by at least one thing, i.e., for any j, there is at least one i such that $m_{ij} = 1$.

The *utilization* u_{ij} of allocating requests for service j on thing i is finally defined as

$$u_{ij} = \begin{cases} t_{ij} / p_j & \text{if } m_{ij} = 1 \\ +\infty & \text{otherwise} \end{cases} \quad (2)$$

The thing allocation problem is then to allocate the k requests to the n things so as to minimize the maximum (normalized) energy cost among things over a hyper-period h, whilst guaranteeing that all service invocations are completely executed before their *implicit deadline*, i.e., the arrival of another invocation of the same service. Formally:

$$\min \left(\max_{1 \le i \le n} \sum_{j=1}^{k} f_{ij} x_{ij} \right) \quad (3)$$

s.t.

$$\sum_{i=1}^{n} x_{ij} = 1, \quad j \in K \quad (4)$$

$$\sum_{j=1}^{k} f_{ij} x_{ij} \le 1, \quad i \in N \quad (5)$$

$$\sum_{j=1}^{k} u_{ij} x_{ij} \le v, \quad i \in N \quad (6)$$

$$x_{ij} \in \{0,1\}, \quad i \in N, \quad j \in K \quad (7)$$

where

$$x_{ij} = \begin{cases} 1 & \text{if request } j \text{ is allocated to thing } i \\ 0 & \text{otherwise} \end{cases}$$

and $K = \{1,\ldots, k\}$, $N = \{1,\ldots, n\}$.

Constraint (4) ensures that each service request is assigned to only one thing, whereas constraint (5) bounds the energy consumption of each thing in the hyper-period h to its battery capacity b_i. Finally, constraint (6) bounds the overall utilization of each thing to a *schedulability limit* v so as to guarantee that the implicit deadline of each invocation is satisfied. The limit v is a bound based on the well-known sufficient condition for the schedulability of a set of periodic tasks on a single CPU [10].

The problem is an Integer Linear Problem that results to be an instance of the *Agent Bottleneck Generalized Assignement Problem* (ABGAP) [11], a variation of the well-known *Generalized Assignment Problem* (GAP) defined in the literature and known to be NP-hard. To the best of our knowledge, there is no well-known general algorithm specifically designed to solve ABGAP. Stemming from heuristics proposed to solve GAP and BGAP in [12] and [13], respectively, we developed a novel greedy polynomial-time heuristic algorithm to solve ABGAP, and applied it to the problem defined by (3)-(6)

IV. THE RTTA ALGORITHM

In this section we describe the proposed heuristic, named *Real Time Thing Allocation* algorithm (*RTTA*), to solve the ABGAP problem defined in the previous section. The input to *RTTA* are: the number of things n, the number of requests k, the normalized energy cost matrix $\mathbf{F} = \{f_{ij}\}$, the utilization matrix $\mathbf{U} = \{u_{ij}\}$, a *precision threshold* ε, and, finally, an optional *priority matrix* $\mathbf{P} = \{p_{ij}\}$. The latter is used to steer the thing

```
Algorithm RTTA
Input: n, k, P, F, U, ε
Output: z, y, isFeasable
1:   [z, y, isFeasable] ← Feas(…)
2:   if isFeasable = True then:
3:     last_feas ← 0; upper ← 1; lower ← 0
4:     while upper – lower > ε do:
5:       θ ← (upper – lower)/2
6:       [z, y, isFeasable] ← Feas(…)
7:       if isFeasable = True then:
8:         last_feas ← θ; lower ← θ
9:         θ ← θ + (upper – lower)/2
10:      Else:
11:        upper ← θ;  θ ← θ - (upper – lower)/2
12:    if isFeasable = False then:
13:      θ ← last_feas
14:      [z, y, isFeasable] ← Feas(…)
```

Figure 2. Pseudo-code of RTTA.

```
Algorithm Feas
Input: n, k, P, F, U, θ
Output: z, y, isFeasable
1:   N ← {1...n}; K ← {1...k}; v ← k (2^(1/k) - 1);
2:   isFeasable ← True
3:   for i ← 1 to n do: c_i ← 0
4:   for i ← 1 to n do: z_i ← 1
5:   while isFeasable = True and K ≠ Ø do:
6:     d* ← -∞
7:     for each j ∈ K do:
8:       F_j ← {i ∈ N : c_i + u_ij < v, z_i - f_ij > θ}
9:       if F_j = Ø then:
10:        isFeasable ← False
11:        return
12:      i' ← arg max{ p_ij : i ∈ F_j }
13:      if F_j \ {i'} = Ø then: d ← +∞
14:      else: d ← p_i'j – max_2 { p_ij : i ∈ F_j }
15:      if d > d* then:
16:        d* ← d; i* ← i'; j* ← j'
17:      if isFeasable = True then:
18:        y_j* ← i*; z_i* ← z_i* - f_i*j*
19:        c_i* ← c_i* + u_i*j*; K ← K \ {j*}
20:  # Local optimization
21:  for each j ∈ K do:
22:    i' = y_j
23:    F_j ← {i ∈ N : c_i + u_ij <v, z_i - f_ij > θ, i ≠ i'}
24:    if F_j = Ø then: continue
25:    i* ← arg max{ z_i - f_ij : i ∈ F_j }
26:    maximum ← max{ z_i - f_ij : i ∈ F_j }
27:    if maximum > z_i' - f_i'j then:
28:      y_j ← i*; z_i' ← z_i' + f_i'j; z_i* ← z_i* - minimum
29:      c_i' ← c_i' – u_i'j; c_i* ← c_i* + u_i*j
```

Figure 3. Pseudo-code of Feas.

allocation procedure *Feas* described in detail below. The output is: a boolean *isFeasable,* which takes the *True* value if at least one allocation exists, the allocation vector **y** that maps service requests to things, and the corresponding residual battery vector **z**.

The rationale behind *RTTA* is to iteratively search for the first feasible allocation that guarantees the highest minimum level of residual battery for all things. To this aim, RTTA leverages a procedure *Feas* that, given a threshold θ, finds an allocation so that the residual battery on each thing after service invocation is no lower than θ. On every iteration, the threshold θ is decreased until a feasible solution is found with an acceptable precision level measured by ε. More specifically, a binary search strategy is used to reduce the time needed to execute the overall procedure. At any iteration, *upper* and *lower* give the current upper and lower bounds of threshold θ, respectively. When the difference between *upper* and *lower* is less than the input parameter ε, the algorithm stops.

The core of the RTTA algorithm is the procedure *Feas*which is derived from a classical approach in the literature to tackle assignment problems [13]. The pseudocode of the *Feas* algorithm is reported in Figure 3. The allocation is based on the values of a priority matrix **P** passed as input. In particular, element p_{ij} of **P** is a measure of the *desirability* of allocating request j to thing i. All requests are then considered iteratively for allocation. At each step, the next request to allocate, say j, is the one having the maximum difference between the largest and the second largest p_{ij} (for all things i such that constraint (6) is met). Request j is then allocated to the thing i for which p_{ij} is a maximum. If a service request for which no feasible assignment is found, i.e., any possible allocation to a thing implies its residual battery level goes below θ, the algorithm returns *isFeasable* equal to *False*. Otherwise, a post-processing *local optimization* procedure is performed in order to improve the optimality of the solution. This is achieved by performing local exchanges: for each request, the procedure verifies that the exchange of a service request j from the selected thing i' to any other thing i^* increases the overall residual battery; if that happens, the allocation is modified to select the thing i^* instead of i'. The final solution represented by **y** and **z** is then returned.

Several choices are possible for setting the priority values in **P**. Preliminary numerical tests have shown that good results are obtained when **P** is set equal to **F** or **U**. Both cases are then considered in the final specification of the heuristic solution.

The computational complexity of the *RTTA* algorithm is given by the result below.

Property 1. The complexity of *RTTA* is $O(\beta(nk^3))$.

$$\beta = \log_2 \frac{upper - lower}{\varepsilon} \tag{8}$$

Hence, the overall complexity is $O(\beta(nk^3))$.

V. Performance Evaluation

In this section we report a numerical evaluation of *RTTA* as compared to the optimal allocation obtained by solving the problem defined by (3)-(6), by means of a standard optimization solver, i.e., IBM ILOG CPLEX. *RTTA* was implemented in C++. Experiments were run on a machine with a Linux 64bit operating system. The machine is equipped with an Intel® Core™ i7-4770 CPU @ 3.40GHz, and 16 GB of RAM.

TABLE 1. EXPERIMENTS PARAMETERS

Parameter	Range
Period	10 – 100 s - step 10 s
Initial battery level	50 – 25 mJ - step 5 mJ
Execution cost	$2 \cdot 10^{-4} – 6 \cdot 10^{-4}$ mW
Execution time	7 – 22.5 ms

The algorithm is evaluated in different scenarios, each one characterized by a number of service requests and available things, and an average number of services exposed by each thing, expressed as a fraction of the overall number of service requests. For each scenario, one hundred different inputs are randomly generated. More specifically, the initial battery levels, computational costs and periods are drawn from a uniform distribution with parameters as reported in Table 1. Moreover, m_{ij}'s, i.e., context information about which services can be invoked on which things, are also randomly generated so that the average number of services per thing characterizing the scenario is fulfilled. Metrics of interest are then estimated for each scenario along with 95% confidence interval.

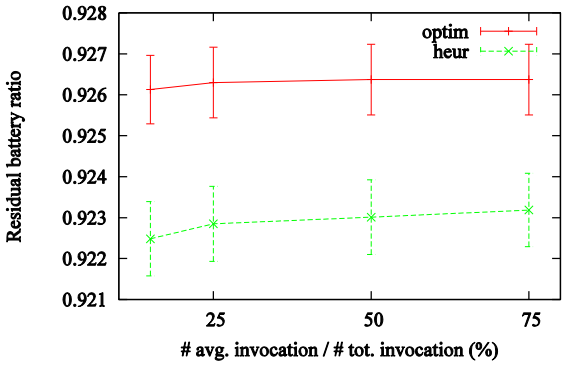

Figure 4. Min residual battery ratio (50 things, 500 requests).

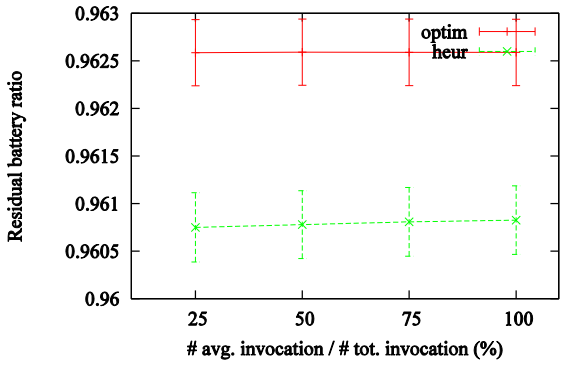

Figure 5. Min residual battery ratio (100 things, 500 requests).

Numerical experiments have been conducted with several combinations of the number of service requests and things. Results are similar in all considered scenario, therefore we limit in this work to report results to two different scenarios only. The first scenario is characterized by 50 things and 500 requests, while the second consists of 100 things and 500 requests. In both scenarios, the following average number of services exposed by each thing are considered (expressed as a fraction of the overall number of service requests, i.e., 500): 15%, 25%, 50%, 75% and 100%, respectively. To evaluate the performance of *RTTA*, we consider the *residual battery ratio* defined as the ratio between the battery level of a thing at the end of the hyper-period and the initial battery level.

Figure 4 and Figure 5 show the average minimum residual battery ratio over the set of different inputs for the first and second scenarios, respectively. The objective of RTTA is to maximize such ratio among all things, hence this metric can provide a measure of how far is the heuristic solution from the optimal one. As can be seen, in all cases the heuristic succeeds into finding an allocation which is very close to the optimum. The difference reduces as the percentage of the average number of service requests that a thing can satisfy increases. This means that *RTTA* is more efficient when there are more opportunities to allocate a service to thing, i.e., the solution space is larger. On the other hand, when low percentages are considered (e.g., 15%), i.e., the sets of equivalent things are small, we can notice that also the optimal solution decreases, though at a lower rate than the heuristic algorithm.

Figure 6 and Figure 7 illustrate the distribution of the residual battery level in the first and second scenario, respectively. The distribution is shown through the box-plot representation where the bottom and top of the box represent the 25th and 75th percentile, the band in the box the median (50th percentile), while the ends of the whiskers represent the minimum and 95th percentile. As can be seen, even if the objective function tries to maximize the remaining battery of the thing with less energy, the algorithm succeeds in distributing the energy consumption among all the things obtaining uniform battery consumption.

Finally, Figure 8 and Figure 9 illustrate the computation time required by the heuristic algorithm and the optimization tool to find the optimal allocation in the first and second scenario, respectively. As can be seen, the time required by the solver to find the optimal solution is at least an order of magnitude higher than the time required by the proposed algorithm to find an allocation that is comparable to the optimal value. It is also worth noting that the performance of the heuristic algorithm is only slightly affected by the size of the problem (100 things in the second scenario vs. 50 things in the first one), whereas this has a much higher impact on the solver, which take almost an additional order of magnitude of time to find the solution.

VI. Related Work

Research efforts aimed at deploying large-scale IoT systems are only recent. For this reason, although the QoS-aware service selection problem has been widely studied in traditional platforms, solutions specifically designed for IoT are still missing. To the best of our knowledge, [14] is the only

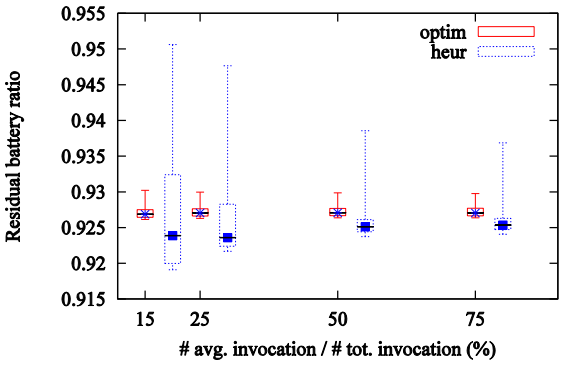

Figure 6. Residual battery ratio (50 things, 500 requests).

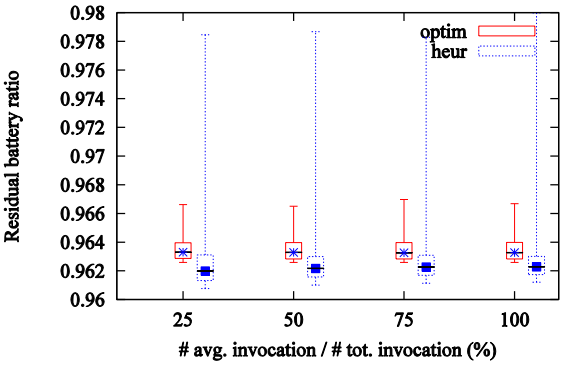

Figure 7. Residual battery ratio (100 things, 500 requests).

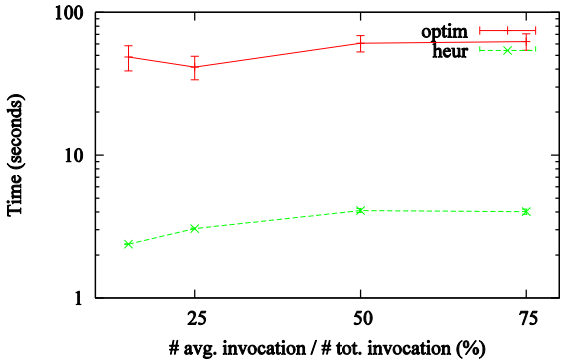

Figure 8. Computation time (50 things, 500 requests).

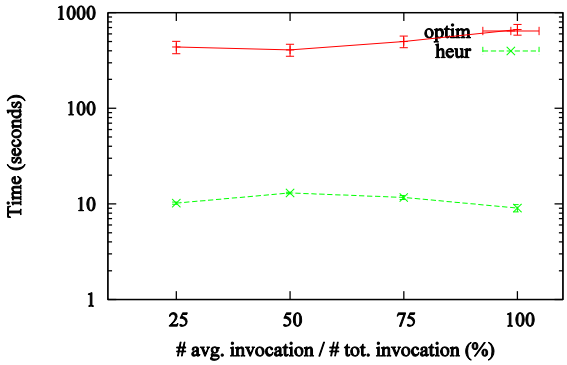

Figure 9.Computation time (100 things, 500 requests).

work proposing a QoS aware scheduling designed for service-oriented IoT platforms. A multi-layered scheduling model is proposed to evaluate the optimal allocation that meets the QoS requirements of applications. Different solutions are deployed at different layers to manage different system resources, such as network resources and IoT services. However, the proposed approach cannot be used for online IoT-service selection, since it is mainly suited for offline provisioning and planning. Our approach, instead, is designed for run-time service selection and takes into account real-time requirements of applications with stringent deadlines. In the field of SOA several solutions for QoS-aware service selection have been proposed. In particular, in Web Service architectures, support for QoS is a challenge considering the requirements of both users and service providers [15]. In [7] authors propose a QoS broker for web service composition that aims at finding the best combination in order to maximize the end-user satisfaction. The proposed approach is based on a utility function that takes into account only end-users without analyzing constraints from server providers. The first work that tries to meet both end-user and server provider requirements is [16], where the authors propose a QoS-aware adaptive load balancing strategy. The proposed solution, however, does not consider hard real-time requirements but only customer fixed priorities. Finally, in [17] the authors present a heuristic that aims at finding the optimal allocation that minimizes the overall response time. The proposed solution, however, provides only probabilistic guarantees that cannot support hard real-time applications.

Solutions proposed in the field of SOA and Web Service architectures are not suited for IoT deployments, as IoT systems are composed of service providers characterized by constrained capabilities with limited battery capacity. On the other hand, such characteristics are considered by design in solutions proposed in the field of Wireless Sensor Networks (WSN), in which several solutions for task assignment have been proposed. In [18] the authors present a novel methodology to assign tasks to sensors in order to minimize the overall energy consumption. The WSNs is considered composed of heterogeneous devices with different capabilities. The proposed solution aims at minimizing the overall energy consumption through a greedy approach: each task is assigned to the sensor that consumes less power. The proposed approach, however, is also specifically designed for WSNs and leverages a depth knowledge of the sensors and their connections, assumptions that do not apply to the general IoT field, which is usually agnostic to network structure and hardware capabilities.

VII. CONCLUSIONS

In this work we tackled the problem of QoS-aware service selection in IoT cloud-based architectures. First, we formally defined the problem of optimum energy-efficient service selection in which real-time QoS requirements are enforced considering context-information. We then derived a heuristic to solve the problem in a polynomial time in order to allow its implementation in real systems. The heuristic algorithm was validated through simulation that demonstrated that it guarantees an allocation close to the optimal value in polynomial time. As future work, we plan to extend the proposed solution so as to include support for sporadic service invocation. Moreover, we plan to experiment the QoS solution in real testbeds to validate the assumptions behind the problem formulation.

ACKNOWLEDGMENT

This work has been carried out within the activities of the project “Building the Environment for the Things as a Service (BETaaS),” which is co-funded by the European Commission under the Seventh Framework Programme (grant no. 317674).

REFERENCES

[1] Atzori, Luigi, Antonio Iera, and Giacomo Morabito. "The internet of things: A survey." Computer networks 54.15 (2010): 2787-2805.

[2] J. Kim; J. Lee; J. Kim; J. Yun, “M2M Service Platforms: Survey, Issues, and Enabling Technologies,” Communications Surveys & Tutorials, IEEE , 2014.

[3] Distefano, S.; Merlino, G.; Puliafito, A, “Enabling the Cloud of Things,” Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2012.

[4] E. Mingozzi, G. Tanganelli, C. Vallati, V. Di Gregorio, "An Open Framework for Accessing Things as a Service" in Proc. of the 16th International Symposium on Wireless Personal Multimedia Communications (WPMC 2013).

[5] Bonomi, F., Milito, R., Zhu, J., & Addepalli, S. Fog computing and its role in the internet of things. In Proc. of the first edition of the MCC workshop on Mobile cloud computing. ACM.2012

[6] Abdelwahab, S.; Hamdaoui, B.; Guizani, M.; Rayes, A, “Enabling Smart Cloud Services Through Remote Sensing: An Internet of Everything Enabler,” Internet of Things Journal, IEEE. 2014.

[7] Yu, Tao, Yue Zhang, and Kwei-Jay Lin. "Efficient algorithms for Web services selection with end-to-end QoS constraints." ACM Transactions on the Web. 2007.

[8] Anind K. Dey. Understanding and Using Context. Personal Ubiquitous Comput. 2001.

[9] E. Mingozzi, G. Tanganelli, C. Vallati, "A framework for QoS Negotiation in Things-as-a-Service oriented architectures" in Proc. of the 4th International Conference on Vehicular Technology, Information Theory, Aerospace and Electronics systems, and Communications. 2014.

[10] Lehoczky, J., Lui S., and Ye D. "The rate monotonic scheduling algorithm: Exact characterization and average case behavior." Real Time Systems Symposium, 1989.

[11] Pentico, D. W. "Assignment problems: A golden anniversary survey."European Journal of Operational Research. 2007.

[12] Martello, S., and Paolo T.. "The bottleneck generalized assignment problem." European Journal of Operational Research. 1995.

[13] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, Inc., New York, NY, USA. 1990.

[14] Ling Li; Shancang Li; Shanshan Zhao, "QoS-Aware Scheduling of Services-Oriented Internet of Things,"Industrial Informatics, IEEE Transactions on. 2014.

[15] Menasce, D. "QoS issues in web services."Internet Computing, IEEE. 2002.

[16] Boone, Bas, et al. "SALSA: QoS-aware load balancing for autonomous service brokering."Journal of Systems and Software. 2010.

[17] Menascé, D. A., E. Casalicchio, and V. Dubey. "On optimal service selection in service oriented architectures."Performance Evaluation 2010.

[18] Reinhardt, Andreas, and Ralf Steinmetz. "Exploiting platform heterogeneity in wireless sensor networks for cooperative data processing."Proc. of the GI/ITG KuVS Fachgespräch “Drahtlose Sensornetze”(August 2009).