A Framework for Efficient and Scalable Service Offloading in the Mist

Eugenio Rubio-Drosdov
Telematics Engineering Department
Universidad Carlos III de Madrid
Leganés, Spain
eurubiod@pa.uc3m.es

Daniel Díaz Sánchez
Telematics Engineering Department
Universidad Carlos III de Madrid
Leganés, Spain
dds@it.uc3m.es

Florina Almenárez
Telematics Engineering Department
Universidad Carlos III de Madrid
Leganés, Spain
florina@it.uc3m.es

Andrés Marín
Telematics Engineering Department
Universidad Carlos III de Madrid
Leganés, Spain
amarin@it.uc3m.es

***Abstract*— Mist computing puts computing on the very edge of the network. With mist computing, the computation is performed in users' devices (smartphones, laptops, etc.) sensors or wearables (something fairly common in the context of the Internet of Things). Devices which are near may be part of a mist where each of these devices contributes to computation tasks. Each of them has computation capabilities which can be used in order to run tasks from other devices. However, devices in the network must be aware of available features in order to execute tasks within a device with the necessary hardware requirements. There is a compatibility problem between tasks and devices: not all devices satisfy all tasks needs. This article proposes a mist computing framework in order to tackle this compatibility problem.**

Keywords—mist computing, offloading, computing architecture

I. Introduction

Today, most data from Internet of Things (IoT) devices is processed in the cloud, which means the data being produced in all devices is sent to a handful of computers located in centralized datacenters. However, with the quantity of IoT devices expected (50 billion by 2022) the volume and velocity with which data are being sent over the internet poses critical challenges to a cloud-only approach [1].

The growing interest in cloud computing has also resulted in other emerging cloud paradigms, such as fog computing. Fog computing is an extension of cloud computing services to the edge of the network to decrease latency and network congestion. Although both cloud and fog offer similar resources and services, the latter is characterized by low latency with a wider spread and geographically distributed nodes to support mobility and real-time interaction.

However, fog computing has high bandwidth requirements as all data must move through gateways and they are points of failure. A gateway executes applications and is responsible of the operation of the network [2]. Due to that, huge amount of data may cause a gateway to collapse.

In order to solve this, a new paradigm appears: mist computing. Mist computing takes fog computing concepts even further, locating computation into the very edge of the network: into nodes (Fig. 1).

This work has been supported by the Comunidad de Madrid (Spain) under the project CYNAMON (P2018/TCS-4566), co-financed by European Structural Funds (ESF end FEDER), project INRISCO: INcident monitoRing In Smart COmmunities (TEC2014-54335-C4-2-R) and project MAGOS: Secure Smart Grid Using Open Source Intelligence (TEC2017-84197-C4-3-R)

978-1-5386-4980-0/19/$31.00 ©2019 IEEE

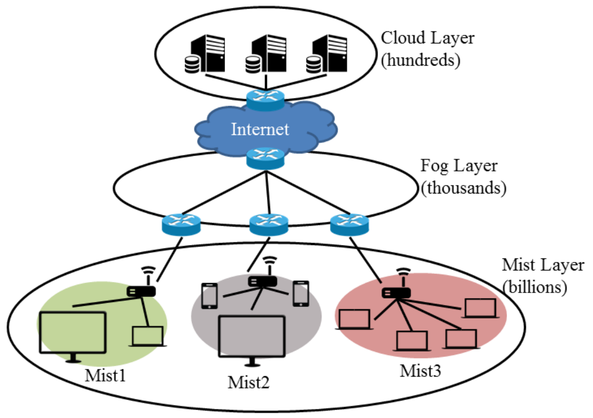

Fig 1. Cloud, Fog, and Mist layers.

Applying the service-based architecture to the mist, any node that has access to the network may subscribe to a service offered by any node of the network. However, services may be offloaded, i.e., migrated to a new instance in other node (for instance, because the current host is running out of battery). Therefore, the mist should select a destination node for the service. In other words, nodes composing mists collaborate in order to deploy services. These services can be moved from one node to other subject to mist behavior. However, deciding where to move a service is not trivial because of the nature of the services: they may need specific hardware to run properly.

The work presented in this paper posits an infrastructure for mist computing where hosting decision is based on service and node analysis in order to make more accurate offloading.

This paper is organized as follows. Section II provides an overview on offloading within cloud, fog or mist computing. Section III presents our framework design patter. Section IV displays the discussion of the implementation and testing results. Finally, section VI presents the main conclusions and future work directions.

II. Computation Offloading

The main goals of computation offloading are to save battery and minimize the execution time of tasks compared to local execution [3]. The election of the target depends on the offloading strategy, context, and network connection.

In cloud and fog computing, when a server has run out of resources, it offloads the additional tasks to another server within the data center. Offloading may take place from fog to cloud or vice versa. The location where to offload is significant because it determines what kind of algorithms need to be executed and the tradeoffs that must be considered: in order to minimize latency, the offloading location used to be close to the source.

Offloading from cloud to fog is done when the first one realizes that the latency requirements of an application cannot be achieved by the cloud computing environment. Both the fog and the cloud work together in a distributed way to fulfill an application's needs. However, there are applications (e.g., real-time applications) that require so low latency that cannot be met by distant locations. Due to that, offloading should take place into nearby devices (usually in the same network or physical location).

In order to solve this problem, mist computing appears. Mist computing pushes computation away from centralized virtual nodes to the extreme edge of the network where IoT devices are located [4]. The nearby devices that constitute mists have the capability of self-awareness and location-awareness in which it allows dynamic deployment of software based on context changes [5]. Mist computing provides computation offloading reducing fog and cloud storage and decreasing the latency and increasing the autonomy of a solution [6].

The main application of mist computing is to provide different services which have been distributed among the computing nodes [5]. Due to that, services live into nodes and any node may subscribe to them. However, nodes may be mobile, e.g., users smartphones or laptops, and may depend on running out batteries. Consequently, services hosted in those nodes may be required to be moved to other nodes when current hosting nodes are unreachable.

There are several works that delve into nearby offloading. The works [7] presents an algorithm to encourage vehicular cloud computing. This algorithm offloads tasks into nearby vehicle in order to minimize the delay. This work on vehicular cloud computing and others can be seen in [8], a survey of this topic. The framework presented with Misco [9] uses nearby devices splitting large computation into multiple smaller tasks. Shi et al. proposed Serendipity [10] where nearby devices with similar characteristics are selected to offload tasks. Honeybee [11] and Sapphire [12] offloads applications into other smartphones. Other works such as [13] [14] focus on code partitioning that allows offloading parts of code in different servers. The work presented in [15] and [16] are examples of offloading based in context-aware decision algorithms which offloads just code parts.

The framework presented in this paper proposes to extend decision patterns adding analysis of the nearby devices and the services taking into account the hardware compatibility between them. This extension makes possible to offload an entire service deployed in a mist and hosted in a node into other nearby node equipped with the hardware needed. We propose a design pattern which enables whole service offloading.

III. Design Pattern for Service Offloading

The design pattern is proposed in this section. It enables an IoT service deployed in a mist to be entirely offloaded.

A mist is composed by devices such as smartphones, laptops, personal computers, smartwatches, sensors, and so on. These devices in the mist are known as nodes. They live nearby and are connected to the same network. In order to do an accurate offloading of a whole service from one node into another, we propose to add two services into every node: (1) Node Summarizer (NS) and (2) Service Summarizer (SS).

The Node Summarizer is responsible for analyzing the node where it is hosted. To analyze a node implicates gathering information about the hardware connected to it such as audio and video input and output, sensors of any kind, and so on. Analogously, the Service Summarizer analyzes the services which are running in the host, concluding which hardware is needed to run properly (Fig. 2). Adding these two services in each node, minimizes complexity at the moment of adopting a new architecture. The mist is ready to host schedulers and orchestrators of any architecture: they only have to subscribe to NS and SS services and take decisions with the information provided.

Architectures based in NS and SS services are characterized by:

- Modularity: the whole system is composed of isolated components where each component contributes to the overall system behavior rather than having a single component that offers full functionality.
- Specialization: each service provides a support for an independent part of application's business logic.
- Soft maintenance: NS and SS are small pieces of code to debug and they are independent.
- Autonomy: each NS and SS can be developed, tested, deployed, destroyed, moved or duplicated independently and automatically.
- Evolution: The architecture stays maintainable while constantly evolving and adding or removing features.
- Multi-language programming: NS and SS provide APIs in different programming languages to access the gathered information.

Each Node Summarizer and Service Summarizer sends the gathered information to a mist coordinator in order to identify which offloading target node is the best option. We propose that the information exchanged be based on taxonomies and ontologies depending on whether the hardware in the mist is well known or not. There are several taxonomies that go deep into sensor or interface categorization, including IoT main sensors. In [17] [18] [19] [20], several sensor taxonomies are presented. The work in [21] proposes a taxonomy for tangible user interfaces. Choi et al. proposes a Brain-Computer Interfaces Taxonomy [22]. The work [23] presents separate identification ad categorization of the main sensors used in IoT services. In addition, ontologies in IoT are a field of interesting research. There are ontologies for sensors [24] [25] [26] [27] including sensors in smart home environments [28].

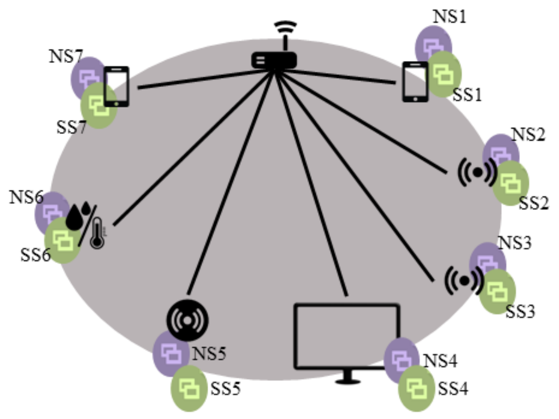

Fig. 2. In a mist, each node has a NS and a SS.

However, there are two approaches when deploying both Node Summarizer and Service Summarizer: (1) NS and SS are provided when deploying the architecture and (2) NS and SS are provided by the node. Fig. 3 shows the difference between these two options.

In first option NS and SS services are provided by the developer of the architecture. When a node becomes part of the mist, the orchestrator instantiates these two services into the node. However, the main problem that appears with this option is that the architecture developer should take into account the nature of the node. Due to that, the variety of nodes is limited: the orchestrator should know all the possible definitions of nodes, requiring save a high amount of data. Nevertheless, the developer of the node is freed from a significant workload.

In second option, the NS and SS are provided by the node. In this case, when a node becomes part of the mist, the architecture orchestrator simply subscribes to these two services. However, the information provided by these services, should fit into the taxonomy or the ontology which is used in the architecture. Due to that, the node developer should configure NS and SS taking into account the taxonomy or the ontology.

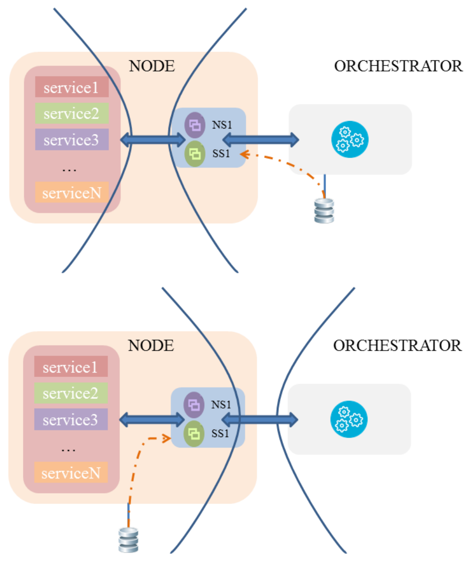

Fig. 3. NS and SS can be provided by the architecture or the node

This framework provides a vision of the mist as a platform where services can be deployed with the only requirement of taking into account the taxonomy or the ontology of the environment. Any node (not necessary part of the same mist) can ask for offloading a service (or a part of it) into the mist and, after a while disconnected, ask for the computing results.

IV. Evaluation

The goal of the evaluation is to validate whether adding node and service analysis improves the offloading decision in a reasonable time.

We applied our framework to support several services offloading: (1) a command-processor application [29] which needs a text-input interface to work properly, (2) a voice recorder, which needs a voice-input interface to work properly, and (3) a sound player, which needs an audio-output interface to work properly. As the context is well known, a taxonomy of our own has been used, based on the basic elements of the nodes of the experiment (i.e., keyboard, microphone and speaker). We used three nodes to create a small mist where services are deployed. The details of these nodes are listed as follows: (1) A node with 2.80 GHz four-cores CPU and 4 GB RAM with a keyboard, (2) a second node with 2.66GHz four-cores CPU and 2GB RAM with a keyboard, speaker and microphone, and (3) a node with 2GHz two-cores CPU and 1GB RAM with speaker and microphone.

We used the Raft Consensus Algorithm [30] in order to select which of the three nodes is the manager in charge of taking offloading decisions. The Raft Consensus Algorithm is designed to substitute the manager to other node if the previous manager is not available. Deciding where to offload the service is based on the information return from the NSs and DSs. This information responds to the taxonomy of our own, and it is formatted in a file noted in JSON (JavaScript Object Notation). The three nodes are configured to be unreachable at any moment (an internal process changes the state of the node).

When a node hosting a service is not reachable, the manager decides where to move the service. If there is not any node that can satisfy the needs of the service (e.g., a node without a speaker is not chosen to host the sound player service), it is moved to any node that can store but not run it. As soon as a node with the specification needed is able, the service is moved to it and initialized.

In order to conclude if the additional processing time is reasonable, we tested the framework several times and timed the time need to analyze the nodes and the services. As result, we found out that the average timing added when analyzing the node and services is less than 20 ms.

V. Conclusions

Offloading tasks from nodes to the fog or to the cloud may not be the best practice for applications which need low latency such as real-time applications. In order to diminish the delay between generating data and processing it, the target nodes where to offload tasks use to be nearby to the source of data. Here comes into play the mist, where nearby devices collaborate into computing tasks. The strongest aspect of deploying a mist is the low latency.

In this paper, we present a framework for service offloading in a mist. The design patter proposes to add two services to mist nodes that analyze the node and the hosted services. These two additional services minimize the complexity of deploying any architecture for the mist because the gathering of context information is resolved by them.

We did several experiments based on taxonomy of our own. We used the Raft Consensus Algorithm to decide which node takes offloading decisions.

As result of this work, the additional time of processing is less than 20 milliseconds, maintaining a low latency and improving the accurate of hosting a service: services are hosted in nodes which satisfy the needs of the service.

Our future work will continue the exploration of our design's potential. We want to minimize the additional timing needed analyzing services and nodes. The next step is to propose an architecture that fits in this framework and probe it in a real world scenario.

REFERENCES

[1] I. Yaqoob, E. Ahmed, I. A. T. Hashem, A. I. A. Ahmed, A. Gani, M. Imran, and M. Guizani, "Internet of things architecture: Recent advances, taxonomy, requirements, and open challenges," IEEE Wireless Communications, vol. 24, no. 3 , 10-16, June 2017.

[2] M. Vogler, J. M., Schleicher, C. Inzinger, and S. Dustdar, "DIANE-dynamic IoT application deployment," in 2015 IEEE International Conference on Mobile Services, pp. 298-305, June 2015.

[3] M. Golkarifard, J. Yang, A. Movaghar, and P. Hui, "A Hitchhiker's Guide to Computation Offloading: Opinions from Practitioners," in IEEE Communications Magazine, vol. 55, no. 7, pp. 193-199, July 2017.

[4] K. Skala, D. Davidovic, E. Afgan, I. Sovic, and Z. Sojat, "Scalable distributed computing hierarchy: Cloud, fog and dew computing," in Open Journal of Cloud Computing (OJCC), vol. 2, no. 1, pp. 16-24, 2015.

[5] J. S. Preden, K. Tammemae, A. Jantsch, M. Leier, A. Riid, and E. Calis, "The Benefits of Self-Awareness and Attention in Fog and Mist Computing," in Computer, vol. 48, no. 7, pp. 37-45, July 2015.

[6] G. Orsini, D. Bade, and W Lamersdorf, "Computing at the Mobile Edge: Designing Elastic Android Applications for Computation Offloading," in 2015 8th IFIP Wireless and Mobile Networking Conference (WMNC), pp. 112-119, October 2015.

[7] Y. Sun, X. Guo, S. Zhou, Z. Jiang, X. Liu, and Z. Niu, "Learning-Based Task Offloading for Vehicular Cloud Computing Systems," in EEE Int. Conf. Commun. (ICC) 2018, accepted.

[8] M. Whaiduzzaman, M. Sookhak, A. Gani, and R. Buyya, "A survey on vehicular cloud computing," in Journal of Network and Computer Applications, vol. 40, pp. 325-344, April 2014.

[9] A. Dou, V. Kalogeraki, D. Gunopulos, T. Mielikainen, and V. H. Tuulos, "Misco: a mapreduce framework for mobile systems," in Proceedings of the 3rd international conference on pervasive technologies related to assistive environments, June 2010.

[10] C. Shi, V. Lakafosis, M. H. Ammar, and E. W. Zegura, "Serendipity: enabling remote computing among intermittently connected mobile devices," In Proceedings of the thirteenth ACM international symposium on Mobile Ad Hoc Networking and Computing, pp. 145-154, June 2012.

[11] N. Fernando, S. W. Loke, and W. Rahayu, "Honeybee: A programming framework for mobile crowd computing," in International Conference on Mobile and Ubiquitous Systems: Computing, Networking, and Services, pp. 224-236, December 2012.

[12] I. Zhang, A. Szekeres, D. Van Aken, I. Ackerman, S.D. Gribble, A. Krishnamurthy, and H. M. Levy, "Customizable and Extensible Deployment for Mobile/Cloud Applications," in Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation, vol. 14, pp. 97-112, October 2014.

[13] R. Kemp, N. Palmer, T. Kielmann, and H. Bal, "Cuckoo: a computation offloading framework for smartphones," in International Conference on Mobile Computing, Applications, and Services, pp. 59-79, October 2010.

[14] Y. Zhang, G. Huang, X. Liu, W. Zhang, H. Mei, and S. Yang, "Refactoring android java code for on-demand computation offloading," in Proceedings of the ACM international conference on Object oriented programming systems languages and applications, vol. 47, no. 10, pp. 233-248, October 2012.

[15] Z. Cheng, P. Li, J. Wang, and S. Guo, "Just-in-time code offloading for wearable computing," in IEEE Transactions on Emerging Topics in Computing, vol. 3, no. 1, pp. 74-83, March 2015.

[16] A. Ravi, and S. K. Peddoju, "Handoff strategy for improving energy efficiency and cloud service availability for mobile devices," in Wireless Personal Communications, vol. 81, no. 1, pp. 101-132, 2015.

[17] I. Ali, A. Gani, I. Ahmedy, I. Yaqoob, S. Khan, and M. H. Anisi, "Data Collection in Smart Communities Using Sensor Cloud: Recent Advances, Taxonomy, and Future Research Directions," in IEEE Communications Magazine, vol. 56, no. 7, pp. 192-197, July 2018.

[18] J. Garrity, "Harnessing the Internet of Things for Global Development," March 2015.

[19] M. Fell, and H. Melin, "Roadmap for the Emerging 'Internet of Things',"in Carré & Strauss, 2014

[20] C. Chen, and S. Helal, S, "A device-centric approach to a safer internet of things," in Proceedings of the 2011 international workshop on Networking and object memories for the internet of things, pp. 1-6, September 2011.

[21] K. P. Fishkin, "A taxonomy for and analysis of tangible interfaces," in Personal and Ubiquitous Computing, vol. 8, no. 5, pp. 347-358, 2004.

[22] I. Choi, I. Rhiu, Y. Lee, M. H. Yun, and C. S. Nam, "A systematic review of hybrid brain-computer interfaces: Taxonomy and usability perspectives," in PloS one, vol. 12, no. 4, April 2017.

[23] V. Rozsa, M. Denisczwicz, M. L. Dutra, P. Ghodous, C. F. da Silva, N. Moayeri, and N. Figay, "An Application Domain-Based Taxonomy for IoT Sensors," in Proceedings of the 23rd ISPE Inc. International onference on Transdisciplinary Engineering, pp. 249-258, October 2016.

[24] P. Hirmer, M. Wieland, U. Breitenbücher, and B. Mitschang, "Dynamic ontology-based sensor binding," in East European Conference on Advances in Databases and Information Systems, pp. 323-337, August 2016.

[25] D. J. Russomanno, C. R. Kothari, and O. A. Thomas, "Building a Sensor Ontology: A Practical Approach Leveraging ISO and OGC Models," in Proceedings of the 2005 International Conference on Artificial Intelligence, ICAI 2005, vol. 2, pp. 637-643, June 2005.

[26] L. Xue, Y. Liu, P. Zeng, H. Yu, and Z. Shi, "An ontology based scheme for sensor description in context awareness system," in 2015 IEEE International Conference on Information and Automation, pp. 817-820, August 2015.

[27] M. Compton, P. Barnaghi, L. Bermudez, R. GarcíA-Castro, O. Corcho, S. Cox, and V. Huang, "The SSN ontology of the W3C semantic sensor network incubator group," in Web semantics: science, services and agents on the World Wide Web, vol. 17, pp. 25-32, 2012

[28] O. B. Sezer, S. Z. Can, and E. Dogdu, "Development of a smart home ontology and the implementation of a semantic sensor network simulator: An internet of things approach," in 2015 International Conference on Collaboration Technologies and Systems (CTS), pp. 12-18, June 2015.

[29] E. Rubio-Drosdov, D. Díaz-Sánchez, F. Almenárez, P. Arias-Cabarcos, and A. Marín, "Seamless human-device interaction in the internet of things," in IEEE Transactions on Consumer Electronics, vol. 63, no. 4, pp. 490-498, November 2017.

[30] D. Ongaro, and J. K. Ousterhout, "In search of an understandable consensus algorithm," in 2014 USENIX Annual Technical Conference, pp. 305-319, June 2014.