future internet

MDPI

Article

A Methodology Based on Computational Patterns for Offloading of Big Data Applications on Cloud-Edge Platforms

Beniamino Di Martino *, Salvatore Venticinque, Antonio Esposito and Salvatore D'Angelo

Dipartimento di Ingegneria, Universita' della Campania "Luigi Vanvitelli", 81031 Aversa (CE), Italy; salvatore.venticinque@unicampania.it (S.V.); antonio.esposito@unicampania.it (A.E.); salvatore.dangelo@unicampania.it (S.D.)

* Correspondence: beniamino.dimartino@unicampania.it

Received: 10 November 2019; Accepted: 28 January 2020; Published: 7 February 2020

Abstract: Internet of Things (IoT) is becoming a widespread reality, as interconnected smart devices and sensors have overtaken the IT market and invaded every aspect of the human life. This kind of development, while already foreseen by IT experts, implies additional stress to already congested networks, and may require further investments in computational power when considering centralized and Cloud based solutions. That is why a common trend is to rely on local resources, provided by smart devices themselves or by aggregators, to deal with part of the required computations: this is the base concept behind Fog Computing, which is becoming increasingly adopted as a distributed calculation solution. In this paper a methodology, initially developed within the TOREADOR European project for the distribution of Big Data computations over Cloud platforms, will be described and applied to an algorithm for the prediction of energy consumption on the basis of data coming from home sensors, already employed within the CoSSMic European Project. The objective is to demonstrate that, by applying such a methodology, it is possible to improve the calculation performances and reduce communication with centralized resources.

Keywords: fog computing; cloud computing; parallelizazion strategies; patterns

1. Introduction

One of the trends followed by IT experts in recent years has been the *"Cloudification"* of most of the existing software applications, and the consequent movement and storage of massive amount of data on large, remote servers. While the Cloud model offers tangible advantages and benefits, especially in terms of revenues, return on investments and better use of existing hardware structures, it still shows weak points. First of all, as data are not in direct possession of the customer, as it is most of the time stored remotely, security issues may arise. Second, but not less important, the simple fact that you need to reach a remote server to start a calculation and receive a result, can hinder the actual applications. Real time applications need to provide fast and immediate responses, which Cloud Platforms cannot always guarantee. Furthermore, Cloud is strongly dependant on Internet connection to operate: if there is a network failure, services simply cannot be reached. This represent a major difficulty when dealing with real time and potentially critical applications.

The **Internet of Things** strongly relies on real time to deliver results. Just imagine smart robots in factories: they need to analyse data coming from sensors immediately, to react to the environment accordingly. If all the calculations were made in Cloud, delays in communications could slow the work or result in potential safety threats. Also, under a more general perspective, the huge amount of data to be transferred using current Internet networks could further aggravate local congestion and cause communication failures.

The answer to such issues can be found in the introduction of an intermediate layer, between local smart devices and the Cloud, in which most (if not all) of the real-time calculations can be executed, strongly reducing the impact on the network and delays. **Fog Computing** promises to act as such an intermediate level, by bringing computational power to the very edge of the applications' architecture, in particular by increasing the computing capabilities of devices themselves or of local gateways and aggregators: this would reduce the amount of data to be transferred, analysed and processed by Cloud server, which could focus on storage and take care of the heaviest computations that cannot be handled by local devices.

Fog and Edge computing shift most of the computing burden to peripheral devices and gateways, which can communicate with each other as long as a local network is up. Such local networks are most of the time separated from the Internet, they are generally created ad-hoc and are self maintained and managed. Communication failures are then handled locally, and are not dependant on the public network's status.

Having an infrastructure to handle real-time and critical computations locally and reduce data traffic represent a huge advantage of Fog Computing, but it is not enough: it is also necessary to accurately restructure the computation in order to take advantage of the infrastructure, and in particular to balance the computational burden weighting on the calculation nodes.

In this paper we present a data and computation distribution methodology, initially developed to distribute Big Data computation over Cloud Services within the TOREADOR European project [1], and we apply it to parallelization and distribution of algorithms created within the CoSSMic European Project [2] for the calculation and prediction of energy consumption in households, by exploiting local smart devices and gateways. The application of the computation distribution methodology allows for the exploitation of computational resources available at the edge of the software network, and for the balancing of computational loads, which will distributed in order to minimize the needing of a central Cloud based server. The remainder of this paper is organized as follows: Section 2 will present related works in the field of Field Computing and computation distribution; Section 3 will present the methodology; Section 4 will describe the Case Study used to test the approach; Section 5 will provide experimental results conducted within the Case Study; Section 6 closes the paper with final consideration on the current work and future developments.

2. Related Works

Exploiting the mobile framework to develop distributed applications can open to new interesting scenarios [3]. Among such applications, Smart Grid related platforms can take great advantages from the application of Fog Computing paradigms. The work presented in [4] describes such advances, focusing on challenges such as latency issues, location awareness and transmission of large amounts of data. In order to reduce the high latencies that may potentially affect real-time applications which exchange high volumes of data with Cloud services, there is the need of a shift in the whole computation paradigm: Fog Computing moves the Cloud Computing paradigm to the edge of networks, in particular those that connect all the devices belonging to the IoT [5].

A commonly accepted definition of Fog Computing, provided in [6], describes the concept as a scenario where a high number of wireless devices communicate with each other, relying on local network services, to cooperate and support intense computational processes.

Fog can thus represent an extension of Cloud Computing, an intermediate dedicated level of interconnections between the Cloud and end devices, bringing benefits like reduced traffic and latencies, and better data protection. The work presented in [7] shows that applications residing on Fog nodes are not simply isolated, but they are integrated in a larger solutions that covers Cloud and user level. Fog nodes are fundamental to collect and forward data for real time processing, but Cloud resources are still necessary to run complex calculations, such as in Big Data analytic.

The work presented in [8] provides an insight on the application of Fog computing to Smart Grids, focusing on the Fog Service Placement Problem, in order to investigate the optimal deployment of

IoT applications on end devices and local gateways. Code-based approaches, that is methodologies that start from an algorithm source code and try to obtain a distributed version of it, have also been investigated in different studies. In [9] the authors have described an innovative auto-parallelizing approach, based on a compiler which implements data flow algorithm. Such an approach leverages domain knowledge as well as high-level semantics of mathematical operations to find the best distributions of data and processing tasks over computing nodes.

Several studies have stressed the important role played by network communications when applications need to to either transfer considerable amounts of data or rapidly exchange information to provide real-time responses, such as in [10]. Indeed data transmission, especially when the volume becomes consistent, is more prone to bit errors, packet dropping and high latency. Also, access networks can contribute to the overall data transmission time, sometimes being determinant [11]. In [12] authors have provided an insight on the issues that transmission traffic can cause to mobile communications, even when the amount of data to be exchanged is relatively small, and have also proposed solutions to resolve the problem in the specific case of Heartbeat Messages. However, since the transmission of considerable amounts of data is still problematic, the re-distribution of workloads over the end devices and the consequent reduction of traffic seem to be the better option, provided that the different capabilities of Cloud and mobile resources are taken in consideration. The data-driven reallocation of tasks on Edge devices has been considered in [13], with a focus on machine-learning algorithms.

Edge devices generally come with limited computational power and need to tackle energy consumption issues, which also arise in hybrid mobile-Cloud contexts, as pointed out in [12], where authors provide their own solution to the issue. Energy consumption is also the main issue considered in [14], where the authors propose an Online Market mechanism to favour the partecipation of distributed cloudlets in Emergency Demand Response (EDR) programs.

The aim of our work is to achieve data and task based reallocation of computation over Edge devices, by guiding the re-engineering of existing applications through Computational patterns. The approach presented in Section 3 is indeed based on the use of annotation, via pre-defined parallelization primitives, of existing source code, in order to determine the pattern to be used. The use of patterns can help in automatically determining the best distribution algorithm to reduce the data exchange and, depending on the user final objective, also to minimize energy consumption.

3. Description of the Methodology

The methodology we exploit to distribute and balance the computation on edge nodes works through the annotation, via pre-defined parallelization directives, of the sequential implementation of an algorithm. The approach, has been designed within the research activities of the TOREADOR project, has been specifically developed to distribute computational load among nodes hosted by different platforms/technologies in multi-platform Big Data and Cloud environments, using State of the Art orchestrators [15]. Use cases where edge computing nodes represented the main target have been considered and demonstrated the feasibility of the approach in Edge and Fog Computing environments [16].

The methodology requires that the user annotates her source code with a set of **Parallelization Primitives** or parallel directives, which are then analysed by a compiler. The compiler determines the exact operations to execute on the original sequential code, thanks to a set of transformation rules which are unique for the specific parallel directive, and employs Skeletons to create the final executable programs. Directives are modelled after well known Parallelization Patterns, which are implemented and adapted according to the considered target. Figure 1 provides an overview of the whole parallelization process, with its three main steps:

1. **Code**: in the first step, we suppose the user owns a good knowledge of the original algorithm to be transformed from a sequential to a parallel version. The user will annotate the original code with the provided Parallel Primitives.

2. **Transform**: The second step consists in the actual transformation of the sequential code, that the user has annotated with the aforementioned primitives, operated by Skeleton-Based Code Compiler (Source to source Transformer). The compiler will produce a series of parallel versions of the original code, each one customized for a specific platform/technology, according to a 3-phases sub-workflow.

   (a) *Parallel Pattern Selection*: on the base of the used primitives, the compiler selects (or asks/helps the user to select) a Parallel Paradigm.
   (b) *Incarnation of agnostic Skeletons*: this is the phase in which the transformation takes place. A parallel agnostic version of the original sequential algorithm will be created, via the incarnation of predefined code Skeletons. Transformation rules, part of the the knowledge base of the compiler, guide the whole transformation and the operation the compiler will perform on the Abstract Syntax Tree.
   (c) *Production of technology dependent Skeletons*: the agnostic Skeletons produced in the previous phase are specialized and multiple parallel versions of the code are created, considering different platform and technologies as a target

3. **Deployment**: Production of Deployment Scripts.

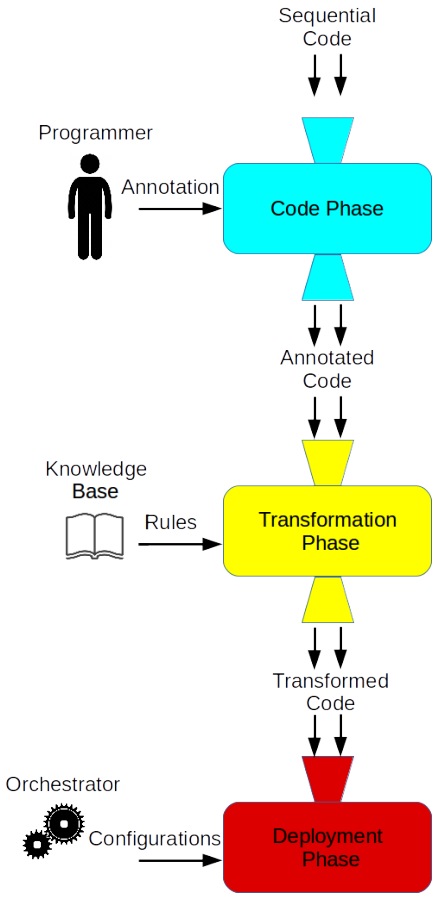

Figure 1. The Code-Based Approach workflow.

3.1. The Code Phase

As already stated, in our approach the user is considered an expert programmer, who is aware of the specific parts of her code that can, or cannot, be actually parallelized. However, once she has annotated the sequential code with parallel primitives, these will allow her to distribute the sequential computation among processing nodes residing on remote platforms and even in multi-platform environments.

The parallel directives used to guide the next transformation phase have well known meaning within the approach, and have been studied to adapt to most of the situations. Also, directives can be nested to achieve several levels of parallelization.

Primitives can be roughly divided into two main categories:

- **Data Parallel** primitives, which organize the distribution of data to be consumed among processing nodes. General primitives exist, which do not refer to a specific Parallel Pattern, but most of the primitives are modelled against one.
- **Task Parallel** primitives, which instead focus on the process, and distribute computing loads according to a Pattern based schema. General primitives also exist, but they will need to be bound to a Specific Pattern before the transformations phase.

Examples of used primitives are:

- The ***data_parallel_region(data, func, *params)*** represents the most general data parallelization diretive, which applies a generic set of functions to input data and optional parameters.
- The ***producer_consumer(data, prod_func, cons_func, *params)*** directive implies a Producer Consumer parallelization approach. The ***data*** input is split into independent chunks which represent the input of the ***prod_func*** function. A set of computing node elaborates the received chunk and puts the result in a shared support data structure (Queue, List, Stack or similar), as also shown in Figure 2. Another set of nodes polls the shared data structure and executes ***cons_func*** on the contained data, until exhaustion.
- The ***pipeline(data, [order_func_list] *params)*** directive represents a well known task parallel paradigm, in which the processing functions to be executed need to be run in the precise order they appear in the ***order_func_list*** input list.

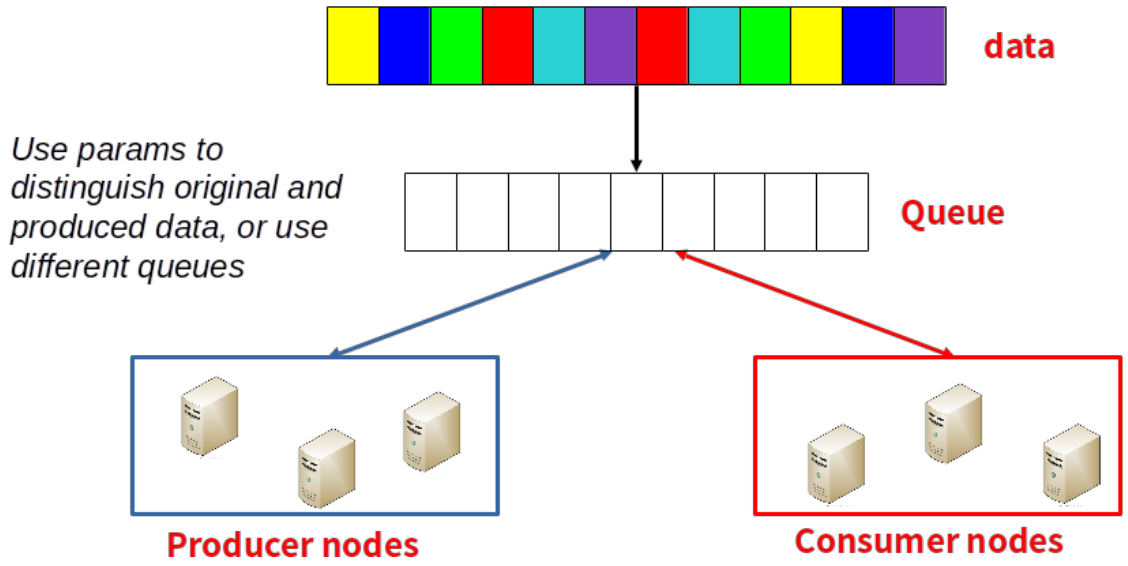

Figure 2. Producer Consumer.

The input ***data*** is generally a list of elements, which are fed one by one to the first function, whose output is passed to the second one and so on, until the last result is obtained. While the i th function is in execution, the $i-1$ th can elaborate another chunk of data (if any), while the $i+1$ th needs to wait for the output from the i th in order to go on. In this way, at regimen, no computational nodes are idle and resources are fully exploited. Figure 3 reports an example of execution of such a Primitive, showing how functions are sequentially executed by Computing nodes and fill the pipeline.

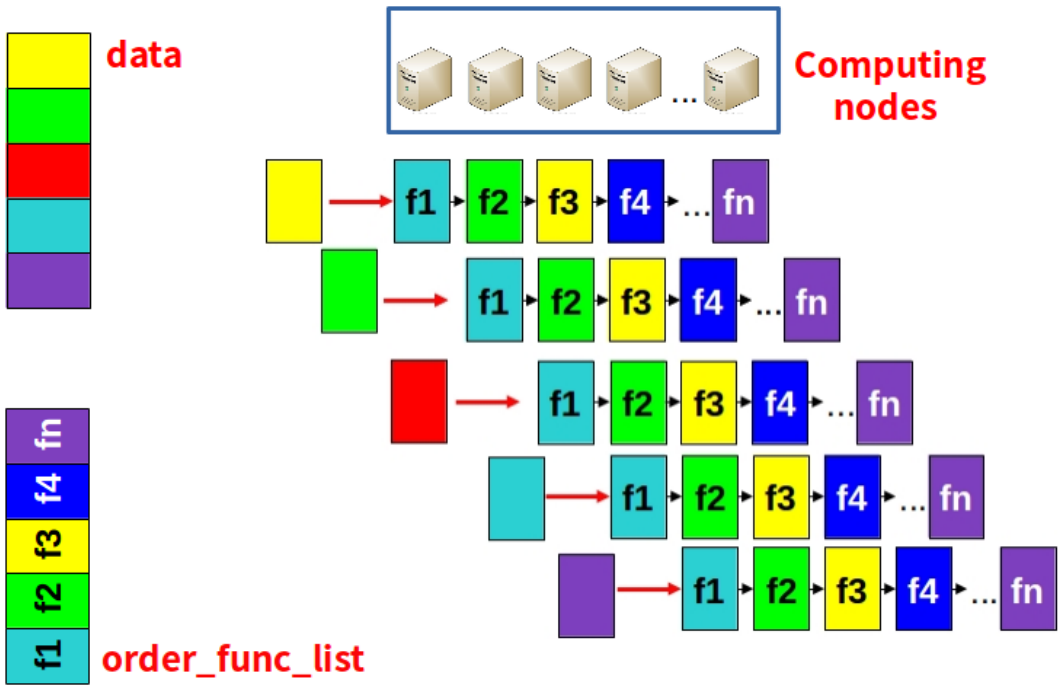

Figure 3. Pipeline.

3.2. Transformation Phase

The Transformation phase represents the core of the entire approach. The annotated code is analysed and, when the parallelization directives are met, a series of transformation rules are applied accordingly.

Such rules are strictly dependent on the Pattern the specific directive has been modeled against, so if the user selects a general primitive she is asked to make a decision at this step.

The final Skeletons obtained after the filling operations can be roughly divided into three categories:

- **Main Scripts** contain the execution "main", that is the entry point of the parallelized application, whose processing is managed by the Skeleton. All code that cannot be distributed is contained within the Main Script, which will also take care of calling the Secondary Scripts;
- **Secondary Scripts** contain the distributed code, that will be directly called by the Main Script and then executed on different computational nodes, according to the selected Parallel Paradigm. The number of secondary scripts is not fixed, as it depends on the selected Pattern;
- **Deployment Templates** provide information regarding all the computational nodes that will be used to execute the filled Skeletons (both Main and Secondary)

The knowledge base of the compiler comprehends a series of Skeleton filling rules, which are used to analyze and transform the original algorithm. The rules are bound to a specific Pattern, as the transformations needed on the sequential code will change by selecting a different Pattern. However, since the Parser will treat the micro-functions contained in the algorithm definition and included in the analyzed primitives always in the same way, despite the specific Pattern selected, the rules are completely independent from the algorithm.

3.3. The Deployment Phase

The Deployment step is the last one which needs to performed in order to make the desired algorithm executable on the target platform. The user does not intervene during the Deployment phase, as it is completely transparent to her, unless she wants to monitor and dynamically act on the execution of the algorithm.

Different target deployment environments have been considered:

- Distributed and parallel frameworks, among which Apache Storm, Spark and Map-Reduce
- Several Cloud Platforms, as an instance Amazon and Azure
- Container-based systems, with a focus on Docker, for which the approach considers two different parallelization strategies:
  - A centralized strategy, where a central server manages the Docker containers. The server can reside, but non necessarily, in a Cloud environment.
  - A distributed strategy, in which a central offloading algorithm takes care of allocating containers on remote devices, selected by following the execution schema. This second approach can be applied in the case of Edge and Fog Computing approaches, as also reported in previous works [16] and further investigated in the present manuscript.

Automatic orchestrators can be employed, if the target environment allows it, as it has been described in [15].

4. The Case Study

The CoSSMic project focuses on the creation and management of **Microgrids**, that is local power grids confined within smart homes or buildings (even adhibited to offices) embedding local generation and storage of solar energy, together with power consuming devices. Such devices comprehend electric vehicles that can connect and disconnect dynamically and therefore introduce variability in the storage capacity.

The CoSSMic user can configure all of her appliances, according to a set of constraints and preferences: earlier or latest start time, priority in respect to other appliances, duration of the work and so on. Also, she can supervise the energy consumption, determine how much power is produced by the local solar facility and how much is shared with the community. All these information help to determine, and ultimately to reduce, the overall costs. The user can also set specific goals: reduce battery use, or maximize the consumption of energy from solar panels.

In order to determine the best course of actions, according to the constraints and goals of the user, a Multi Agent System (MAS) has been exploited to deploy agents that actively participate in the energy distribution. Agents make use of the information coming from the user's plan, the weather forecast and the consumption profiles of the appliances to find the optimal schedule, which will maximize the neighborhood grid self-consumption.

The main configuration of the CoSSMic platform is **All-In-Home**, that is all the software resides on a Home Gateway, which is connected to the local power grid and to the Internet, and encapsulates the functions of device management, information system and MAS. The computation for the energy optimization is performed at each home, and the energy exchange occurs within the neighborhood. Cloud services can be used by agents to storage info about energy consumption.

In order to optimize the energy management, the local nodes execute a series of computations to determine the consumption profiles of the several devices and appliances connected to the CoSSMic microgrid. The consumption data coming of each device are analysed and consumption profiles are built. The calculation of such profiles is fundamental to foresee the future behaviour of the devices and create an optimized utilization scheduling.

In the original CoSSMic prototype users need to set in advance which kind of program they are running manually to allow to the system for taking into account energy requirements of that program. This is a tedious task. Moreover, it needs to run many times the same program of the appliance before an average profile that represents energy requirements of that program is available. K-means allows for implementing an extended functionality of the CoSSMic platform to automatically learn energy profiles corresponding to different working programs of an appliance, and can be used to predict at device switch time which program is actually going to run. K-means clustering is used to group similar measured energy consumption time-series. Each cluster corresponds to a different working program. The centroid of each cluster is used to predict the energy consumption when the same program is

starting. The clusters are updated after each run of the appliance in order to use the most recent and significant measures. Collecting and clustering of measures coming from many instances of the same appliance could help to increase precision, but would require greater computational resources. Automatic prediction about which program is going to start is out of the scope here, but the interested reader can refer to [8].

K-means algorithm can be parallelized and distributed over Fog nodes, in order to achieve better performances and lessen the load burden on each node. Indeed, in order to fully exploit the Fog stratum, we need to rethink the distribution of the computation to also determine the best hardware allocation: this is where the application of our approach comes into play.

In our case study we are focusing on the parallelization of the Clustering algorithms, with particular attention to the k-means implementation. As it will be shown in Section 5 through code-examples, the *task_parallel_region* primitives will be mainly used, together with a distributed container approach, as seen in Section 3.3. The **Bag of Tasks** Parallel Pattern will be used in our test case.

5. Application of the Approach and Experimental Results

In this Section we are going to show how we have applied our approach by using a specific parallel primitive, and we confront the results obtained taking by running the parallelized code on two Raspberry PIs and the sequential code on a centralized server, acting as a Gateway.

In particular, we have focused on the parallelization of a Clustering algorithm, which is executed on a single device (the Home Gateway) in the current CoSSmic scenario. In the following, we will use Python as a reference programming language.

The sequential program run on the Gateway is simply started through the execution of a **compute_cluster** function, whose signature is as follows:

compute_cluster(run, data, history) where **run** is the maximum number of consecutive iterations the clusterization algorithm can run before stopping and giving a result, **data** is a reference to the data to be analyzed and **history** reports the cluster configuration obtained at the previous run.

The algorithms has been built in order to be embarrassingly parallel, so a data or task parallel region can be immediately adopted.

As shown in Listing 1 we first provide a definition of the Task Parallel primitive, then we pass the arguments which should be fed to the clusterization function to a **parser** in order to format and prepare them for the parallel execution. Such arguments are necessary to execute the parallel function and to correctly and store the input/output data. Finally we simply recall the Task Parallel Region primitive using the function to be parallelized as one of its arguments.

Listing 1: Task Parallel Primitive: definition and application.

```
#definition of the primitive
def task_parallel_docker(items, docker, func, *args):
avg = len(items) / float(docker)
out = []
last = 0.0

while last < len(items):
out.append(items[int(last):int(last + avg)])
last += avg

return [func(x, *args) for x in out]

#example of application

if __name__ == "__main__":
start = time.time()
```

```
usage = "usage: %prog [options] filename"
parser = ArgumentParser(usage)

parser.add_argument("filename", metavar="FILENAME")

parser.add_argument("-r", "--runs", default=0,
dest="runs", metavar="RUNS", type=int,
help="number of runs")

parser.add_argument("-d", "--docker", default=0,
dest="docker", metavar="DOCKER", type=int,
help="number of dockers")

parser.add_argument("-n", "--history", default=0,
dest="history", metavar="RUNID", type=int,
help="number of timeseries for clustering")

data_dir = "./paper_data"
args = parser.parse_args()
history = args.history
filename = args.filename
docker = args.docker

task_parallel_docker(list(range(args.runs)), docker, compute_cluster,
filename, history)
```

The Yaml configuration used to set-up the Dockers running on the final device has been provided in Listing 2. In the proposed configuration, one master and 4 slaves have been taken in consideration. The provided code only reports the configuration of the master and of one of the slaves, as they are all identical. In particular, the instructions that will be executed by the master and the slaves are included in two python files, which will be, in the future, automatically produced by a parser.

Listing 2: Master and Slave configurations.

```
master:
build: .
depends_on:
- redis
command: python /fog/docker_master.py
volumes:
- type: bind
source: ./
target: /fog
networks:
redis-network:
ipv4_address:
stdin_open: true
tty: true

slave0:
build: .
command: python /fog/docker_slave.py
depends_on:
- redis
- master
restart: on-failure
```

```
volumes:
- type: bind
source: ./
target: /fog
networks:
- redis-network
stdin_open: true
tty: true
```

The Dockers will run on the target environment simultaneously, being it a Raspberry or the centralized server. Considering the Raspberries, each of the Dockers will run on a different virtual CPU. Observing the measurements reported in Figure 4 it is clear that CPU001 is in charge of the master Docker and of one of the Slaves. Also, from Figure 5 it is possible to determine that not many process switches take place during the execution. Overall, the Raspberry are not overwhelmed by the computations, so they can be still be exploited for other concurrent tasks.

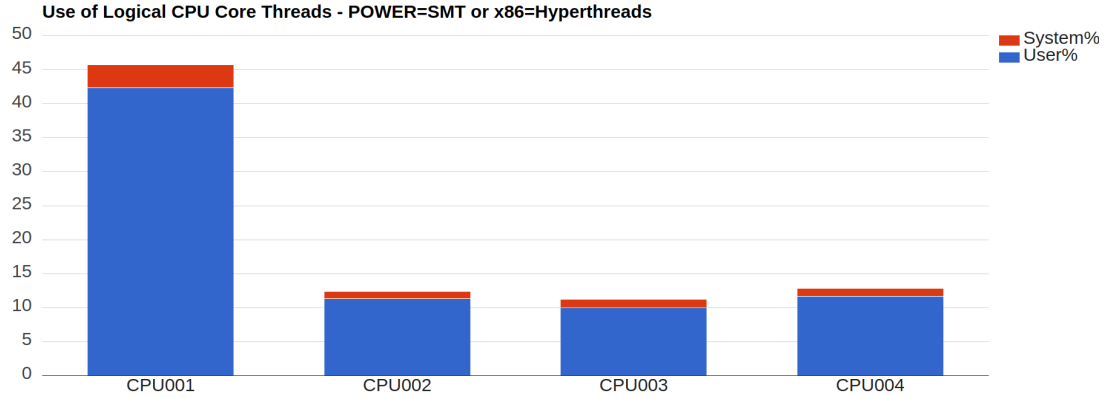

Figure 4. CPU utilization in one of the Raspberries.

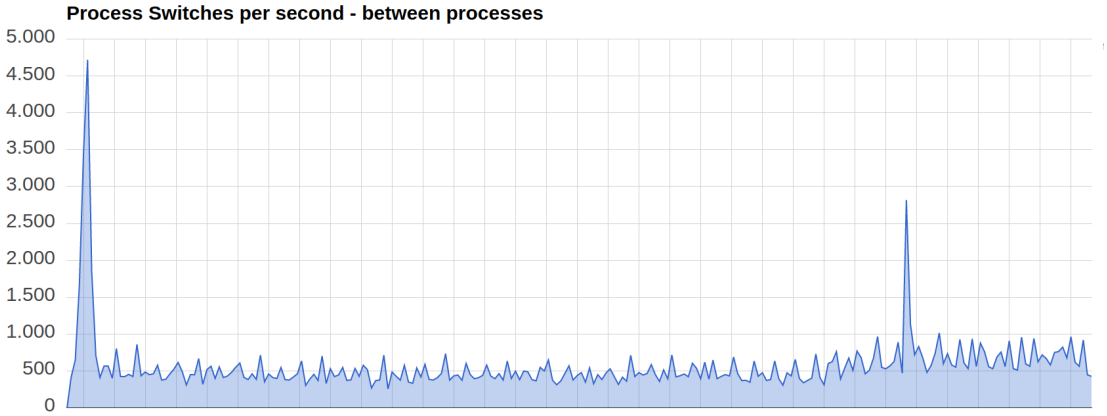

Figure 5. Process Switching during execution.

The execution times are, of course, much different if we compare the Raspberries with the centralized server.

As shown in Figure 6, the centralized server is far more efficient than the single Raspberries, the performances of which slightly differ from one another. However, this last fact is simply due to the difference in the reading speed of the SD cards used in the two Raspberries, despite it being rather small: 22.9 MB/s for Raspberry 1 and 23.1 MB/s for Raspberry 2.

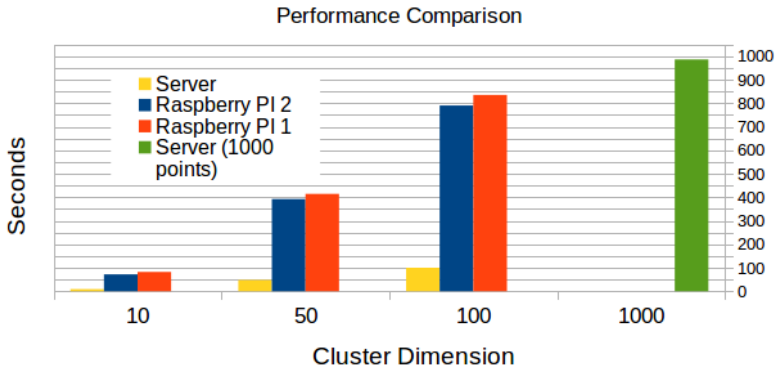

Figure 6. Comparison between Execution Times for different cluster dimensions.

If we take in consideration the medium size of a cluster, which in our case has be considered to be of 1000 points, the server will take 986 s on average to complete the computation. If we consider a configuration with N Raspberries, with each of them being given a portion of the points to be clustered, we would roughly obtain an execution time of 800 s if we considered 10 Raspberries working in parallel, and not completely focused on the specific clusterization task.

We are not taking in consideration data transmission times at them moment, as all data will be transmitted within the same local network, with small to negligible delays.

Furthermore, the Raspberries would be available to host the computation of data coming from different households in the neighborhood, provided they can access a common communication network, as in the current CoSSMic scenario.

6. Conclusions and Future Work

In this paper, an approach for the parallelization of algorithms on distributed devices, residing in a Fog Computing layer, has been presented. In particular, the approach has been tested against a real case scenario, provided by the CoSSMic European Project, regarding the clusterization and classification of data coming from sensors previously installed in households. Such clusters are then used to predict energy consumption and plan the use of the devices to maximize the use of solar energy.

What we wanted to achieve was to demonstrate that, through the application of the approach, it is possible to obtain a performance improvement in the algorithm execution time.

The initial results seem promising, as with the opportune configuration of data and tasks it is possible to obtain a sensible enhancement of the algorithm performances, provided that a sufficient number of devices (in the test case we used Raspberry PIs) are available.

However, the approach needs to be polished and to be completely automated, in order to reduce possible setbacks in the selection of the right configuration and to support the auto-tuning of the data and tasks distribution. Furthermore, in the future there will the possibility to automatically detect parallelizable sections of code to support the user in the annotation phase, or possibly to even completely automatize the whole annotation step.

Author Contributions: Conceptualization and supervision: B.D.M.; methodolody: B.D.M., S.V. and A.E.; software: S.D.; data curation: S.D. and A.E.; writing–original draft preparation, writing–review and editing: B.D.M., S.V., A.E., S.D. All authors have read and agree to the published version of the manuscript.

Funding: This work has received funding from the European Union's Horizon 2020 research and innovation programme under the TOREADOR project, grant agreement Number 688797 and the CoSSMic project (Collaborating Smart Solar powered Micro grids - FP7 SMARTCITIES 2013 - Project ID: 608806).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Toreador: TrustwOrthy model-awaRE Analytics Data platfORm. Available online: http://www.toreador-project.eu/ (accessed on 30 October 2019).
2. Collaborating Smart Solar-Powered Micro-Grids. Available online: https://cordis.europa.eu/project/rcn/110134/en/ (accessed on 24 September 2019).
3. Liu, F.; Shu, P.; Jin, H.; Ding, L.; Yu, J.; Niu, D.; Li, B. Gearing resource-poor mobile devices with powerful clouds: Architectures, challenges, and applications. *IEEE Wirel. Commun.* **2013**, *20*, 14–22.
4. Gia, T.N.; Jiang, M.; Rahmani, A.M.; Westerlund, T.; Liljeberg, P.; Tenhunen, H. Fog computing in healthcare internet of things: A case study on ecg feature extraction. In Proceedings of the 2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing, Liverpool, UK, 26–28 October 2015.
5. Bonomi, F.; Milito, R.; Zhu, J.; Addepalli, S. Fog Computing and Its Role in the Internet of Things. In Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing, Helsinki, Finland, 17 August 2012. [CrossRef]
6. Vaquero, L.M.; Rodero-Merino, L. Finding Your Way in the Fog: Towards a Comprehensive Definition of Fog Computing. *SIGCOMM Comput. Commun. Rev.* **2014**, *44*, 27–32. [CrossRef]
7. Skarlat, O.; Nardelli, M.; Schulte, S.; Borkowski, M.; Leitner, P. Optimized IoT service placement in the fog. *Serv. Oriented Comput. Appl.* **2017**, *11*, 427–443. [CrossRef]
8. Venticinque, S.; Amato, A. A methodology for deployment of IoT application in fog. *J. Ambient Intell. Hum. Comput.* **2019**, *10*, 1955–1976. [CrossRef]
9. Totoni, E.; Anderson, T.A.; Shpeisman, T. HPAT: High performance analytics with scripting ease-of-use. In Proceedings of the International Conference on Supercomputing, Chicago, IL, USA, 14–16 June 2017.
10. Yi, X.; Liu, F.; Liu, J.; Jin, H. Building a network highway for big data: Architecture and challenges. *IEEE Netw.* **2014**, *28*, 5–13. [CrossRef]
11. Gao, B.; Zhou, Z.; Liu, F.; Xu, F. Winning at the Starting Line: Joint Network Selection and Service Placement for Mobile Edge Computing. In Proceedings of the IEEE INFOCOM 2019—IEEE Conference on Computer Communications, Paris, France, 29 April–2 May 2019. [CrossRef]
12. Jin, Y.; Liu, F.; Yi, X.; Chen, M. Reducing Cellular Signaling Traffic for Heartbeat Messages via Energy-Efficient D2D Forwarding. In Proceedings of the 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS), Atlanta, GA, USA, 5–8 June 2017. [CrossRef]
13. Chen, Q.; Zheng, Z.; Hu, C.; Wang, D.; Liu, F. Data-driven Task Allocation for Multi-task Transfer Learning on the Edge. In Proceedings of the 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS), Dallas, TX, USA, 7–10 July 2019. [CrossRef]
14. Chen, S.; Jiao, L.; Wang, L.; Liu, F. An Online Market Mechanism for Edge Emergency Demand Response via Cloudlet Control. In Proceedings of the IEEE INFOCOM 2019—IEEE Conference on Computer Communications, Paris, France, 29 April–2 May 2019. [CrossRef]
15. Di Martino, B.; D'Angelo, S.; Esposito, A.; Martinez, I.; Montero, J.; Pariente Lobo, T. Parallelization and Deployment of Big Data algorithms: the TOREADOR approach. In Proceedings of the 2018 32nd International Conference on Advanced Information Networking and Applications Workshops (WAINA), Krakow, Poland, 16–18 May 2018.
16. Di Martino, B.; Esposito, A.; D'Angelo, S.; Maisto, S.A.; Nacchia, S. A Compiler for Agnostic Programming and Deployment of Big Data Analytics on Multiple Platforms. *IEEE Trans. Parallel Distrib. Syst.* **2019**, *30*, 1920–1931. [CrossRef]