A cloud-based parallel system for locating customers in indoor malls

Pedro Álvarez[1], Noelia Hernández[2], Javier Fabra[1], and Manuel Ocaña[3]

[1] University of Zaragoza, Spain, {alvaper,jfabra}@unizar.es
[2] University of Alcalá, Computer Engineering Department, Spain, noelia.hernandez@uah.es
[3] University of Alcalá, Department of Electronics, Spain, mocana@depeca.uah.es

Abstract. Advances in techniques of locating mobile users have promoted the development of marketing campaigns based on customers' location. WiFi-based location methods have proven their usefulness in tracking and locating customers within a indoor mall. Nevertheless, in some cases the performance of these methods prevents them from being used in real scenarios. In this paper, we have faced the problem of improving the execution time and reducing the cost of one of these WiFi-based location methods. Parallel programming techniques, service-oriented technologies and the cloud computing paradigm have been combined to solve efficiently these problems. The resulting system has been deployed in the *Amazon EC2 environment*, evaluating different configuration and deployment options.

Keywords: WiFi-based location · Parallel algorithms · Cloud computing · Time and cost analysis

1 Introduction

The BAI4SOW project [5] provides a software platform to create marketing campaigns based on three main concepts: mobile devices, social networks and gamification. A marketing campaign is modelled as a social workflow that consists of a sequence of game-based activities. These activities typically involve different stores within an indoor mall and must be successfully completed by customers interested on getting a prize and/or a discount on a product. Some of the proposed activities are based on customers' location (to make a photo in front of a store window, to pick up a coupon in a specific store or to go to an advertising short-event, for instance). Therefore, the proposed system requires to accurately track and determine customers' location in indoor environments. Outdoor localization technologies are not suitable indoors because of the Non-Line-of-Sight (NLOS) effect. To solve this problem, WiFi-based methods are a common choice to provide indoor localization due to its many advantages: there are WiFi access points in any indoor mall, measuring WiFi signal is free of charge and almost every customer device (mobile phones, tablets and laptops) has a WiFi interface.

WiFi-based fingerprinting localization methods [6][4] require collecting WiFi measurements at different positions covering completely the target area. The cost in time and effort to collect the required measurements in wide areas, such as an indoor mall, is too high and, besides, these measurements must be periodically updated. To solve this problem, we designed an algorithm to estimate the WiFi signal at positions not site-surveyed during the training stage [3]. This method reduces the effort of collecting signal measurements and it has proved to be really useful to locate customers in this type of environments. Nevertheless, the computational cost of the method is too high to be used in wide areas, as is the case of indoor malls. Therefore, the location of the device can not be estimated in real time if the localization process is computed over a mobile device or a conventional computer. To be able to provide real-time positioning, the localization process has to be performed within a few seconds. To achieve this objective, in this paper we propose a new and efficient version of the location method based on the use of parallel programming techniques and the cloud computing paradigm. The functionality of the method is published by two Web-accessible services and deployed in the Amazon-EC2 Cloud environment. Different cloud-based deployments are evaluated to analyse the performance improvements of the proposal with respect to the original method.

The rest of the paper is organized as follows. The proposed localization method and the computational cost is described in Section 2. Section 3 and Section 4 detail the parallel service for user's location and how this service has been deployed in Amazon EC2, respectively. Finally, Section 5 presents the conclusions and future work.

2 A WiFi-based location method

Figure 1 depicts the proposed method. As it is shown, it consists of two phases. First, a map of signal intensity is computed for each WiFi access point (AP). These maps are represented by large-sized matrices calculated from a set of training measurements. The proposed method uses Support Vector Regression (SVR) [2] to extrapolate the expected RSS (Received Signal Strength) at positions with no available data [3]. Second, the customer's location can be determined with a high degree of precision combining the computed maps and the RSS values recorded by the user's mobile device.

The main objective of this process is to obtain a high resolution localization system without the need of collecting training samples at a high number of positions. This way, the effort and time required to provide the localization system in wide environments can be reduced while increasing the localization resolution. To achieve this objective, during the training stage the training samples collected at discrete positions of the environment are used to infer the expected RSS at positions with no available measurements by using SVR. As a result, the localization system will be composed of a set of continuous reference surfaces that will contain the expected RSS at each coordinate of the environment. Each

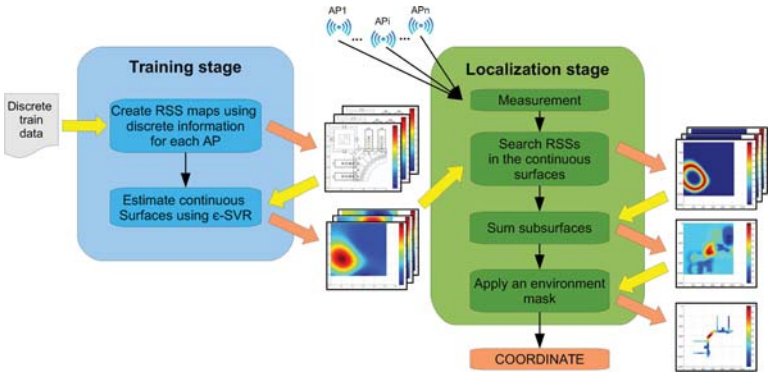

Fig. 1. General architecture of the system [3]

reference surface will correspond with the estimated RSS from a different AP (Access Point).

After that, during the localization stage, a new sample collected at an unknown position will be used to estimate the real location of the user's device. The RSS from each AP will be searched in the corresponding reference surface and a new sub-surface will be created by means of the assignation of scores to the coordinates of the environment depending on how similar is the collected RSS with the stored one. This way, the coordinate will obtain the maximum score (equal to one) if the RSS exactly matches the value stored in the corresponding surface and will be reduced accordingly to the difference between the measured and the stored values.

Once the sub-surfaces for all the APs are computed, they are summed up to obtain a resulting surface containing higher scores as the coordinates are more probable to be the real location of the device. However, as the complete surface could contain non-reachable areas by the users (e.g. storage rooms at shops), a mask is applied in order to remove the non-reachable areas. Therefore, the most probable location of the user's device will be the one with the highest score in the masked resulting surface.

It is important to highlight that the time required to process the continuous reference surfaces and to obtain the device location during the localization stage will depend on two variables chosen during the design of the localization system: the number of the APs and the resolution of the system.

In this paper, the experimental environment has been set up in a medium-size indoor scenario (around $2500m^2$). Although the use of cells with 15cm-side was proved to be the most effective [3], the environment has been divided into 1cm-side cells to evaluate the performance of the proposed system as is were tested in a bigger environment (equivalent to a $562000m^2$ scenario divided in 15cm cells). All the detected APs (a total of 100) have been used to perform the localization. As a consequence, 100 surfaces composed of 25 millions of cells will be used by the localization method. With this configuration, the execution times

are 44.4 hours to compute all maps during the training stage and 325 seconds per location request. The experiments were conducted using a server with 8 Intel i7-4790K CPUs at 4.00GHz, a local SSD disk and 32GB of RAM.

These times must be improved to offer an efficient response from the point of view of BAI4SOW system, especially during the localization stage, as the solution must be able to locate hundreds of customers that are simultaneously participating in a marketing campaign. The proposed solution to solve this problem will be described in the following sections.

3 A parallel service for WiFi-based users location

Two issues must be addressed to integrate the WiFi-based location method into the BAI4SOW system. On the one hand, the execution time of algorithms must be reduced to locate efficiently customers in a medium-size indoor mall. We propose the design and programming of a new parallel version of these algorithms. This new version is based on a master-worker architecture that can be executed in a concurrent or distributed execution environment. And, on the other hand, it is necessary that these new algorithms are deployed in high-performance computing resources while their functionality is easily accessible from BAI4SOW applications. For these reasons, we have decided to integrate them into RESTful Web services that will be executed in the *Amazon cloud* infrastructure.

In the following sections we detail the solutions proposed to overcome the two introduced issues.

3.1 A parallelization strategy based on the master-slave architecture

In software engineering, a master-worker architecture is a high-level design pattern that facilitates the parallel execution of applications composed by a set of independent tasks. The pattern consists of two class of processes: a master and a pool of workers. The former is responsible of assigning tasks to workers and guaranteeing that all of them are correctly completed; whereas the workers simply execute the assigned tasks by returning the corresponding results to the master. This architectural model is highly scalable by increasing (or decreasing) the size of pool according to the execution requirements. It also facilitates the deployment of the solution in a distributed computing environment, such as in a cloud infrastructure, for instance.

The training and location algorithms can be programmed as a master-worker system. On the one hand, the process of creating a continuous reference surface is an independent task computed from RSS measures. Therefore, the generation of the different surfaces can be parallelized in order to accelerate the original solution. On the other hand, the location algorithm requires checking the sample of the customer's unknown position against to each of the reference surfaces. These checks are independent and, therefore, can be also programmed as parallel tasks.

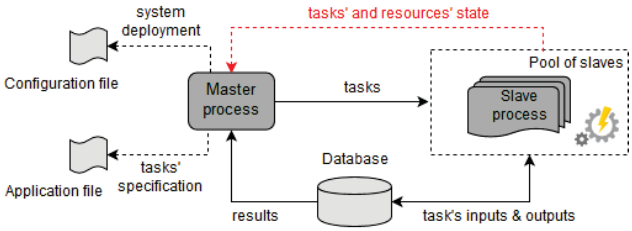

Fig. 2. An abstraction of the implemented master-worker architecture

The parallelization of these algorithms has consisted of two stages. Firstly, we have implemented a generic master-worker architecture that can be easily customized and reused for solving different problems. The architecture was developed using the Java programming language. Figure 2 shows an abstraction of the proposed architecture. The configuration file determines the number of workers and the IP addresses of the computing instances where they will be executed. On the other hand, the application file describes the set of tasks to be completed (task's identification, input parameters and output results, mainly). The master node uses the configuration data for deploying and managing the workers. Then, it assigns tasks to the workers so that they are always busy. Besides, the task's and workers' state are monitored in order to guarantee the fault tolerance of the system. On the other hand, the data involved in solving the problem are stored into a shared database. This reduces the data transfers among the processes.

The second stage of the parallelization consists of dividing the original algorithms in a set of independent tasks than can be executed in parallel. Loop-level parallelization techniques has been applied to code the new version of both algorithms. Besides, all the *MATLAB* code used to create and process the reference surfaces has been converted to *Octave* code. This decision is based on economic criteria: running *MATLAB* code in cloud instances represents an additional cost because of the licences required.

3.2 Parallel algorithms as RESTful services executed on the cloud

Once the algorithms have been parallelized, the goal is to make them accessible to BAI4SOW applications. We decided to integrate them into two RESTful Web services that were published as part of the BAI4SOW infrastructure. This approach allows any application connected to the network to make use of them.

Figure 3 shows the cloud-based implementation of the system. The *Surface Creation Service* (SCS) provides operations to submit training measures and compute the corresponding continuous reference surfaces. The master-worker version of the corresponding algorithm has been integrated into the logic of the service and it has been deployed into the *Amazon EC2 infrastructure*. The flexibility of the master-worker architecture allows us to deploy the service over a concurrent environment (using an only virtual instance that has a high number

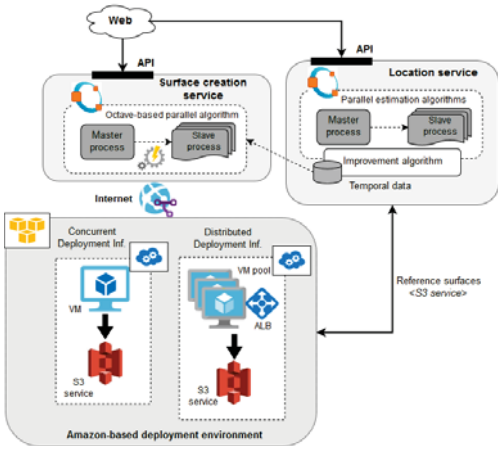

Fig. 3. Cloud-based architectural solution

of CPU cores) or a distributed environment (using a pool of virtual instances). In both cases the data are stored into the *Amazon S3 service*.

On the other hand, the *Location Service* (LS) offers functionality to estimate users' location. RSS samples measured by the user's mobile are aggregated and sent to the service as input parameters. These are processed by the master-worker version of the location algorithm which checks them against the reference surfaces previously computed (therefore, the *Amazon S3* storage is shared by both services). Finally, the service can be also deployed over a concurrent or distributed environment.

Because the services are deployed in the cloud it is necessary to select the computing resources to be provisioned according to the BAI4SOW applications' requirements. The execution times and costs are briefly analysed and discussed in the following section.

4 Time and cost analysis for deploying the services in the Amazon EC2

In this evaluation an user has freely walked in the environment described in Section 2. During the walk, the user's mobile has measured 300 RSS values in different positions and a location request has been sent for each of these values to determine her/his location. Previously, 100 reference surfaces with an accuracy of 1 centimetre were computed to be used by the location method.

In this scenario we are interested in configuring different service deployments in the *Amazon EC2 cloud* and comparing their execution times and costs. Some decisions related to the provisioning of resources are based on our experience in the deployment of cloud-based systems [1]. The goal is to evaluate the time and cost of creating the surfaces ($Phase_1$), responding all the location requests

($Phase_2$), and completing the experiment for each of the proposed deployments. In the case of $Phase_2$, the time per request can be calculated by dividing *Time* $Phase_2$ by 300. We have solved the experiment executing the sequential version of algorithms in a server with 8 *Intel* cores i7-4790K CPUs at 4.00GHz and 32GB of RAM. The overall execution time was 44.4 hours, and the experiment had no costs.

Table 1. Time and cost evaluation

EC2 Instance	**Cores**	**Memory (Gb)**	**Cost/hour (€)**	**Time Phase$_1$ (min)**	**Time Phase$_2$ (min)**	**Total time (h)**	**Total cost (€)**
Sequential	8	32	-	1085.2	1579.1	44.4	-
Concurrent deployments							
1, r4.xlarge	4	13.5	0.24	-	-	-	-
1, r4.2xlarge	8	27	0.48	165.9	640	13.4	8.17
1, r4.4xlarge	16	53	0.95	166.2	326	8.2	10
1, r4.8xlarge	32	99	1.9	85.8	163.5	4.2	10.95
Distributed deployments							
3, r4.8xlarge	96	99	5.7	29	63.4	1.5	12.85
6, r4.8xlarge	192	99	11.4	15,9	31.9	0.8	12.85

Table 1 shows the execution times of the parallel algorithms by considering different deployments in the *Amazon EC2 infrastructure*. First, different models of *R4 virtual instances* have been hired following an on-demand schema in order to evaluate a concurrent deployment. These instances are optimized for memory-intensive applications, and they can offer a good performance for this type of processing problems. The number of cores and the memory of these instances are specified in the second and third columns of Table 1, respectively. As it is shown, the *r4.xlarge* instance suffered memory problems and the experiment was not successfully completed. The other three *R4 instance* models execute correctly the experiment and present a linear speedup according to the number of available cores. The best execution time was achieved by the *r4.8xlarge* instance (4.2 hours) which improves an order of magnitude the time of the sequential version (44.4 hours). On the other hand, the cost of experiments (column *Total cost*) considers the costs of computing resources and of storing the data into the *Amazon S3 service*. The former is calculated multiplying the cost per hour of the selected instances (column *Cost/hour*) by the total execution time (column *Total time*); whereas, the second is the same in all the deployments (the size of involved data is around 66Gb and the storage price is 0.022 €/Gb for the first 50TB/month). As a conclusion, the cost of these instances is proportional to the number of cores and, therefore, the total cost of experiment is very similar for all the concurrent deployments.

On the other hand, we have provisioned and configured two different distributed environments as well. These environments are composed by a pool of three and six *r4.8xlarge* instances, respectively. Let us to remark that the workers are now distributed between the cores of the different instances. Therefore, a maximum of 95/191 workers may be running simultaneously on the provisioned instances (the other core is occupied by the master). The total execution time

continues to improve in proportion to the available cores completing the experiment in less than 1 hour with a pool of six instances. These results show that the overhead of the parallelization has not a significant influence in the times. On the other hand, the total costs slightly increased compared to the costs of concurrent deployments, but this is due to the fact that there is an excess in the computing capacity hired (in the first distributed deployment 96 cores are not used during 30 minutes, for instance).

5 Conclusions and future work

In this paper we have presented a parallel and service-oriented system to improve the execution time of a WiFi-based location method. The system is based on the master-worker architecture and has been tested and deployed in the cloud. The experimental validation has shown a significant improvement in execution time with respect to the sequential solution.

As future work, we are interested in validating the system in a real indoor mall (with hundreds of mobile users participating in a marketing campaign) and in considering new deployments based on the mobile-edge computing paradigm to reduce the communication latency between mobile users and services.

Acknowledgment

This work has been partially supported by the Ctedra de Ingeniera Avanzada Escribano of the UAH (Catedra2017-005), the TIN2017-84796-C2-2-R project, granted by the Spanish Ministry of Economy, Industry and Competitiveness, and the JIUZ-2018-TEC-04 project, granted by the Ibercaja Foundation (UZ).

References

1. Álvarez, P., Hernández, S., Fabra, J., Ezpeleta, J.: Cost-driven provisioning and execution of a computing-intensive service on the Amazon EC2. The Computer Journal **61**(9), 1407–1421 (02 2018). https://doi.org/10.1093/comjnl/bxy006
2. Drucker, H., Burges, C.J.C., Kaufman, L., Smola, A., Vapnik, V.: Support vector regression machines. Advances in Neural Information Processing Systems **9**(9), 155–161 (1997)
3. Hernández, N., Ocaña, M., Alonso, J.M., Kim, E.: Continuous space estimation: Increasing wifi-based indoor localization resolution without increasing the site-survey effort. Sensors **17**(1), 147 (2017)
4. Kim, J., Han, D.: Passive WiFi fingerprinting method. In: 2018 International Conference on Indoor Positioning and Indoor Navigation (IPIN). pp. 1–8 (2018)
5. Lama, M., Álvarez, P., Ocaña, M., Mucientes, M., Ezpeleta, J., Garrido, M.Á.: Análisis inteligente de flujos de trabajo sociales. In: Jornadas de Ciencia e Ingeniera de Servicios (JCIS). Sistedes (2016)
6. Li, W., Wei, D., Yuan, H., Ouyang, G.: A novel method of WiFi fingerprint positioning using spatial multi-points matching. In: Proceedings of the 2016 International Conference on Indoor Positioning and Indoor Navigation. pp. 1–8 (2016)