Extending Queuing Networks to Assess Mobile CrowdSensing Application Performance

Riccardo Pinciroli
DEIB, Politecnico di Milano
Via Ponzio 34/5, 20133 Milano, Italy.
riccardo.pinciroli@polimi.it

Salvatore Distefano
Università di Messina
Contrada di Dio, S. Agata, 98166 Messina, Italy.
Kazan Federal University
35 Kremlyovskaya street, 420008 Kazan, Russia.
sdistefano@{unime.it, kpfu.ru}

ABSTRACT

The widespread and pervasive adoption of smart devices is boosting Internet of Things and contribution-based paradigms. In particular, Mobile Crowdsensing (MCS), due to its big potential of sharing and collecting large population of contributors-devices, is acquiring interest. Devices such as smartphones and smart boards are equipped with different sensors and actuators able to probe data about the physical environment. In a typical MCS scenario, data produced by sensors are sent to the remote server, where they are collected and processed by the applications. To exploit the MCS paradigm in large-scale business contexts the quality of service of MCS applications must be monitored and guaranteed. Therefore, techniques and tools able to represent and evaluate MCS system quality attributes such as performance and energy consumption are required. However, modeling MCS system is quite challenging since not only the number of users but also the number of contributors may vary. In this paper, we propose to adopt queuing networks, a well-known formalism able to deal with large number of requests, to address this issue. In particular we introduce and implement a new policy allowing the number of server to be variable. The proposed model is then adopted in the evaluation of an example, providing interesting insights on contribution, provisioning and usage impacts in terms of some performance and energy consumption metrics.

Keywords

Mobile Crowdsensing; queuing networks; performance; energy consumption.

1. INTRODUCTION

Mobile CrowdSensing (MCS) is a quite new paradigm lying at the intersection of Internet of things (IoT) and volunteer-/crowd-based computing [9]. Its success, confirmed by more and more applications adopting the MCS paradigm [4, 7, 17], is due to the capability of gathering and managing a large population of nodes, mainly providing sensing functionalities, while dealing with scalability issues. Consequently, both academic and business communities are investing resources and efforts to both develop appropriate middleware mechanisms [18, 21] and investigate on a potential commercial exploitation for MCS.

This calls for a refactoring of MCS, rethinking main concepts and ideas of this paradigm to enlarge its scope towards business contexts where quality of service (QoS) requirements are specified through a service level agreement (SLA). Tools supporting design and operation of MCS services, such as those related to QoS modeling and evaluation, monitoring, prediction and enforcement are therefore strongly required. The main challenge to deal with in QoS modeling and evaluation is related to the *churning* management. Contributors may join and leave the system randomly and unpredictably, at their needs and wills, entering or exiting the region of interest, according to their device battery life.

This poses an interesting challenge to the QoS/performance community: how to deal with a system where not only end-users, jobs or requests, but also contributors, nodes or servers may vary in time? A proper MCS system model has to adopt stochastic processes to properly represent arrival and service processes. Furthermore, appropriate memory management mechanisms should be implemented, since the stochastic quantities characterizing these processes are usually generally distributed. In MCS systems, service requests are usually split in simple tasks to be performed by contributor nodes. Due to the churning, these tasks could be interrupted before termination, and thus rescheduled. Some MCS services implement checkpointing policies to limit the impact of this phenomenon on QoS and performance. This way, after rescheduling the task restarts from the last checkpoint condition, thus requiring adequate mechanisms for dealing with memory issues in modeling and evaluation.

Existing solutions partially address these issues, mainly through state space based models [6, 5, 12, 11] and simulation [16]. An open problem in literature is related to complexity, in particular for state space based models where state space may *"explode"*. Some of them allow to represent contribution dynamics and churning, mainly neglecting the memory problem, except [6]. None of them take into account checkpointing policies and related issues.

Thus, the main contribution of this paper is to provide a technique addressing all such issues altogether. It is based on queueing networks (QNs), able to deal with open and closed workloads, also taking into account queueing phenomena. To cope with contribution dynamics, a new policy allowing variable number of servers is introduced and a specific model is defined for MCS system. Checkpointing strategies and related memory mechanisms are implemented, thus providing a framework for the assessment of actual MCS services that could be applied in real applications. Furthermore, to the best of our knowledge, this is the first attempt in evaluating energy consumption metrics, providing information from three different perspectives, i.e., contributors, service provider and end users.

This paper is therefore organized as follows. Section 2 describes the MCS scenario also providing an overview of related work. The QN model is then detailed in Section 3, while Section 4 presents the results obtained by evaluating an MCS example. Finally, conlusions and future work are discussed in Section 5.

2. PROBLEM OVERVIEW AND RELATED WORK

2.1 Scenario

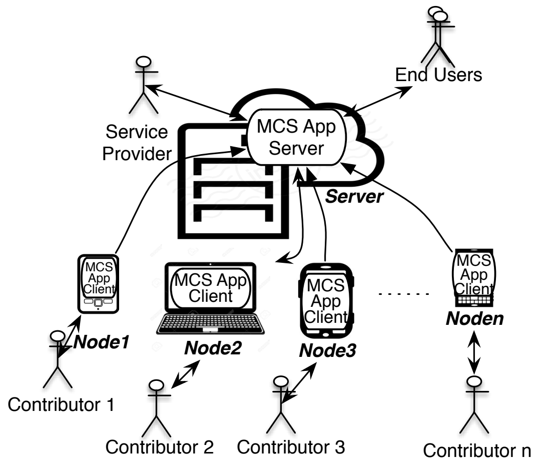


Figure 1: MCS application scenario.

MCS [9] is a kind of volunteer computing paradigm where contributors share their sensors and data, usually provided by their personal devices (smartphones, tablets, PDAs, laptops, etc.), with specific applications gathering and aggregating all the collected data for further processing. In this scenario, depicted in Figure 1, three different stakeholders can be identified:

- *contributors* are the owners of the devices shared to build up the sensing infrastructure environment;
- the *service provider* manages the whole system and service by gathering, managing and processing data and results from contributors;
- *end users* use the MCS application to access the services and the information it provides.

An MCS application, from a high level perspective, is composed of two main phases implemented by the elements showed in Figure 1. Firstly, through the MCS application client deployed on each involved node (usually after it has been explicitly downloaded and installed by the contributor on the device), data are produced, (pre-)processed and sent to the server. Then, MCS application server collects, aggregates, further processes (if required) and stores the data, providing the information and results to the end users.

Each node can elaborate zero, one or more *tasks* whose results are collected for further processing by the application server. The application client can also be intelligent and predict the leave of a contributor (e.g., in case of low battery). In that case, it sends partially processed data, (i.e., a checkpoint) to restart future elaboration on other available nodes.

The MCS paradigm is already adopted in several tools such as OpenStreetmap [10], Waze [20], Nericell [17], as an effective solution for problems related to mobility [22, 7, 4], traffic monitoring [20, 17], public safety [1], smart cities [2] environment and pollution monitoring [22], emergency and crowd management [15], to mention but a few. But most of the potential of MCS is still untapped due to the limits of contribution approach, often unreliable and unpredictable, implying high uncertainty in the service fruition. Indeed, in MCS applications, contributors join and leave the system randomly, often without even completing their assigned tasks. Thus, focusing on the MCS pattern rather than on a specific application, we have to skip the application functional properties while investigating the behaviour of the overall MCS system and its impact on the MCS application. Indeed, the overall MCS system infrastructure is not able to change the behavior of the application from inside, i.e., to modify the application functionalities and functional properties, while could strongly affect the non functional properties of the application. In other words, to properly evaluate the suitability of the MCS pattern/paradigm we have to investigate how it influences the MCS application behavior and, since it cannot affect the application functionalities, proper metrics to assess the impact are those related to non functional properties. For example, reliability, availability and performance of an MCS applications strongly depends on the number of available contributors.

In particular, several metrics of interest can be identified from the different perspectives in Table 1. A contributor is mainly interested in knowing the impact of contribution on his/her device or node, in terms of computing resource utilization (CPU, memory, storage) and battery charge. Instead, service provider is mainly interested in managing the resources, i.e., maximizing their utilization and reducing power consumption, while reducing the time for processing a task and increasing the system throughput. This also affects the response time for processing a request, which is the main metric of interest for the end users.

A model or a modeling technique can provide a valid support to the design, deployment and assessment of the MCS applications. However, several aspects must be taken into account to properly represent an MCS system. Among them, one of the most challenging is the fluctuation of the number of contributors, that strongly impacts on the MCS underlying infrastructure and on the non functional properties of the application. An MCS system representation should take into consideration also the contribution dynamics, in order to stochastically characterize both arrival (join) and departure (leave) time of the contributors

Perspective Metric	***Contributor***	***Service Provider***	***End User***
Resource Utilization	Node	Server	-
Energy Consumption	Battery	Server	-
Response Time	-	Task	Service
Throughput	-	Task	

Table 1: MCS system metrics of interests

by specific probability distributions. Stochastic characterization by proper distribution is also needed for service time of both the nodes (clients) and the servers, and for arrival rate of the jobs. In general such distributions are not always Markovian, since their stochastic behaviors not necessarily observes the memoryless property of exponential distribution. Furthermore, the model has also to deal with the complexity of an MCS system, for example, taking into account queueing effects as well as checkpointing policies.

2.2 Related work

Several approaches to the MCS challenges can be found in literature, since it is an umbrella term enclosing different paradigms such as opportunistic, participatory and social sensing [9]. Modeling techniques are required to understand the MCS application behaviours when this application operate under different scenarios. As an example, simulation is proposed for the evaluation of MCS system performance in [5], while Karaliopoulos et al. [13] use stochastic model to face the problem of user mobility, selecting users for crowdsensing campaign through an optimization problem. Stochastic models are also used to investigate how to maximize utility [12] and profit [11] in MCS applications. In both cases, Lyapunov optimization is used to find the best solution for the considered problem. Liu et al. [14] adopt a continuous time Markov chain (CTMC) to study the data collected by sensing vehicles, focusing on the spatial distribution of the vehicles. In [6] and [16] MCS applications are modeled to analyze their QoS. In particular, the authors adopt an extended version of Stochastic Petri net formalism to consider the elements that can affect the performance of an MCS system (i.e., contributors and workload fluctuations). In [16] they also propose an MCS as-a-Service paradigm to address actual business challenges. The main problem of all such techniques lies on complexity. State space based models have intrinsic limitations in dealing with such an issue due to the well know state space explosion problem. Also simulation time could be significantly affected by the MCS system complexity, even if in a less critical way than state space models.

In this paper we recur to queueing theory to model MCS user and contributor arrival processes, since it allows to represent complex systems with both open and closed workloads. The only problem is that the contributors have to be considered as servers in the corresponding queueing network (QN) model, thus implying to introduce a new multiserver policy able to cope with contributor-server fluctuations. Furthermore, in order to provide quantitative results from different perspectives (contributors, service provider and end users), we also have to evaluate energy related metrics on nodes and servers. Conti et al. [5] identify this aspect as one of the key challenges to motivate participants in contributing. Indeed, since the battery capacity of these devices is often very limited, MCS applications should have a low impact on the battery depletion, letting users be almost unaware of their contributions to MCS.

3. A QUEUING NETWORK MODEL

Based on the scenario of Figure 1, two different layers have been identified for the proposed QN model: the contribution and the processing subsystems (Figure 2). Contributors are modeled in the contribution QN, whereas end user's requests are represented by jobs in the processing QN. These two networks are strictly related to model the stochastic behaviour of the MCS system. In fact, when a contributor joins the contribution QN, a server in the processing network is turned on and can serve incoming requests. The server is turned off when the contributor associated with that server leaves the contribution QN. Note that a server is related to a specific contributor until it leaves the system. Once a contributor leaves the contribution QN, the server in the processing network may be turned on by another contributor. In other words, at most only one active contributor at time can be associated with a server, while any contributor can be associated with a server when it is idle. The two sub-networks are detailed in next Sections.

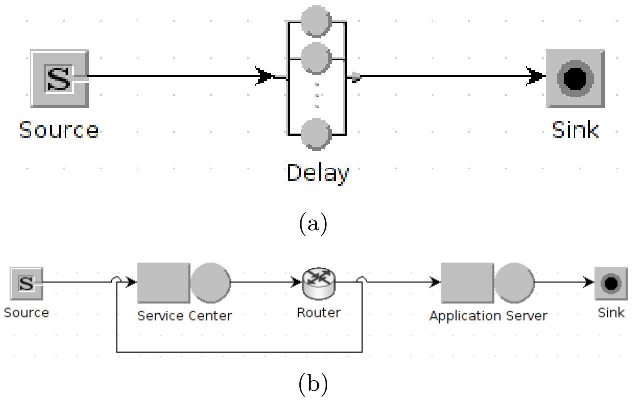


Figure 2: The MCS application model with the contribution (a) and processing (b) queueing networks.

3.1 The contribution QN

The contribution QN represents contribution dynamics. Contributors join and leave the MCS system randomly, spending in the system their contribution time. The contribution QN is shown in Figure 2(a).

Every job in the `Delay` station (i.e., a $G/G/\infty$ queue) models a contributor that is sharing its resources. The arrival rate to `Delay` and the time a job spends in it may be characterized by any general distribution. They represent the arrivals of the contributors in the system and their contribution times (i.e., the time they are available to serve the end user's requests), respectively. As discussed above, the `Delay` station of this network is connected to the `Service Center` one in the processing QN. The number of servers in the processing QN is equal to the number of jobs in the `Delay` station of contribution QN.

3.2 The processing QN

The processing QN represents the requests incoming from MCS end users, which are served by the contributors' nodes. Figure 2(b) shows the corresponding QN. An end user request is decomposed into n task requests which are served by contributor nodes. The task request arrival process is characterized by a general distribution and does not depend on the contribution QN. It is modeled by the `Source` station. Also the service time of the `Application Server` station does not depend on the contribution QN and is generally distributed (i.e., a $G/G/1$ queue). This station models the commit of a request once it has been completed. On the other hand, the `Service Center` station depends on the contribution QN. It is a $G/G/x$ queue, with generally distributed arrival and service times and variable number of servers $x \in [0, \infty)$. It is important to remark that, the number of servers x follows the number of contributors that are in the system and therefore is variable depending on the contribution dynamics.

A task request is served by only one server at a time, but it may be served by different servers till it is completely processed. Indeed, a request exits from the `Service Center` whether it has been fully served or the i-th server that is serving the task is turned off, meaning that the contributor related to that server leaves the system. In the former case, the completed task request is sent to the `Application Server` by the `Router` and it leaves the system after the end-user request is processed by the server, i.e., once n task requests are processed. In the latter case, when the task request is not completed, it is sent by the `Router` back to the `Service Center`, waiting to be fully processed by a free server.

Priority policies may be applied to the `Service Center` in order to prioritize dropped requests, since the number of available servers may be lower than the number of requests that need to be served. This way, a task request that has been interrupted can be rescheduled for processing with a higher priority. Indeed only x requests can be served at the same time by the `Service Center`.

3.3 Measurements

Once the model has been specified, we have to discuss how to evaluate its performance. Referring to Table 1, an end user is interested in the average *Push time*, the time needed by the system to process an end user request, i.e. the processing of n task requests and their further elaboration by the application server. It can be specified as:

$$PushTime = \frac{\sum_{i=0}^{\lfloor C_T/n \rfloor} CompletionTime_i}{\lfloor C_T/n \rfloor} \tag{1}$$

where C_T is the number of completed task requests, n is the number of committed requests that the system collects before returning some results and $CompletionTime_i$ is the time at which the i-th set of n requests have been committed.

The service provider may be interested in the average system response time R_P and the throughput X_P. The former is computed observing each task request and evaluating how long it spends in the system. The latter as $X_P = C_T/T$, where T is the observation time.

Finally, a contributor is interested in knowing the impact of contribution on his/her devices, quantified by the utilization U_C and the battery energy consumed by the MCS app during contribution. To this purpose, we evaluate U_C of each node as:

$$U_C = \frac{\sum_{i=0}^{x_{max}} ActivityTime_i}{\sum_{i=0}^{x_{max}} OnTime_i} \tag{2}$$

where $ActivityTime_i$ is the time spent by the server i in the `Service Center` serving a request, $OnTime_i$ is the total time that server i is available for the MCS application, either busy to serve a task request or idle, waiting for a task request to serve, and x_{max} is the maximum number of server in the `Service Center`. Assuming that the device is mainly used for computations, we evaluate the power consumption P_C of each node as shown in [8]:

$$P_C(U_C) = P_C^{idle} + (P_C^{busy} - P_C^{idle})U_C \tag{3}$$

where P_C^{idle} is the power consumption when the device is idle, P_C^{busy} is the power consumption of the device when it is fully utilized and U_C is the utilization of the device as obtained by eq. (2). Once the power consumption has been computed, the average energy consumption E_C is derived as:

$$E_C = P_C \cdot S_C \tag{4}$$

where S_C is the contributor service time, i.e., the overall time the contributor spends in the system. We estimate the percentage of battery depleted by the contributor for the contribution to the MCS system as:

$$Battery\ consumption = \frac{E_C}{W_C} \tag{5}$$

where W_C is the battery capacity of the node.

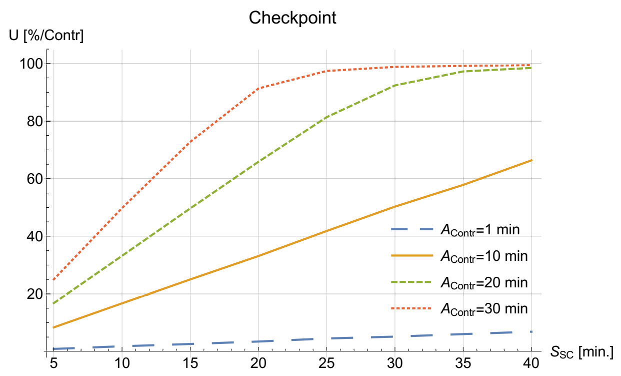
4. A NUMERICAL EXAMPLE

In order to explain the proposed technique and the model thus obtained, in this Section we discuss an example on an MCS system, taking parameters from existing literature when available.

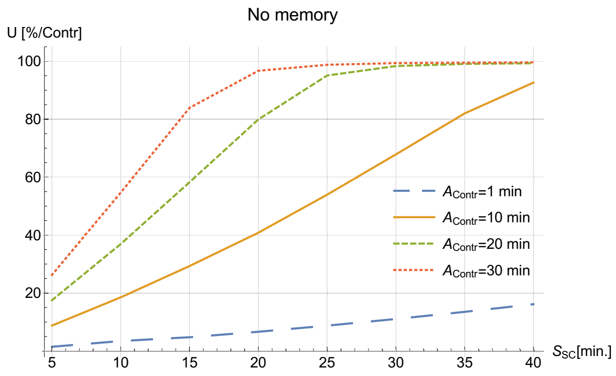
4.1 Description

In the MCS system described above, end users submit requests to the MCS application. Each request is split in n simpler tasks, that are performed by the contributor nodes. Once the n tasks have been processed, the provider application server gathers the results and provides the overall elaboration outcomes to the end users. As discussed in Section 2.1, memory strategies may be taken into account in task request processing, since these may be often interrupted and therefore rescheduled for processing. We therefore evaluate an MCS system with and without memory strategies. In the former case, a checkpoint strategy lets the system keep memory of the task requests processing. In the latter case, the computation of the task requests restarts from scratch every time they are served by a new contributor.

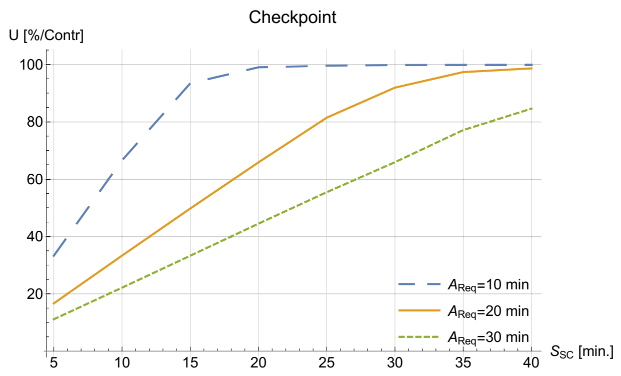
The parameters used in the experiments have been taken from literature when possible. The contributor and request arrivals are exponentially distributed, as well as the service time of the `Delay` and the `Application Server` stations. The service time of each server in the `Service Center` station is characterized by an Erlang distribution with 3 stages, similarly to [6]. The average service time of `Application Server` S_{AS} and `Delay` S_{Contr} is always the same for all the experiments, i.e. 10 seconds and 30 minutes, respectively. The inter-arrival time of the two networks (i.e., A_{Contr} and



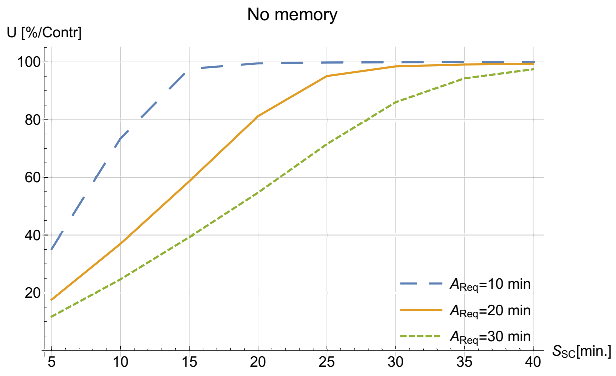
(a)



(b)



(c)



(d)

Figure 3: Average utilization per contributor as a function of service time with fixed request inter-arrival time and either checkpoint (a) or without checkpoint (b), and with fixed contributors inter-arrival time and either checkpoint (c) or without checkpoint (d).

A_{Req} for contribution and processing QN, respectively) and the average service time of server in the `Service Center` S_{SC} have different values in each experiment. More specifically: $A_{Contr} = \{1, 10, 20, 30\}$ minutes; $A_{Req} = \{10, 20, 30\}$ minutes; $S_{SC} = \{5, 10, 15, 20, 25, 30, 35, 40\}$ minutes. When the system is analyzed varying A_{Contr} (or A_{Req}), the parameter A_{Req} (or A_{Contr}) is set to 20 minutes. Each experiment is evaluated for all the values that S_{SC} can assume. Notice that, a more complex request is associated with a larger value of S_{SC}. The system is always observed for 100 hours. The strategy used in the `Service Center` to select the next request to be processed is a FIFO with priority. The priority of a request increases every time the request is sent from the server back to the queue. The job with the largest priority is the first to be served. If two or more jobs have identical priority value, FIFO strategy is adopted. The number of requests that must be committed before the system returns some results to end user is $n = 10$.

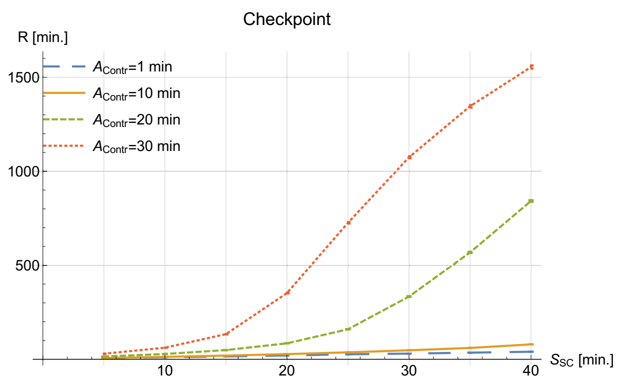
In all the experiments, the idle and the maximum power consumptions and the battery capacity for all the devices are $P_C^{idle} = 0.268W$, $P_C^{busy} = 1W$ and $W_C = 7.77$ Watthour [3], respectively.

4.2 Evaluation

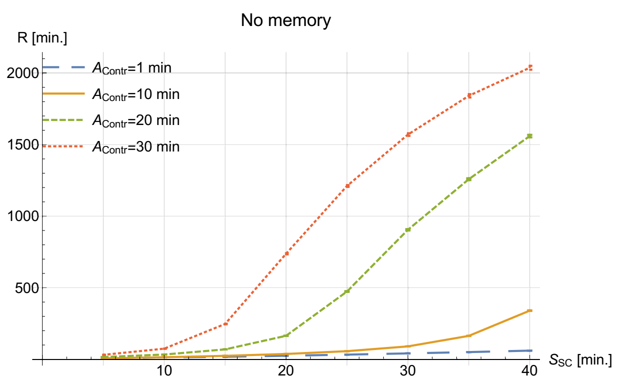
All the experiments have been performed through discrete event simulation using OMNeT++ [19]. It is an extensible, modular, component-based C++ simulation library and framework used to build network simulators. We patched version 4.6 in order to study the model described in Section 3. In particular, we implemented the relation between the contribution and the processing queueing networks and edited the `Job` implementation in order to consider different memory strategies as described in Section 4.1. The results obtained are discussed in the following Sections from the different stakeholder perspectives identified for the MCS system. Each simulation performed 500 runs and all the measurements have been computed with 99% confidence intervals.

4.2.1 Contributors

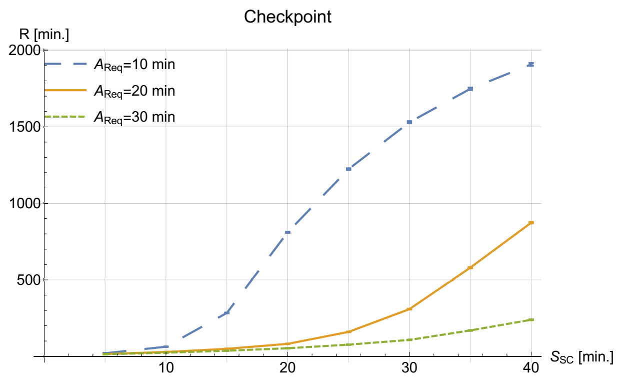
Contributors are mainly interested in quantifying the impact of the contribution on their resources, mainly evaluated by the utilization and the battery consumption of their nodes. We analyze these quantities through the measurements specified by eq. (2-5). Results are shown in Figure 3, where the average utilization per contributor as a function of the average service time of `Service Center` is plotted. In Figures 3(a) and 3(b) different inter-arrival time of the contributors are considered whit requests arrival time of 20 minutes, comparing the two strategies (i.e., checkpoint and no memory, respectively). As expected, the smaller the contribution inter-arrival time, the lower the node utilization since the total number of contributors (i.e., $N_C = S_C/A_C$) increases when the inter-arrival time A_C decreases, being the average contribution time constant, i.e., $S_C = 30$ minutes. Furthermore, if checkpoint memory strategy is used,



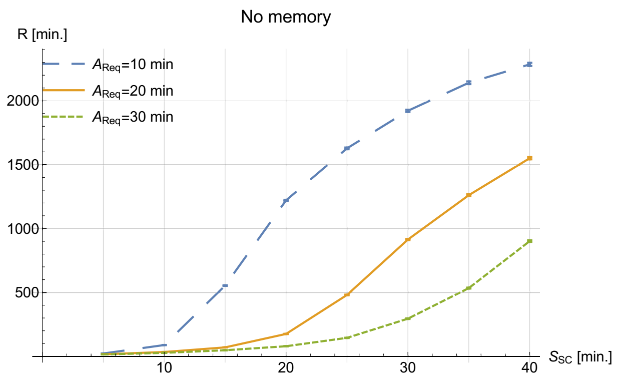
(a)



(b)



(c)



(d)

Figure 4: Average system response time for task requests, as a function of service time with fixed request inter-arrival time and either checkpoint (a) or without checkpoint (b), and with fixed contributors inter-arrival time and either checkpoint (c) or without checkpoint (d).

the utilization of each device is lower than the service without memory, since contributors have to perform a greater amount of work when no checkpoint is adopted. In Figures 3(c) and 3(d) the inter-arrival time of contributors is set to 20 minutes, whereas arrival time of the requests varies. Also in this case the contributors utilization is larger when requests are served with no memory. A larger utilization is also observed when the inter-arrival time of the requests is smaller. Indeed, in that case a greater amount of requests is in the system.

The utilization analysis is useful to study the energy consumption. The power absorption – by which energy consumption is derived – is computed through eq. (3) that is linear with respect to utilization. For this reason, energy consumption curves have the same trend of utilization ones and have not been plotted for the sake of space. However, considering the parameters given in Section 4, the battery depletion for a 30 minutes contribution is never greater than 7%, when $U = 100\%$.

4.2.2 Service provider

The MCS service provider is interested in several metrics that allow to monitor the MCS system. Utilization is an important measure also for the service provider. In this case, it is used to see when the system saturates (i.e., utilization of each contributor is 100%). When it happens, the performance of the system degrades and the requested QoS cannot be satisfied, driving to SLA violations.

System response time is plotted in Figure 4 against service time. Also in this case, different memory strategies have been considered. In Figures 4(a) and 4(b) arrival time of the requests is set to 20 minutes and the one of the contributors varies. In the former the checkpoint memory strategy is used, in the latter no memory strategies are adopted. It can be observed that the smaller is the number of contributors in the system – and consequently the greater is their utilization – the larger is the system response time. This performance index strongly depends on how many contributors are in the system. Indeed, if there are enough contributors, the task requests are served without interruption and they are completed as soon as possible. Instead, when the number of the contributors is too low, each task request may spend several time waiting to be served. This behaviour is worse when no memory about the work done by previous contributors is kept, since a task request must be processed from the beginning every time it enters `Service Center`.

Figures 4(c) and 4(d) show the system response time for the requests when contributors arrival time is 20 minutes and inter-arrival time of requests changes. As in the previous case, similar consideration can be drawn. In particular, the higher the requests arrival time (i.e., the larger is the number of requests concurrently in the system) the higher their response time.

The provider may be also interested in the system throughput, since it is often constrained by SLAs. For the sake of space, system throughput graphs are not shown. A larger number of contributors (i.e., smaller value of contributors arrival time) lets the system provide the largest throughput also when complex requests are in the system. When the requests inter-arrival time is larger, the throughput has its maximum value when the activity is simple, whereas it decreases when the job to be completed is harder. Also in this case the checkpoint memory policy is better than the no memory strategy.

4.2.3 *End user*

The push time is the time elapsed from the submission of the end user's request, until the system provides the results. In our example, the system processes $n = 10$ task requests before returning a value. The push time is plotted in Figure 5 against service time, comparing checkpoint and no memory strategies, for different values of either contributors or requests inter-arrival time. In Figures 5(a) and 5(b) the contributors arrival time varies. The minimum push time value is 200 minutes for each value of A_C, with request inter-arrival time of 20 minutes. The push time increases if either the requests are too complex to solve (i.e., for larger service time values) or the number of available contributors is lower (i.e., for larger A_C values).

Different behavior is observed when A_T changes and A_C is set. In this case, each curve has its own minimum, since different values for requests inter-arrival time are considered. It is worth notice that the larger the number of requests in the system, the earlier the push time starts increasing. Indeed, requests with the same complexity can easily affect the system performance when several other requests are already in the system. In both the considered cases, checkpoint strategy provides the best results.

5. CONCLUSIONS AND FUTURE WORK

MCS applications is attracting more and more interest as a new research field in IoT. In this paper, they are analyzed through queuing network models. In particular, we focused on the QoS of the MCS system, and we analyzed several performance metrics that may be of interest for the different stakeholders (i.e., contributors, system provider and end users). System provider can implement some memory strategies in order to improve performance and optimize battery life. Energy consumption is an important performance metric, since it may stimulate contributors to join the MCS system. Indeed, if energy consumption is too high only few contributors may join the system, affecting all the other performance indices. Through simulation on an MCS system taken as example, it has been shown that a checkpoint strategy lets the system work with better performance.

This paper is only a first step toward a new model for the MCS applications. Even if the results here presented show the importance of this model, we still need to validate it against either real system or a detailed simulation of the MCS services. Another interesting development could be on considering different quality properties and metrics altogether. To this purpose, an optimization problem should be specified, to identify the best configuration for the MCS system. Moreover, it may be of interest to consider different memory strategies and different classes of requests and contributors. In the former case, several policies can be evaluated and compared to the checkpointing one. In the latter case a more detailed model of an MCS system can be provided. For the same reason, also different priority strategies for choosing the request to be processed should be further investigated. Finally, it could also be interesting to study a single server version of `Service Center` as proposed in [6], comparing the results thus obtained against the latter ones.

6. REFERENCES

[1] E. Aubry, T. Silverston, A. Lahmadi, and O. Festor. Crowdout: A mobile crowdsourcing service for road safety in digital cities. In *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2014 IEEE International Conference on*, pages 86–91, March 2014.

[2] G. Cardone, L. Foschini, P. Bellavista, A. Corradi, C. Borcea, M. Talasila, and R. Curtmola. Fostering participaction in smart cities: a geo-social crowdsensing platform. *Communications Magazine, IEEE*, 51(6):112–119, June 2013.

[3] A. Carroll and G. Heiser. An analysis of power consumption in a smartphone. In *USENIX annual technical conference*, volume 14. Boston, MA, 2010.

[4] Y. Chon, N. D. Lane, F. Li, H. Cha, and F. Zhao. Automatically characterizing places with opportunistic crowdsensing using smartphones. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, pages 481–490. ACM, 2012.

[5] M. Conti, C. Boldrini, S. S. Kanhere, E. Mingozzi, E. Pagani, P. M. Ruiz, and M. Younis. From manet to people-centric networking: milestones and open research challenges. *Computer Communications*, 71:1–21, 2015.

[6] S. Distefano, F. Longo, and M. Scarpa. Qos assessment of mobile crowdsensing services. *Journal of Grid computing*, 14, 2016.

[7] S. B. Eisenman, E. Miluzzo, N. D. Lane, R. A. Peterson, G.-S. Ahn, and A. T. Campbell. Bikenet: A mobile sensing system for cyclist experience mapping. *ACM Transactions on Sensor Networks (TOSN)*, 6(1):6, 2009.

[8] X. Fan, W.-D. Weber, and L. A. Barroso. Power provisioning for a warehouse-sized computer. In *ACM SIGARCH Computer Architecture News*, volume 35, pages 13–23. ACM, 2007.

[9] R. K. Ganti, F. Ye, and H. Lei. Mobile crowdsensing: current state and future challenges. *Communications Magazine, IEEE*, 49(11):32–39, 2011.

[10] M. Haklay and P. Weber. Openstreetmap: User-generated street maps. *Pervasive Computing, IEEE*, 7(4):12–18, Oct 2008.

[11] Y. Han and Y. Zhu. Profit-maximizing stochastic control for mobile crowd sensing platforms. In *2014 IEEE 11th International Conference on Mobile Ad Hoc and Sensor Systems*, pages 145–153. IEEE, 2014.

[12] Y. Han, Y. Zhu, and J. Yu. Utility-maximizing data collection in crowd sensing: An optimal scheduling approach. In *Sensing, Communication, and Networking (SECON), 2015 12th Annual IEEE International Conference on*, pages 345–353. IEEE, 2015.

[13] M. Karaliopoulos, O. Telelis, and I. Koutsopoulos. User recruitment for mobile crowdsensing over opportunistic networks. In *2015 IEEE Conference on*

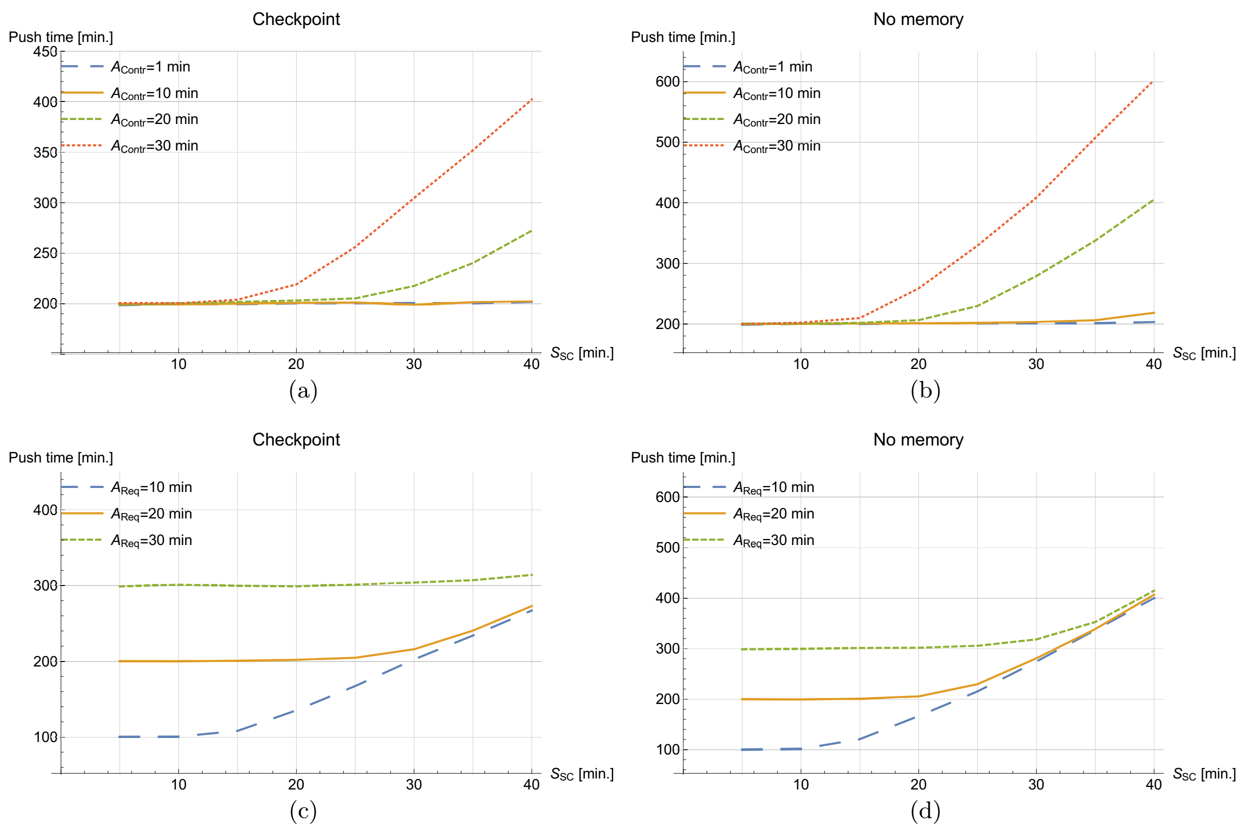


Figure 5: Average push time, as a function of service time with fixed request inter-arrival time and either checkpoint (a) or without checkpoint (b), and with fixed contributors inter-arrival time and either checkpoint (c) or without checkpoint (d).

Computer Communications (INFOCOM), pages 2254–2262. IEEE, 2015.

[14] Y. Liu, J. Niu, and X. Liu. Comprehensive tempo-spatial data collection in crowd sensing using a heterogeneous sensing vehicle selection method. *Personal and Ubiquitous Computing*, 20(3):397–411, 2016.

[15] T. Ludwig, C. Reuter, T. Siebigteroth, and V. Pipek. Crowdmonitor: Mobile crowd sensing for assessing physical and digital activities of citizens during emergencies. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, pages 4083–4092, New York, NY, USA, 2015. ACM.

[16] G. Merlino, S. Arkoulis, S. Distefano, C. Papagianni, A. Puliafito, and S. Papavassiliou. Mobile crowdsensing as a service: a platform for applications on top of sensing clouds. *Future Generation Computer Systems*, 56:623–639, 2016.

[17] P. Mohan, V. N. Padmanabhan, and R. Ramjee. Nericell: rich monitoring of road and traffic conditions using mobile smartphones. In *Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 323–336. ACM, 2008.

[18] M.-R. Ra, B. Liu, T. F. La Porta, and R. Govindan. Medusa: A programming framework for crowd-sensing applications. In *Proceedings of the 10th international conference on Mobile systems, applications, and services*, pages 337–350. ACM, 2012.

[19] A. Varga and R. Hornig. An overview of the omnet++ simulation environment. In *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, page 60. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008.

[20] Waze Mobile. Waze website, 2006.

[21] Y. Xiao, P. Simoens, P. Pillai, K. Ha, and M. Satyanarayanan. Lowering the barriers to large-scale mobile crowdsensing. In *Proceedings of the 14th Workshop on Mobile Computing Systems and Applications*, page 9. ACM, 2013.

[22] X. Yu, W. Zhang, L. Zhang, V. O. Li, J. Yuan, and I. You. Understanding urban dynamics based on pervasive sensing: An experimental study on traffic density and air pollution. *Mathematical and Computer Modelling*, 58(5âĂŞ6):1328 – 1339, 2013. The Measurement of Undesirable Outputs: Models Development and Empirical Analyses and Advances in mobile, ubiquitous and cognitive computing.