On Implementing Social Community Clouds Based on Markov Models

Stavros Souravlas, *Member, IEEE,* and Sofia Anastasiadou

***Abstract*—Social networks reflect, to a wide extent, the real world relationships that allow users to connect and share information. The number of people that interact in social networks keeps increasing and the devices used are equipped with more and more computational capacities. This gives rise to the formulation of social clouds, which refer to resource sharing infrastructures that enable friends to share their resources within the social network. As the modern applications become more and more sophisticated, users should be able to share their own services and computing resources through social networks. This poses many challenges for the design options of a computing system composed of a set of trusted friends. The spotlight turns on the design of a proper trust model which considers the suitability of the trusted users to execute an application's tasks and on the fair distribution of these tasks among these users. Therefore, social networks and their trust-based applications in a distributed environment have seen an increasing attention in the research community. In this regard, we present a social community cloud implementation model, where friendly relationships determine the resource provisioning. The issues of fairness and allocation time are of great importance and they are thoroughly investigated. We use extensive simulations to illustrate that the communities can be employed to construct cloud infrastructures, such that, the shared resources can be utilized in a fair and efficient manner. Our experiments have shown that our model achieves higher allocation rate (percentage of tasks successfully allocated and completed) than competitive models and reduces the average response time and the total execution time. Finally, our work does not over-utilize the resources.**

***Index Terms*—Social Clouds, Social Networks, Cloud Computing, Resource Allocation**

I. Introduction

Cloud computing is a very popular internet-based technology, which provides resource and computing services to users with different needs [1]. Nowadays, the number of people who interact on social networks increases, while these users are equiped with higher computer capabilities. In this regard, there is a good potential of setting up a resource sharing network which enables users to share their resources within the community. This is the concept of a "social cloud". Apart from resource sharing, social clouds are important because they enhance cooperation among multiple users and they provide a means of making available additional capacity to friends and the capability of executing heavier applications. Finally, a community cloud can execute internal tasks (within the community) as well as external tasks (in cooperation with other communities). To make the importance of social clouds more clear, we note that "if only 0.5% of Facebook users provided CPU time on their personal compute resources the potential computational power available would be comparable to a www.top500.org supercomputer" [2]. Community clouds have a number of applications. For example: (1) *Technical forums*, which are a common place for professionals to ask questions, discuss a problem, and get suggestions or solutions for a problem of their interest. All the data is placed in the cloud, for future reference and use, and (2) *Mobile applications:* Many mobile users have difficulties in maintaining the phone memory to accommodate the data gathered from all the applications. The cloud memory is a solution. Also, community clouds have can be used in education and healthcare sectors.

Stavros Souravlas is with the Department of Applied Informatics, University of Macedonia, Thessaloniki, Greece. Also, a Postdoc researcher in the Department of Midwafery, University of Western Macedonia
Email: sourstav@uom.edu.gr, ssouravlas@uowm.gr
Sofia Anastasiadou is with Department of Midwafery, University of Western Macedonia, School of Health Sciences.
Email: sanastasiadou@uowm.gr

Social networking has become more than an everyday need for a huge number of people. Every day, vast data amounts are generated by the famous social networks such as Facebook, Twitter, Instagram, and so on. For example, Facebook alone generates 4 petabytes of data per day. The study of communities has received considerable attention a long time ago and it is one of the most important research topics in social networking. Usually, graphs are used to show the connections among individuals who are related as colleagues, friends and so on. The rapid growth of cloud computing and the extensive use of social networks have given rise to the exploration of new means of interaction between users, in order to benefit from these infrastructures.

Given some information (like sharing of interests such as music, sports or politics), clusters called communities can be detected. The detection of such communities has been widely investigated with a number approaches ranging from data structure to machine learning based approaches [3]. A community can be thought of as a structure of nodes (users) connected by edges. This connection represents the trust relationship. The trust is used as the basis for resource sharing in the community cloud. In a community cloud, a large number of computing devices like personal computers or mobile phones use the available resources to complete any processing, storage, or other type of task. In other words, the community cloud is an infrastructure that extends the computing and storage capabilities by resources contributed by users among a group of friends. The model is similar to a volunteer computing approach, where friends share their resources for some or no gain through the inherited friendship and trust relationships [4].

There are many challenges in the construction of a social cloud: (1) **Technical challenges:** Technically, to enable resource sharing, the social community cloud needs to have the proper mechanisms to handle the non-static IP addresses of the mobile users and ensure of the best quality of service. (2) **Shared policies and protocols:** A community cloud requires the specification of certain policies regarding the building, the operation, and the maintenance of the entire system. These policies include: (i) Access policies which are tailored to the requirements of every interested part, (ii) Resource aggregation and allocation policies, which clearly define how the resources are aggregated, how and when they are allocated and what happens when members disconnect or withdraw, and (iii) Security policies which guarantee security (like protecting the shared resources and the running applications from malicious attacks). (3) **Design of a trust model:** To utilize social structures for resource sharing, first the identification of trusted users is required. This relies on everyday social network data, which is retrieved from each user's everyday transactions within the community. Moreover, an efficient community detection scheme is required in order to define the set of trusted users that will share their resources. Apart from defining these users, their computing capabilities must be known so that the social cloud can be aware if they are suitable to execute tasks of an application. In turn, this necessitates a fast and efficient procedure, which selects the proper users whose resources are going to be shared. (4) **Fairness:** The tasks should be fairly assigned to the selected users. This is important, in order to utilize resources of different capabilities (for example, CPU power, memory, etc) in an efficient manner, so that the execution is as fast as possible. Also, there is another important issue regarding fairness: the allocation of resources for monetary exchange. In this case, the resources are sold to the interested part and in this case, the need for fairness grows even bigger. In such a competing environment, the social cloud users need to assign values to their resources and also to declare which tasks they are willing (or able) to perform. The implementation of economic models in this regard is of high importance, but it is not part of this work.

In this work, we consider the last two challenges (apart from monetary issues) and we show how resources can be shared once communities have been formed. For each user within a community, there is a set of trusted friends which, based on their computing capabilities, are employed by the user to execute a set of tasks. We implement a distributed cloud framework based on community resources and we show how these resources can be allocated fairly to execute the user tasks. This approach has been used in numerous research papers and has a number of advantages: (1) it facilitates the access to a number of services by enhancing their visibility to users, (2) the resource selection process can remain localized (within the borders of a community), (3) it improves the QoS and user management, and (4) the overall system relies on existing trust relationships that have already been formed.

The basic components of a social community cloud are shown in Fig. 1: The *monitor* us a system that keeps all the information regarding the community users, they resources they can donate, the required resources, their preferences, friend and trust information, their availability and current resource allocations. The *trust evaluation and management-TEM* is a control system, which is necessary to guarantee that the resource allocation is reliable and trustworthy and thus, it determines which community users can interact. The *service discovery and matching mechanism* refers to the system that is responsible to aggregate the shared resources and implement a strategy for their proper and fair allocation.

Our proposed resource allocation strategy employs a queuing network model and each node is modeled as a single queue where the assigned tasks are inserted and wait for their execution. A task can enter the queue provided that the corresponding device satisfy the requirements defined by the requester. Such requirements can include, CP (computing power in MIPS), RAM capacity, type and location of device and availability. We divide the time into time slots and perform the allocation during each slot. We can assume that during this short time the network the number of requested tasks remains fixed, that is, our community behaves like a closed network. The main contributions of our work are summarized as follows:

1. We provide a distributed framework for task execution within a community or among different communities. The framework consists of a candidate selection procedure a fair resource allocation policy.
2. The proposed candidate selection policy is linear in terms of time as it is based on a community detection scheme which has logarithmic running time (based on threaded binary tree structures).
3. We provide a fair, envy-free resource allocation policy within the community. This model is based on Markovian system modeling and its computations are proven to be linear.
4. From contributions (2) and (3), it is clear that the model is easy to apply and it is computationally efficient with linear complexity.
5. We conduct extensive simulations under different scenarios to evaluate the proposed resource allocation policy. The results have shown that the proposed scheme outperforms competitive strategies in terms of allocation rate, average response time and total execution time without over-utilizing the resources.

The remaining of this work is organized as follows: In Section 2, we describe the related work. Section 3 defines the problem to be solved. In Section 4 we present our resource allocation policy within the community and we prove that it guarantees fairness and high utilization. Section 5 presents a variety of simulation scenarios. Section 6 concludes this paper and discusses issues that need to be further investigated in the future.

II. Related Work

Volunteer computing is a distributed computing model, in which users donate their computing resources to specific user applications [4]. For a social community cloud, some level of accountability and reliability is required. In some models [5], there is a credit system in which the contributors earn credits when contributing their resources. Also, community users that borrow resources need to spend their credits. The aforementioned limitations can be overcome by employing the concept

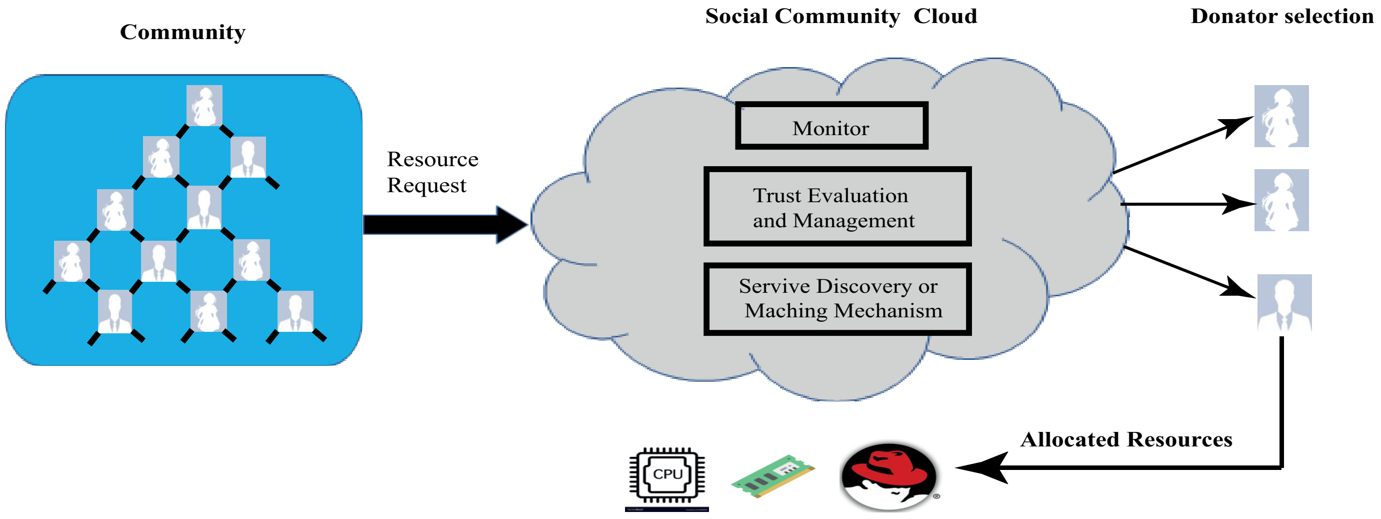

Fig. 1. The system model.

of *friendship* or *trust* among community members. In such a context, the computing services are offered by trustworthy users, taking advantage of pre-existing trust relationships [6].

A lot of papers devoted on social community clouds have focused individually on trust management [7]–[12]. These strategies can be classified in many categories based on the strategy being used [13], [14]. The *objective* TEM strategies are based on the QoS properties of each machine [15], while in *subjective* TEM strategies, the criterion is the individual opinions of the community members [16], [17]. The *context based* models are based on object behavior and use techniques like crediting, in order to evaluate the trust among users. Users that provide good services are highly credited [18]. The *composite* models combine friend opinions and recommendations, social contact and common interests within the community to define trust [9], [10], [19]. The *direct* trust models are based on the direct experience of a user with the neighbors [20], while the *indirect* trust models are based on indirect recommendations and reputation, from other users experience. Usually, both direct and direct factors are taken into account [21], [22] to develop more concise TEM models. The aforementioned trust models can also be characterized as *dynamic*, in the sense that the trust values do change over time, when the community configuration and conditions also change. An exception is [10], which is an example of a *static* model. The main disadvantages of these problems is that the selection of machines to perform the tasks is based merely on trust while their complexity is rather high [13].

A number of papers has been devoted to cooperative strategies (mobile crowdsensing), where users which have been assigned a complex task need to cooperate and communicate in order to accomplish it [23]–[28]. Such tasks may require specific amount of users, with specific machine characteristics. The user selection and the task allocation is an important and challenging problem for crowdsensing and affects the quality of the services offered. The majority of the aforementioned strategies focus on a tradeoff between quality and cost and they try to find the appropriate users that will cooperate to accomplish the task ([23] is an exception). However, the issue of trust is not taken into account, although it is obvious that such a cooperation would produce better results if the recruited users have direct social relationships. In this regard, the selected group of users provides no guarantee of quality of service. Moreover, these schemes do not take into account the task allocation problem in a large-scale scenario. A a few exceptions are [28]–[30] however, explicit fairness mechanisms are still missing.

Efficient allocation mechanisms can be constructed by employing community detection: in [31] the authors investigate community detection algorithms for social networks. The objective is to organize and gather the devices available in communities that share mutual interests and characteristics. In [32], a framework that detects different communities of devices is proposed, and trustworthy peers with social relations are identified. Then a machine learning algorithm is used to predict the total time needed to process the requested tasks, by candidates of the same community. The study in [33] proposed an automated framework to handle mobile crowdsourcing requests within a large-scale network. The network is split into multiple virtual communities using the social relations. Both overlapping and non-overlapping communities are detected. Then, a natural language process (NLP)-based approach is executed to capture the information from the textual request to match the with communities, so that a list of candidate devices is found to execute the tasks. A simple algorithm for resource/service discovery within communities is presented in [34]: Two key ideas are combined, i) Detection of communities among established networks and ii) intra-community and inter-community service search algorithms for efficient service discovery among communities. This work also introduced the idea of locality similarity based on the objects position. Wu et al. [35] addressed the case where some exemplar nodes are provided in advance and the set of interested communities is mined for some specific task services.

This work presents a resource allocation strategy for social community clouds. We employ the community detection

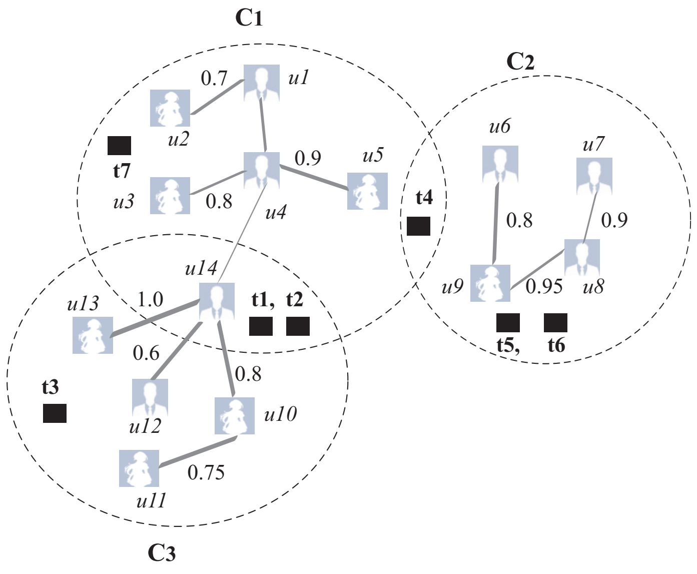

Fig. 2. Example of tasks within communities.

paradigm [36] to guarantee trust and social relationships among the recruited users.

There are plenty of community detection schemes in the literature and the detailed presentation of them is beyond our purposes here. Generally, they can be categorized into three major groups: (1) **Top-down graph approaches:** they divide the overall network into small group [37], [38], (2) **Bottom-up graph approaches:** They start from local structures and expand to the overall network [39], [40], (3) **Machine Learning Schemes:** This is the more recent trend in community detection, which use machine and deep learning strategies [41].

III. Problem Overview and Definitions

Let $G = (V, E)$ be a weighted, undirected graph, where V is the set of nodes and G is the set of edges. The nodes represent the users and the edges represent the relationships between them. The *community detection* paradigm is used to define communities of related users. Let us consider a set of such communities C_i, as shown in Fig.2. Each community contains a number of users (nodes) u_i, which can be connected directly or indirectly via a small number of hops, to indicate a social relationship among them. The social relationship is a sign of trust among the community members. The weight of each edge is used to describe the strength of this relationship. More specifically, the *similarity* $w_{i,j}$ between users u_i and u_j is defined as the weight of the edge that connects i and j and lies in the interval $[0, \dots, 1]$. It is important to point out the existence of *signed networks*, where the weight values can be positive or negative, indicating a positive or negative relationship. In this paper, we do not consider signed networks but only positive relationships. The community users can be considered as candidates to provide for resources for task execution.

Within the system, there are two types of tasks that can be executed: [6], [23]:

- *Internal tasks:* The tasks that can be completed solely within a community. For such tasks, the resources are donated by the community members.
- *External tasks:* The tasks that require collaboration among members of different communities.

Each task T is divided to a set of N activities or subtasks (like execution, or data storage), that is $t = \{t_1, t_2, \dots, t_N\}$. Each community is equipped with a monitor, which is responsible to maintain important information regarding each requested task and the community itself (like subtasks, computing requirements, similarities among users and processing capabilities of each user). Also, based on the current information, it is responsible to select the nodes that will execute each task, using a fair policy that will be described later in this work.

In the example of Fig.2, there are 7 subtasks, $t_1, \dots, t_7$, which have been submitted from user u_{14} (the one within the red frame) and 3 communities, C_1, C_2, and C_3. The communities have been defined using a community detection strategy. Based on the information regarding the computing requirements of each task as well as the computing capabilities of each node, the monitor system has determined which community can serve each subtask. In our example, t_1 and t_2 can be completed by communities C_1, and C_3, while task t_3 can be covered only by the members of C_3. Task t_4 can be covered by the members of C_1 and C_2. Finally, subtasks $t_5, \dots, t_7$ can be covered solely in communities C_2 and C_1, respectively. In this regard, $t_3, t_5, \dots, t_7$ are internal subtasks, while t_1, t_2, and t_4 are external subtasks.

The social relationship guarantees good cooperation, when users are assigned subtasks of the same task, which they have to accomplish. This approach suggests that, when the users are connected either directly or indirectly, the social cost (which is discussed later in this section) of task assignment is reduced. Also, the quality of task service is improved [6], [23].

Based on the problem overview, we need to make a number of definitions.

Definition 1: *A task T consists of a set of smaller subtasks t_n, $n = 1, \dots, N$. The following rules apply:*

- Each subtask t_i is completed by a user that has the sufficient resources or by multiple cooperative users, if this is required. In the second case, the subtask is broken into smaller pieces that we call *processes*.
- The task T is submitted when there are related users with adequate computing capabilities to run all the subtasks, to avoid huge social costs and cooperation issues.
- A task T or a subtask t_i is shared among a set of users that have an average node connectivity degree (see Definition 2) above a certain threshold.
- An internal task is assigned within a community.
- An external task can either be assigned to a member or members of one community or to members of different communities depending on the current computing capabilities and the node connectivity degree of the users involved.

Definition 2: *The social cost c between two nodes i, j is computed by their similarity if they are directly connected or* *by the sum of similarities of the "path" that links the two nodes. If there is no linking at all, then the cost is infinitely large. That is:*

$$c_{i,j} = \begin{cases} \dfrac{1}{w_{i,j}}, \ i \ \text{ and } j \text{ are directly connected} \\ \dfrac{\ell^2}{\sum w_{j\to j}} \ i \ \text{ and } j \text{ are indirectly connected} \\ \infty, \ i \ \text{ and } j \text{ are not connected} \end{cases} \quad (1)$$

where $\sum w_{j\to j}$ indicates the sum of the weights of all the links leading from i to j and ℓ is the number of these links. Apparently, as the similarities of the links are larger, the corresponding social costs are smaller. Also, the direct connections are clearly preferable in terms of cost. In this regard, the average social cost $\bar{c}$ between a node i and a set of K cooperative users L_m is given by

$$\bar{c}_{i,L_m} = \frac{\sum_{K'=1}^{K} c_{i,K'}}{K} \quad (2)$$

Definition 3: *The user suitability is a measure that computes how suitable is a node to implement part of the task submitted by a user $\sqcap$. The suitability of a user depends on its similarity with $\sqcap$ and its computing capabilities:*

$$S_i^{\sqcap} = w_{i,\sqcap} \times \left[\alpha\left(\frac{CP_i}{CP_{\sqcap}}\right) + \beta\left(\frac{M_i}{M_{\sqcap}}\right) + \gamma\left(\frac{D_i}{D_{\sqcap}}\right)\right] \quad (3)$$

where:

- CP is the computing power of a device expressed in Millions of Instructions Per Second (MIPS)
- M is the amount of a device's available RAM memory
- D is the amount of a device's available bandwidth

Equation 4 computes the ration of the computing capabilities (CPU, RAM, disk) of a user's device compared to the device of user $\sqcap$ that submitted the task. The parameters α, β, and γ are used to weigh the suitability value in case the task submitted is source-intensive (CPU-intensive, memory-intensive or bandwidth-intensive). If this is not the case, the three parameters can be equal. The computing capability is multiplied by the similarity value between i and $\sqcap$, which guarantees better quality of service.

Definition 4: *The notion of fairness refers to the decision on how to allocate the resources in such a way that the resource contributors execute the tasks according to their processing capacity [42].*

With the definitions given, the problem we try to solve can by defined as follows:

Definition 5: *Given:*

1) *A task T submitted by a user $\sqcap$, which is broken into a set N of cooperative tasks $t_1, \ldots, t_N$, each requiring one or more users to complete*,
2) *A set of communities C, to which $\sqcap$ is connected either directly or indirectly*
3) *The similarity values and the social cost between $\sqcap$ and the community users, which are stored in the monitor.*
4) *The user suitability to accomplish a subtask or a number of subtasks based on their computing capabilities and their similarity to $\sqcap$*

we try to allocate the set of cooperative tasks among the suitable users in a fair manner based on their computing capabilities so that the number of accomplished tasks increases and the tasks are completed in a timely efficient manner.

TABLE I
NOTATIONS USED IN THIS PAPER

Notation	Description
G(V,E)	A weighted, undirected graph
C	A set of communities
V	A set of graph nodes, each node is a user
E	A set of graph edges, that represent user relationships
T	A task that requires extra resources
t_N	Subtask of T
$w_{i,j}$	Weight of the link connecting two users i, j, denoting the similarity between i and j
u_i	Community user
K	A number of users collaborating to accomplish a task or a subtask
$\sum w_{j\to j}$	The sum of weights of all the links leading from i to j
ℓ	The number of links that connect users i, j indirectly
$c_{i,j}$	The social cost between two nodes i and j
$\sqcap$	The user that submits a task that requires extra resources
$S_i^{\sqcap}$	Suitability of user i to accomplish part of user's $\sqcap$ task
s	The number of selected users after the application of the first phase

Table 1 provides the notations used in this paper.

IV. Our Approach on Resource Allocation

Our approach is composed of two stages: the first stage applies a selection policy to pick-up the appropriate candidates to implement the task. The second stage is the fair resource allocation policy among the selected candidates.

A. Candidate Selection

In the Social Internet of Things (SIoT) context , the communities are formed by smart devices, which establish social relationships autonomously via their users. Our fair resource allocation scheme requires an efficient service discovery mechanism, which is input by a set of selected representative objects or leaders ([23], [34]). The representatives periodically collect important data like the user similarities and the network connectivity degrees based on the users' social behavior, as well as the computing capabilities of the various devices. This information is fed into our community detection scheme [43], which provides updated communities as well as similarities between different communities. The later information is very useful in cases where the request for resources has to be forwarded outside the community. Our community detection scheme implements pipeline-based parallel processing techniques on a typical data structure like binary trees, which have been enhanced with threads to accelerate processing. It has been implemented over weighted networks with irregular topologies and it is based on a stepwise path detection strategy, where each step finds a link that increases the overall strength of the path being detected. Its functionality has been verified in a number of real world data sets like Facebook, Twitter,

Algorithm 1: Candidate Selection

```
input : Users suitability $S_i^{\sqcap}$ and the updated community data from [43] (similarities between users $w_{i,j}$ )
output: The group of users $\mathcal{G}$ that will execute tasks submitted by $\sqcap$
1  begin
2  Set suitability threshold $\mathcal{T}'_{S_i^{\sqcap}}$
3  // Update lists of suitable nodes
4  for all $m' \in [1, \ldots, m]$ such that $\sqcap \in C_m$ do
5      $L_{m'} \leftarrow \emptyset$
6      for all $u_i \in C_{m'}$ do
7          compute $S_i^{\sqcap}$ from Eq. (3)
8          if $S_i^{\sqcap} \geq \mathcal{T}'_{S_i^{\sqcap}}$ then
9              $L_{m'} \leftarrow L_{m'} \cup u_i$
10         end if
11     end for
12 end for
13 // Remove unrelated nodes from the lists
14 for $m' = 1$ to $m$ do
15     if $\exists\ i$ such that $c_{i,j} = \infty\ \forall\ j$ then
16         $L_{m'} \leftarrow L_{m'} \backslash \{i\}$
17     end if
18 end for
19 // Compute the average social cost and sort
20 $L \leftarrow L_1, \ldots, L_m$ // List of lists
21 for $m' = 1$ to $m$ do
22     for all $s \in L_{m'}$
23         Compute the average social cost by Eq.(2)
24     end for
25 end for
26 Sort $L$ by ascending average social costs
27 Generate final candidate $\mathcal{G}$.
       $\mathcal{G} = \begin{cases} L\backslash\{L_{m'}\},\ m' = k, \ldots, m, & \textit{non-exhausting} \\ L, & \textit{exhausting} \end{cases}$
```

Google+, Pokec, and LiveJournal. All the updated information provided by our scheme is stored on the monitor, to be used upon a new task submission. We advocate the use of our scheme because of its logarithmic running time (based on threaded binary tree structures).After the submission of a task T from user $\sqcap$, the candidate selection is implemented as shown in Alg. 1.

As shown in Algorithm 1, we initially set a suitability threshold value (line 2) and then for each community defined by the community detection scheme we initialize the list of all the possible candidate users (line 5). Then, we use Eq.(3) to compute each users suitability and if the value computed is above the threshold we add the user to the candidate list $L_{m'}$ (line 9). Then, we need to remove the list nodes, which are completely unrelated, as they are considered unsuitable to cooperate in order to complete a task. This can be examined by checking if, for a node i, its communication cost with every other node j within the community is ∞ (line 15). Such nodes are removed from the candidate list $L_{m'}$ of each community m' (line 16). All the lists $L_{m'}$ are merged into L (line 20), which contains all the candidates from all the neighbors. Then, for each list $L_{m'}$ that has remained after the update, we compute the average social costs by Eq. (2) (line 23). Then, we sort list L, in ascending order of the average social costs of the sublists (communities) L_m, which were previously computed. In the end, the final candidate list $\mathcal{G}$ can be generated by adding the communities found in L: if the users found in the first k communities are enough to complete all the submitted subtasks (non-exhaustive approach), then these communities are selected. Otherwise, all the communities in L may be needed. This usually occurs when there are many internal tasks which need to be assigned to a specific community or a few communities only.

The computational cost for the list update (lines 4-12) is $O(mu_i)$. In lines 13-18. we remove the nodes which are completely unrelated from the lists. The complexity is $O(ms)$, where s is the number of nodes per community, as defined in the list update phase. Similarly, the average social cost computation is also $O(ms)$. Finally, the list sorting is $O(m)$. Therefore, the total complexity of Algorithm 1 can be easily obtained to be at most $O(mu_i)$, which is linear on the number of community users.

Algorithm 1 guarantees that the candidate nodes will cooperate to complete the tasks assigned to each community. Moreover, all the assigned selected candidates have a bounded social cost (not infinite), since they are all related to the submitter. In the example of Fig.2, the submitting node u_{14} indirectly connects all the nodes in communities C_1 to C_3. In the next paragraph, we discuss the issue of assigning the subtasks equally among the candidate nodes.

B. Fair Task Allocation Policy

We let $s = u_i'$ represent the overall number of selected users found in the selection stage. Particularly, throughout this work, we use the index value of 1 for the submitting node and indices 2 to $s{+}1$ for the selected users. Also, we assume that we divide the total time in small time slots, where a time slot is the time it takes for a user to complete a portion of its assigned work. The *state* of our network with $s+1$ elements is given by a vector $\mathbf{N} = (N_1, N_2, \ldots N_{s+1})$, where N_i is the number of subtasks being assigned to a user and N is the overall number of tasks, that is $\sum_{i=2}^{s+1} N_i = N$. In this regard, our system can be modeled as a Markov system model. A Markov proposes model is irreducible, that is, at any time, each state can be reached from any other state with non-zero probability [44]. Therefore, the equilibrium state probability distribution can be derived as follows:

$$P(\mathbf{N}) = F(N)\ P(N, 0, \ldots, 0), \tag{4}$$

where and $P(N, 0, \ldots, 0)$ is the probability that all the tasks are located in the submitting user x_1 and the normalization function $F(N)$, is computed by [44]:

$$F(N) = \sum_{\text{for all N}} \prod_{i=2}^{s+1} x_i^{N_i} \tag{5}$$

The x_i's of Eq. 5 are the users' *relative processing capacity* which are expressed as the ratio of user's i suitability compared to the suitability of the submitter to execute his own tasks.

$$x_i = \frac{S_i^{\sqcap}}{S_1^{\sqcap}}, \tag{6}$$

1) Computing the User Utilization: An important aspect of our proposed community cloud scheme is, at any time, to keep the users with the highest relative capacity as utilized as possible. In a Markov process model, the state of the system is described, at any time, by the number of tasks distributed among the system nodes. To compute the utilization of each user, we must consider *only the states* where $N_i > 0$, that is, the user is utilized. This is expressed by $G_i(N)$:

$$G_i(N) = \sum_{\substack{\text{for all N,}\\ N_i>0}} \prod_{i=2}^{s+1} x_i^{N_i}, \tag{7}$$

The *utilization* p of each user i is then computed by dividing $G_i(N)$ by $F(N)$, where $F(N)$ includes also the states where user i can be completely unutilized:

$$p_i = \frac{G_i(N)}{F(N)}. \tag{8}$$

The $F(N)$ function can be efficiently computed by using the recursive function [44]:

$$F(N) = F_{i-1}(N) + x_i F_i(N-1), \text{ for all } N,\ i = 2, \ldots, s+1 \tag{9}$$

To compute the $G_i(N)$ values for all users i, we need to subtract from $F(N)$ all the terms for which $N_i = 0$, that is, all the combinations of task distributions for which user i is assigned no tasks:

$$G_i(N) = F(N) - \sum_{\substack{\text{for all N,}\\ N_i=0}} \prod_{i=2}^{s+1} x_i^{N_i} \tag{10}$$

For a user i, the products for which $N_i = 0$ are in the form (here, we use index j for the remaining users other than i):

$$x_i^0 \prod_{j=2}^{s+1} x_j^{N_j} \tag{11}$$

Thus, Eq. 10 becomes:

$$\begin{aligned} G_i(N) = F(N) - \Big[& x_i^0 \left(x_2^0 x_3^0 \cdots x_{s+1}^N\right) \\ & + x_i^0 \left(x_2^0 x_3^0 \cdots x_s^1\ x_{s+1}^{N-1}\right) \\ & + x_i^0 \left(x_2^0 x_3^0 \cdots x_{s-1}^1\ x_s^0\ x_{s+1}^{N-1}\right) \\ & + x_i^0 \left(x_2^1 x_3^0 \cdots x_s^0\ x_{s+1}^{N-1}\right) \\ & + \qquad \vdots \\ & + x_i^0 \left(x_2^N x_3^0 \cdots x_{s+1}^0\right) \Big] \end{aligned} \tag{12}$$

Proposition 1: During a task allocation problem, the users are utilized according to their relative processing capacity.

Proof: Consider two users i_1 and i_2 with $x_{i_1} > x_{i_2}$ (x_{i_1} and x_{i_2} will appear in boldface during the proof, to separate them from the remaining relative processing capacities). The utilizations p_{i_1} and p_{i_2} are given by Eq.8. We will show that $p_{i_1} > p_{i_2}$. From Eq. 12, for x_{i_1} we have:

$$G_{i_1}(N) = F(N) - x_{i_1}^0 \prod_{j=2}^{s+1} x_j^{N_j} = F(N) \tag{13}$$

$$\begin{aligned} - \Big[& \mathbf{x_{i_1}^0} \left(\mathbf{x_{i_2}^0} x_3^0 \cdots x_{s+1}^N\right) \\ & + \mathbf{x_{i_1}^0} \left(\mathbf{x_{i_2}^0} x_3^0 \cdots x_s^1\ x_{s+1}^{N-1}\right) \\ & + \mathbf{x_{i_1}^0} \left(\mathbf{x_{i_2}^0} x_3^0 \cdots x_{s-1}^1\ x_s^0\ x_{s+1}^{N-1}\right) \\ & + \mathbf{x_{i_1}^0} \left(\mathbf{x_{i_2}^1} x_3^0 \cdots x_s^0\ x_{s+1}^{N-1}\right) \\ & + \qquad \vdots \\ & + \mathbf{x_{i_1}^0} \left(\mathbf{x_{i_2}^N} x_3^0 \cdots x_{s+1}^0\right) \Big] \end{aligned} \tag{14}$$

To obtain $G_{i_2}(N)$, we simply have to mutually exchange the positions for factors x_{i_1} and x_{i_2} in Eq. 13:

$$\begin{aligned} G_{i_2}(N) = F(N) - \prod_{j=2}^{s+1} x_j^{N_j} = F(N) - \Big[& \mathbf{x_{i_2}^0} \left(\mathbf{x_{i_1}^0} x_3^0 \cdots x_{s+1}^N\right) \\ & + \mathbf{x_{i_2}^0} \left(\mathbf{x_{i_1}^0} x_3^0 \cdots x_s^1\ x_{s+1}^{N-1}\right) \\ & + \mathbf{x_{i_2}^0} \left(\mathbf{x_{i_1}^0} x_3^0 \cdots x_{s-1}^1\ x_s^0\ x_{s+1}^{N-1}\right) \\ & + \mathbf{x_{i_2}^0} \left(\mathbf{x_{i_1}^1} x_3^0 \cdots x_s^0\ x_{s+1}^{N-1}\right) \\ & + \qquad \vdots \\ & + \mathbf{x_{i_2}^0} \left(\mathbf{x_{i_1}^N} x_3^0 \cdots x_{s+1}^0\right) \Big] \end{aligned} \tag{15}$$

Let us consider separately the products subtracted from $F(N)$ for to produce G_{i_1} and G_{i_2}. From Eq. 13 and Eq. 14, the first products subtracted from $F(N)$ are

$$\mathbf{x_{i_1}^0} \left(\mathbf{x_{i_2}^0} x_3^0 \cdots x_{s+1}^N\right) \qquad \text{and} \qquad \mathbf{x_{i_2}^0} \left(\mathbf{x_{i_1}^0} x_3^0 \cdots x_{s+1}^N\right)$$

for G_{i_1} and G_{i_2} respectively. Then, the products

$$\mathbf{x_{i_1}^0} \left(\mathbf{x_{i_2}^0} x_3^0 \cdots x_s^1\ x_{s+1}^{N-1}\right) \qquad \text{and} \qquad \mathbf{x_{i_2}^0} \left(\mathbf{x_{i_1}^0} x_3^0 \cdots x_s^1\ x_{s+1}^{N-1}\right)$$

are subtracted. One can easily notice that as long as the exponents of $\mathbf{x_{i_1}}$ and $\mathbf{x_{i_2}}$ are zero, the products subtracted are equal. However, let us consider the portions of $\prod_{j=2}^{s+1} x_j^{N_j}$ for which the the exponents of $\mathbf{x_{i_1}}$ and $\mathbf{x_{i_2}}$ are non-zero. These factors are

$$\mathbf{x_{i_1}^0} \left(\mathbf{x_{i_2}^1} x_3^0 \cdots x_s^0\ x_{s+1}^{N-1}\right) \quad \text{and} \quad \mathbf{x_{i_2}^0} \left(\mathbf{x_{i_1}^1} x_3^0 \cdots x_s^0\ x_{s+1}^{N-1}\right) \tag{16}$$

$$\vdots$$

$$\mathbf{x_{i_1}^0} \left(\mathbf{x_{i_2}^N} x_3^0 \cdots x_{s+1}^0\right) \qquad \text{and} \qquad \mathbf{x_{i_2}^0} \left(\mathbf{x_{i_1}^N} x_3^0 \cdots x_{s+1}^0\right) \tag{17}$$

Comparing the pairs, we can easily see that (because $\mathbf{x_{i_1}} > \mathbf{x_{i_2}}$).

$$\begin{aligned}\mathbf{x^0_{i_1}}\left(\mathbf{x^1_{i_2}} x^0_3 \cdots x^0_s\ x^{N-1}_{s+1}\right) &< \mathbf{x^0_{i_2}}\left(\mathbf{x^1_{i_1}} x^0_3 \cdots x^0_s\ x^{N-1}_{s+1}\right)\\ \vdots &< \qquad \vdots\\ \mathbf{x^0_{i_1}}\left(\mathbf{x^N_{i_2}} x^0_3 \cdots x^0_{s+1}\right) &< \mathbf{x^0_{i_2}}\left(\mathbf{x^N_{i_1}} x^0_3 \cdots x^0_{s+1}\right)\end{aligned}$$

Thus, the sum of the factors subtracted from $F(N)$ is larger for user i_2 (the one with the smaller relative capacity $\mathbf{x_{i_2}}$. Thus, $G_{i_2} < G_{i_1} \Rightarrow \frac{G_{i_2}}{F(N)} < \frac{G_{i_1}}{F(N)}$ or $p_{i_1} > p_{i_2}$. Thus, the users are utilized according to their relative processing capacity. ■

Proposition 2: The utilization computation has linear complexity

Proof: This derives simply from the way the terms involved in Eq. 8 are computed. See Eq. 9. The $F(N)$ terms are computed by a linear recursive function. The $G_i(N)$ factors are the $F(N)$ factors subtracted by the factors produced in cases when $N_i = 0$, so they are also linear. This completes the proof. ■

2) Fair Task Allocation: To embed fairness in the Markov process model, we need to distribute the workload of N tasks in such a manner that each user takes on the workload proportionally to its processing capacity and (ii) the variance of all the user utilizations approaches zero for a set of homogeneous, in terms of processing power, users. The steps of the fair task allocation strategy are as follows:

`STEP 1`: For each user we compute r, the sum of all the relative processing capacities:

$$r = \sum_{i=2}^{s} x_i \tag{18}$$

`STEP 2`: The *load distribution factor*, f, can simply be computed as the fraction of N and r:

$$f = \frac{N}{r}, \tag{19}$$

`STEP 3`: The workload per user is computed as

$$\mathcal{W}_i = f \times x_i, \quad i = 2, \ldots, s \tag{20}$$

Clearly, the three steps described distribute the tasks proportionally to the x_i values. The complexity of the fair allocation scheme is clearly linear, depending on the number of members that form $\mathcal{G}$. We introduce the $\delta(t)$, which computes the total variance between the average utilization and the utilization of each user:

$$\delta(p) = \frac{\sum_{i=2}^{s}\left[\overline{\delta(p)} - p_i\right]^2}{s} \tag{21}$$

where s is the number selected community users from the first stage and $\overline{\delta(p)}$ is the average utilization given by:

$$\overline{\delta(p)} = \frac{\sum\limits_{i=2}^{s} p_i}{s} \tag{22}$$

We use Proposition 2 to prove the fairness of our scheme in terms of the users' utilization.

Proposition 2: The variance among all the utilizations approaches zero for a homogeneous, in terms of processing power, set of users.

Proof: The proposition applies in case all the selected users have equal processing capacities, where the variance is zero. Let is consider the variance of the utilizations between two users i_1 and i_2. From Equations 16 and 17, it is clear that the utilization difference among the system users is clearly driven by their x_i^N values:

$$p_{i_1} - p_{i_2} = \frac{(\mathbf{x^1_{i_2}} - \mathbf{x^1_{i_1}})(x^0_3 \cdots x^0_s\ x^{N-1}_{s+1}) + \cdots + (x^N_{i_2} - x^N_{i_1})(x^0_3 \cdots x^0_{s+1})}{F(N)} \tag{23}$$

Obviously, the difference between the two utilizations is dictated by the fraction $\frac{x_{i_1}}{x_{i_1}}$. When this fraction approaches 1, the numerator in Eq. 22 reduces, *regardless of the number of tasks*. Also, two different pairs of users, say (i_1, i_2) and (i_3, i_4) for which $\frac{x_{i_1}}{x_{i_1}} = \frac{x_{i_3}}{x_{i_4}}$ will produce the same term in the computation of $\delta(p)$. Thus, for infinitely large number of executed tasks, the difference $\overline{\delta(p)} - p_i$ can be considered to be bounded by a relatively small value Δ over a set of homogeneous, in terms of processing capacity, users. ■

From Proposition 2, it is made clear that the proposed model does not exhaust the selected users, in the sense that they are loaded according to their processing capacity. For an almost homogeneous network of selected users, the contributors are almost equally loaded. Queues appear when a set of users is, to a large extent, more computationally powerful compared to others, thus their utilization increases thus causing queues of waiting tasks. In this regard, our model produces quite short task queues. Regularly, the tasks remain in the queues just for the time required until they are executed. This will be shown in the simulation results.

C. Combining Parts to Create a Big Picture of our Approach

In this paragraph, we briefly show how to combine the ideas presented in the previous paragraph, to create a big picture of our scheme. We use Algorithm 2 for this purpose.

We initially apply Algorithm 1, in order to allocate the users which have the capacity and are sufficiently trusted by user $\sqcap$ to share their resources. Then, the processing capacity of each user is computed. According to the Markovian model we have described, each user's utilization must be proportional to his/her computing capacity. These utilization values are computed, verified and used to test for possible over-utilizations. These computations indicate that, under the current community structure, relationships and processing capacities, each user will be utilized by a certain percentage, which is based on his/her processing capacity. In case some of these values approach or equal 1, there is a clear indication that one user (or many users) may be overloaded. Then, the stepwise task allocation procedure that follows should be aware of this fact and remove a portion of this extra load from the overloaded

Algorithm 2: Overall View of the Proposed Model

```
1 begin
2   Apply Algorithm 1 to find the group of users $\mathcal{G}$ which
      are suitable execute tasks submitted by $\sqcap$
3   For $\mathcal{G}$, compute the relative processing capacity of all
      its users, using Eq. 6
4   Compute each users utilization according to Eq. 8.
      Make sure that no over-utilization occurs and verify
      that all the users are utilized based on their
      processing capacity.
5   In case very large value is found (close or equal to 1,
      which denotes possible over-utilization), some of the
      load must be distributed to another user during the
      load distribution step (Step 6 that follows).
6   Distribute the overall number of tasks proportionally to
      the selected users, by applying Eq. 17- Eq 19 in a
      stepwise fashion (Steps 1-3 of the fair task allocation
      strategy).
7 end
```

user/users. In this scenario, this extra load may be shared equally among the other users. Also, the community detection scheme should be able to detect overlaps. Then, such a re-assignment can be found more easily.

V. Experimental Results and Discussion

In this section, we provide experimental results to test and validate the performance of our model. The objective of these experiments is to test the efficiency of our solution in terms of throughput, response time and source (CPU, memory, disk) utilization. For this reason, we have conducted our experiments using CloudSim environment. The machines are equipped with an Intel Core i7-8559U Processor system, with clock speed at 2.7GHz, four cores and two threads/core, for a total of 8 logical processors and 16 GB RAM. The basic components of Cloudsim, which enable the simulation of realistic cloud environments are (see [45]): (1) *Datacenter:* It is used to model the core services at the system level of a cloud infrastructure. It consists of a set of hosts which manage a set of virtual machines whose tasks are to handle ”low level” processing, and at least one data center must be created to start the simulation, (2) *Host:* It is used to assign processing capabilities (which is specified in the milion of instruction per second that the processor could perform), memory and a scheduling policy to allocate different processing cores to multiple VMs, (3) *Virtual machines:* This component manages the allocation of different virtual machines different hosts, (4) *Datacenter broker:* The broker identifies which service provider is suitable for the user based on the information it has from the Cloud Information Service, (5) *Cloudlet:* It represents the application service whose complexity is modeled in CloudSim in terms of the computational requirements, and (6) *CloudCoordinator:* It manages the communication between other CloudCoordinator services and brokers.

For the cloud, we create a datacenter with 6 physical machines each of which is assigned with a number of VMs varying from 5 to 50. Every VM is equipped with a CPU with a capacity of MIPS, RAM capacity of 16 GB and hard disk storage of 512 GB. We used real datasets of an existing application, the typical word count. Each tasks process a part of a large file and seeks for all the words starting from a selected letter. Once the task is complete, the proper data are given back to the cloud. Our experimental assumptions are the following: (1) We assumed an incremental scenario, where we start from 20 tasks and we gradually increment the number of tasks to 200. In this way, we prove that our scheme can improve the performance for different task numbers, (2) We assign a predefined probability to each user to complete a task. In cases when a user fails to complete a series of tasks, we perform rescheduling and the tasks are assigned to one or more users. Community overlaps are important to locate more candidates for this purpose. This assumption is made to have fair comparison to the compatible schemes, (3) The user’s utilization approach 1 only in cases when their capacity is significantly higher compared to other users in $\mathcal{G}$. Typically, our model prevents such cases.

TABLE II
Social networks and their features

Network	Network size	Number of Selected Nodes
ego-Facebook	4.39	170
Wiki-vote	7.115	405
musae Facebook	22.470	620
feather-deezer	28.291	780
musae twitch	34.118	955

For the community formulations, we we used 5 relatively small real-world datasets, namely ego-Facebook, wiki-vote, musae facebook, feather-deezer, and musae twitch. The data we used is available online at [46]. The network sizes vary from 4.000 - 38.000 nodes which are enough for serving our purposes. Table 2 shows the network size and the number of user that were selected from the first stage of our scheme from the aforementioned networks.

To evaluate our work, we compare various performance metrics like the allocation rate, the throughput, and the execution time against two very recent state-of-the-art works, namely [17] and [28]. In [17], the authors develop a trust model that uses objective and subjective sources of trust to optimize the credibility of the trust scores. Based on these trust relationships, the hypervisor can learn about the optimal detection load percentage that should be allocated to each of its guest VMs in real time. In [28], the authors propose a service computing framework for time constrained-task allocation systems. This framework relies on a recruitment algorithm that implements a multi-objective task allocation algorithm based on Particle Swarm Optimization and a queuing scheme that efficiently handles the incoming tasks. Also, a task delegation mechanism is used to avoid delays. The selection of users is based on a reputation management component, which manages the reputation of users based on their sensing activities. There are three reasons behind the choice of the comparable schemes: (1) The are novel, state of the art schemes, (2) They develop trust models as well as task allocation/load distribution

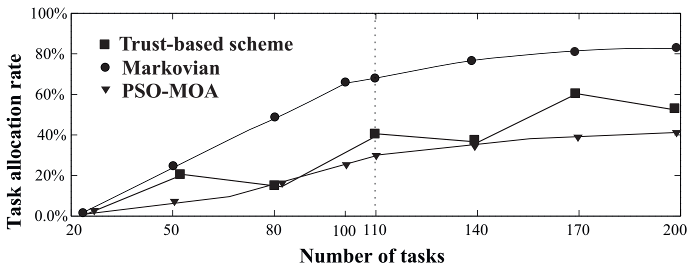

Fig. 3. Task allocation rate.

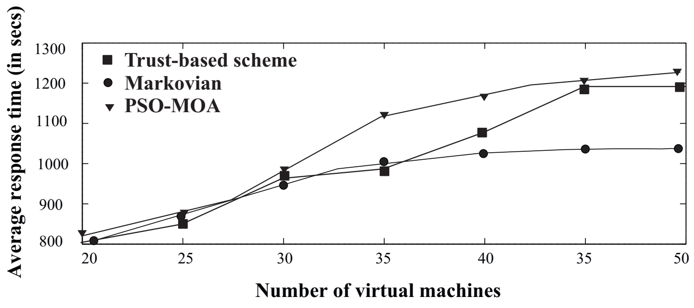

Fig. 4. Average response times.

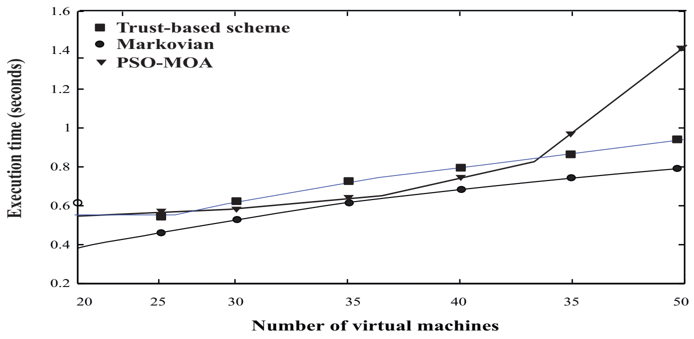

Fig. 5. Average execution times.

schemes, and (3) They provide straightforward comparisons to performance metrics we also consider.

A. Allocation Rate and Response Times

The task allocation paradigm which is examined involves a task queue within the service discovery and matching mechanism which implements all the scheduling. In this paragraph, we discuss the *allocation rate*. This metric represents the percentage of tasks being effectively allocated to workers and completed successfully. We compare our scheme against the PSO-MOA (Particlie Swarm Optimization-Based Multi-Objective Task Allocation Algorithm) and the trust-based schemes.

The task allocation rate increases by long queues which can incur because of the following reasons: (1) unavailability of users to implement the tasks, and (2) rescheduling due to system changes (for example, some users disconnect from the network while other users are connected). In our experiments, the contributing users were used exhaustively only when they crushingly overpower others in terms of processing capacity. This results in long task queues for those users. However, when the network of contributing users is composed of somehow homogeneous (although not similar) processing elements, their utilization variance is small, there is no over-utilization and the queue lengths are short. Moreover, in case re-scheduling is necessary, the computations involved in the Markovian model of Section 4.2 take only linear time. On the average, the PSO-MOA and the trust-based models generated longer queues, thus our model has higher allocation rate as can be seen in Fig. 3. During our simulations, we forced a re-scheduling when the number of tasks exceeded 110 (see the vertical line) to indicate the fact that our task allocation rate plot remains smooth while the other two schemes incur certain reduction in its task allocation rate before improving it later.

Also, we study the average time the machines need to respond to the tasks being distributed to them. The response time is taken by the average of all the differences between the time a task is assigned and the time it starts execution. As a result of obtaining shorter queues and higher allocation rates, our model reduces the average response time compared its competitors, as shown in Fig. 4.

B. Total Execution Time

In this section, we compare the total execution time between the Markovian model, the PSO-MOA scheme and the trust-based scheme. The trust-based model uses a fair allocation model that simply divides the load equally among the machines, thus reducing the total time spent on computing an optimal distribution, but it needs time to collect and compute the trust scores. On the other hand, the PSO-MOA scheme is based on a priority queue system, which queues the tasks until a potential user is found to execute it. This priority is defined by the response time. In this regard, the PSO-MOA scheme selects the task that needs to be served first according to the remaining response time. As its response time was found larger (see Fig. 4) the PSO-MOA had the largest execution time among the the compared schemes. Our work executes the tasks proportionally to the processing capacities of the selected machines and, in this regard, it manages to outperform the compared schemes in terms of execution time (see Fig.5).

C. Utilization

In the final set of experiments, we study the effectiveness of the Markovian model in terms of the resources being used. We compare the CPU, memory and bandwidth consumption to the results found in the trust-based scheme. The trust-based scheme relies on a defensive hypervisor policy that aims at optimizing the resource allocation strategy by detecting possibly malicious requests. However, the completely fair policy it uses, where the loads are equally distributed results in exhausting some of the resources, especially the resources of the “weaker” machines. Another disadvantage of this policy is that, as the number of tasks (and thus the number of user machines) increases, such detection are not always possible or they are time-consuming. In this regard, the performance decreases (more and more resources are utilized) as the number of machines increases, as can be seen in Figures

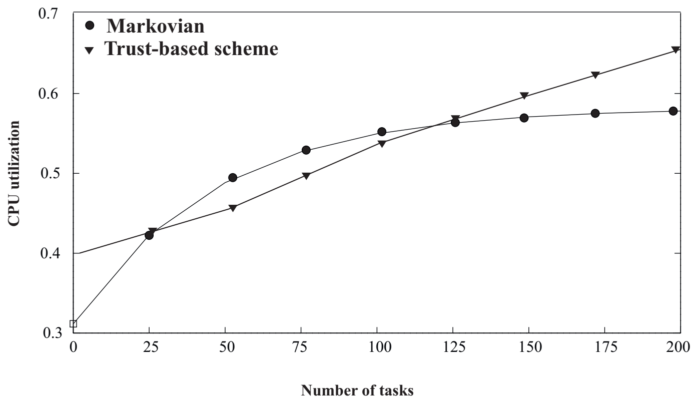

Fig. 6. Average CPU utilization.

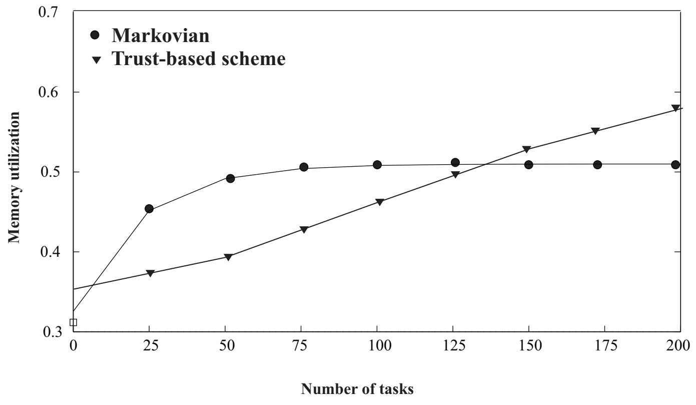

Fig. 7. Average memory utilization.

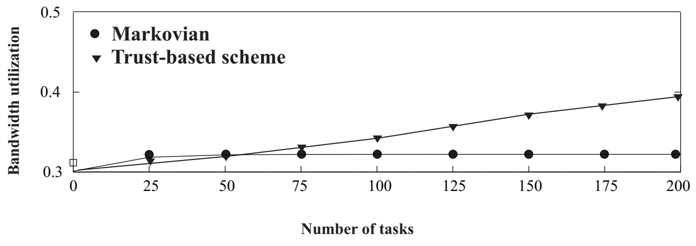

Fig. 8. Average bandwidth utilization.

6 to 8. Indeed, for small number of tasks this policy appeared to have better results compared to the Markovian based model. On the other hand, our proposed scheme does not over-utilize any resources (see Proposition 2) regardless of the number of tasks and the load distribution computations are linear (see Eq. 9). As can be seen from Figures 6 to 8, the utilization curves for our strategy appear to be rather smooth as the number of tasks keep on increasing. Also, recall in our examples that we considered that our tasks are mostly CPU and memory-intensive. This explains the fact that the average CPU and memory utilizations were larger compared to the bandwidth utilization.

VI. Conclusions - Future Work

This paper presented a framework for task allocation in social community clouds. It consists of the candidate selection phase and the task allocation algorithm. The user selection policy uses a community detection strategy to locate the suitable users which will implement the tasks. The task allocation scheme was implemented using the Markov-based model, which can be used to fairly assign the tasks among the users based on their processing capacity. The users are utilized based on their capacity. Our scheme was compared to two other schemes: the trust-based scheme and the PSO-MOA scheme. The results have shown that our scheme improves the task allocation rate by about 20-30%, because of the shorter queues it produces and thus lower response times. Also, our strategy decreases the total execution time by about 40% compared to the trust-bases scheme. This improvement is even higher when compared to the PSO-MOA scheme, when the number of VMs increases. Finally, our scheme was found to use the system resources more efficiently without exhausting them, as the number of assigned tasks keeps on increasing. In the future, we will investigate the use of open network models in implementing community clouds. Also, we will try to embody the Markov based model as a benchmark to assess the results of a community detection scheme. Finally, we need to investigate signed networks and relative relationships and add them to the candidate selection process. Basing resource allocations only on positive relationships is adequate, but it does not tell the whole story. There are many different types of relationships (e.g. family, friends, colleagues, simple followers and so on). These relationships can include negative feelings and judgments among users, without necessarily affecting the trust between them. Moreover, there can be different trust levels corresponding to different types relationships (for example an opposed family member can still be trusted while an opposed follower cannot).

References

[1] C. Liu, K. Li, and K. Li, "A game approach to multi-servers load balancing with load-dependent server availability consideration," *IEEE Trans. Cloud Comp.*, vol. 9, no. 1, pp. 1–13, 2021.

[2] S. Caton, C. Haas, K. Chard, K. Bubendorfer, and O. Rana, "A social compute cloud: Allocating and sharing infrastructure resources via social networks," *IEEE Trans. Serv Comp.*, vol. 7, no. 3, pp. 359–372, 2014.

[3] S. S., A. S., and K. S., "A social compute cloud: Allocating and sharing infrastructure resources via social networks," *Applied Sciences*, vol. 11, no. 16, p. 7179, 2021.

[4] K. Chard, S. Caton, O. Rana, and p. y. K. Bubendorfer, booktitle=IEEE 3rd International Conference on Cloud Computing, "Social cloud: Cloud computing in social networks."

[5] G. Cui, Z. Wang, L. Dong, X. Cao, Y. Liu, and Z. Zhang, "Influence of contribution-based resource allocation mechanism on individual resource sharing cooperation in social networks," *International Journal of Modern Physics C*, vol. 30, no. 12, p. 2050007, 2019.

[6] J. I. Petri, Diaz-Montes, M. O. Rana, Punceva, I. Rodero, and M. Parashar, "Modelling and implementing social community clouds," *IEEE Transactions on Services Computing*, vol. 10, no. 3, pp. 410–422, 2015.

[7] F. Hao, G. Min, J. Chen, F. Wang, M. Lin, C. Luo, and L. T. Yang, "An optimized computational model for multi-community-cloud social collaborative computing," *IEEE Trans Serv. Comput.*, vol. 7, no. 3, pp. 346 – 358, 2014.

[8] F. Hao, D.-S. Park, J. Kang, and G. Min, "2l-mc^3: A two-layer multi-community-cloud/cloudlet social collaborative paradigm for mobile edge computing," *IEEE Internet of Things.*, vol. 6, no. 3, pp. 4764 – 4773, 2019.

[9] C. R, G. J., and B. F, "Trust management for soa-based iot and its application to service composition," *IEEE Trans Serv. Comput.*, vol. 9, no. 3, p. 482–495, 2014.

[10] C. IR, B. F, and G. J, “Trust-based service management for social internet of thing systems,” *IEEE Trans. Dependable Secure Comput*, vol. 13, no. 6, pp. 684–696, 2015.

[11] B. F, C. I.R, and p. y. Guo J, booktitle=IEEE Eleventh International Symposium on Autonomous Decentralized Systems (ISADS), “Scalable, adaptive and survivable trust management for community of interest based internet of things systems.”

[12] V. Sharma, I. You, D. Nalin, and M. Atiquzzaman, “Cooperative trust relaying and privacy preservation via edge-crowdsourcing in social internet of things,” *Future Generation Computer Systems*, vol. 92, no. 6, p. 758–776, 2018.

[13] M. M. Rad, A. M. Rahmani, A. Sahafi, and N. N. Qader, “Social internet of things: vision, challenges, and trends,” *Human-centric Computing and Information Sciences*, vol. 10, no. 52, pp. 1–40, 2020.

[14] B. R. e. a. Roopa MS, Pattar S, “Social internet of things (siot): foundations, thrust areas, systematic review and future directions,” *Comput. Commun*, vol. 139, pp. 32–57, 2019.

[15] O. B. Abderrahim, M. H. Elhedhili, and L. Saidane, “Ctms-siot: A context-based trust management system for the social internet of things,” in *13th International Wireless Communications and Mobile Computing Conference (IWCMC)*, 2017, pp. 1903–1908.

[16] N. M, G. R, and A. L, “Trustworthiness management in the social internet of things,” *IEEE Trans Knowl Data Eng*, vol. 26, no. 5, p. 1253–1266, 2014.

[17] O. A. Wahab, J. Bentahar, H. Otrok, and A. Mourad, “Optimal load distribution for the detection of vm-based ddos attacks in the cloud,” *IEEE Trans. on Serv Comp*, vol. 13, no. 1, pp. 114–129, 2020.

[18] X. H, S. N, and C. B, “Guarantor and reputation based trust model for social internet of things,” in *International Wireless Communications and Mobile Computing Conference (IWCMC)*, 2015, pp. 600–605.

[19] T. NB, U. TW, and L. GM, “A reputation and knowledge based trust service platform for trustworthy social internet of things,” in *19th International ICIN Conference - Innovations in Clouds, Internet and Networks*, 2016, pp. 430–441.

[20] V. T. Kokoris-Kogias E, Voutyras O, “Trm-siot: A scalable hybrid trust and reputation model for the social internet of things,” in *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2016, pp. 1–9.

[21] K. M and V. ML, “Trust management in the social internet of things,” *Wireless Peers Communications*, vol. 96, no. 2, p. 2681–2691, 2017.

[22] K. AM and V. ML, “Trust management for reliable decision making among social objects in the social internet of things,” *IET Networks*, vol. 6, no. 4, pp. 75–80, 2017.

[23] W. Tan, L. Zhao, B. Li, L. D. Xu, and Y. Yang, “Multiple cooperative task allocation in group-oriented social mobile crowdsensing,” *IEEE Trans. on Serv. Comp.*, vol. Early Access, pp. 1–1, 2021.

[24] J. Wei, Y. Lin, X. Yao, and J. Zhang, “Differential privacy-based location protection in spatial crowdsourcing,” *IEEE Trans. on Serv. Comp*, vol. 15, no. 1, pp. 45 – 58, 2019.

[25] F. Basık, B. Gedik, H. Ferhatosmanoglu, and K.-L. Wu, “Fair task allocation in crowdsourced delivery,” *IEEE Trans. on Serv. Comp*, vol. 14, no. 4, pp. 1040 – 1053, 2018.

[26] Y. Li, W. Xu, and M. L. Yiu, “Client-side service for recommending rewarding routes to mobile crowdsourcing workers,” *IEEE Trans. on Serv. Comp*, vol. 14, no. 6, pp. 1995 – 2010, 2019.

[27] L. Zhao, W. Tan, N. Xie, and L. Huang, “An optimal service selection approach for service-oriented business collaboration using crowd-based cooperative computing,” *Applied Soft Computing*, vol. 92, p. 106270, 2020.

[28] R. Estrada, R. Mizouni, H. Otrok, A. Ouali, and J. Bentahar, “A crowd-sensing framework for allocation of time-constrained and location-based tasks,” *IEEE Trans. on Serv. Comp, year=2020, volume=13, number=5, pages=769–785*.

[29] Y. Liu, B. Guo, Y. Wang, W. Wu, Z. Yu, , and D. Zhang, in *ACM Int. Joint Conf. Pervasive Ubiquitous Comput*, 2016, pp. 403–414.

[30] B. Guo, Y. Liu, W. Wu, Z. Yu, and Q. Han, “Activecrowd: A framework for optimized multitask allocation in mobile crowdsensing systems,” *IEEE Trans. Human-Mach Syst.*, vol. 47, no. 3, p. 392–403, 2017.

[31] A. Khanfor, H. Ghazzai, Y. Yang, and Y. Massoud, “Application of community detection algorithms on social internet-of-things networks,” in *IEEE International Conference on Microelectronics (ICM’19)*, 2019, p. 94–97.

[32] A. Khanfor, R. Hamadi, H. Ghazzai, Y. Yang, M. R. Haider, and Y. Massoud, “Computational resource allocation for edge computing in social internet-of-things,,” in *IEEE 63rd International Midwest Symposium on Circuits and Systems (MWSCAS)*, 2020, pp. 233–236.

[33] A. Khanfor, H. Ghazzai, Y. Yang, M. R. Haider, and Y. Massoud, “Automated service discovery for social internet-of-things systems,” in *IEEE International Symposium on Circuits and Systems (ISCAS’20)*, 2020, pp. 1–5.

[34] A. M. Kowshalyaa, Xiao-Zhi, Gaob, and V. MLc, “Efficient service search among social internet of things through construction of communities,” *yber-Physical Systems*, vol. 6, no. 1, pp. 33–49, 2020.

[35] P. Wu, L. Pan, and C. Zheng, “Mining set of interested communities with limited exemplar nodes for network based services.”

[36] L. Ma, M. Gong, J. L. Qing, and C. L. Jiao, “Multi-level learning based memetic algorithm for community detection,” *Applied Soft Computing*, vol. 19, pp. 121–133, 2014.

[37] S. Qiao, N. Han, Y. Gao, R.-H. Li, J. Huang, J. Guo, L. A. Gutierrez, and X. Wu, “A fast parallel community discovery model on complex networks through approximate optimization,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 9, pp. 1638–1651, 2018.

[38] Z. Lu, X. Sun, Y. Wen, G. Cao, and T. La Porta, “Algorithms and applications for community detection in weighted networks,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 11, pp. 2916–2926, 2015.

[39] D. Džamić, D. Aloise, and N. Mladenović, “Ascent–descent variable neighborhood decomposition search for community detection by modularity maximization,” *Annals of Operations Research*, vol. 272, no. 1-2, pp. 273–287, 2017.

[40] R. Santiago and L. C. Lamb, “Efficient modularity density heuristics for large graphs,” *European Journal of Operational Research*, vol. 258, no. 3, pp. 844–865, 2017.

[41] L. Ma, Z. S. an; Xiaocong Li, and Q. L. et al., “Influence maximization in complex networks by using evolutionary deep reinforcement learning,” *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. Early Access, pp. 1–15, 2022.

[42] T. Bahreini, H. Badri, and D. Grosu, “An envy-free auction mechanism for resource allocation in edge computing systems,” in *IEEE/ACM Symposium on Edge Computing (SEC)*, 2018, pp. 313–322.

[43] S. Souravlas, A. Sifaleras, and S. Katsavounis, “A parallel algorithm for community detection in social networks, based on path analysis and threaded binary trees,” *IEEE Access*, vol. 7, pp. 20 499–20 519, 2019.

[44] W. J. Gordon and G. F. Newell, “Closed queuing systems with exponential servers,” *Operations Research*, vol. 15, no. 2, pp. 254–265, 1967.

[45] S-Logix. (2017) Last time accessed on Aug 22th, 2022. [Online]. Available: https://slogix.in/source-code/cloudsim-samples/what-are-the-basic-components-and-features-of-cloudsim/

[46] J. Leskovec and A. Krevl. (2014, June) SNAP datasets: Stanford large network dataset collection. University of Stanford. Stanford. [Online]. Available: http://snap.stanford.edu/data

Stavros Souravlas is an Assistant Professor of Computer Architecture at the Department of Applied Informatics, School of Information Sciences, University of Macedonia, where he joined in 2014. His research interests include scheduling of parallel and distributed computing systems, big data stream scheduling, cloud computing, systems modeling and simulation. He has published more than 40 papers in peer reviewed journals and conferences including 6 papers in *IEEE Transactions on Parallel and Distributed Systems, IEEE Transactions on Computational Social Systems and IEEE Access*. He is a member of the IEEE.

Sofia D. Anastasiadou is a Professor of Statistics and Research Methodology at the Department of Midwafery, School of Health Sciences, University of Western Macedonia. Her areas of specialization is Qualitative and Quantitative Methods in Social Sciences, Applied Statistics, Implicative Statistic Analysis, Multivariate Statistical Analysis, Biostatistics, Meta-Analysis, Structural equation models, Analyse des donnees and Big Data.