Trustworthy Distributed Computing on Social Networks

Aziz Mohaisen, *Member, IEEE*, Huy Tran, Abhishek Chandra, *Member, IEEE*, and Yongdae Kim, *Member, IEEE*

Abstract—In this paper we investigate a new computing paradigm, called SocialCloud, in which computing nodes are governed by social ties driven from a bootstrapping trust-possessing social graph. We investigate how this paradigm differs from existing computing paradigms, such as grid computing and the conventional cloud computing paradigms. We show that incentives to adopt this paradigm are intuitive and natural, and security and trust guarantees provided by it are solid. We propose metrics for measuring the utility and advantage of this computing paradigm, and using real-world social graphs and structures of social traces; we investigate the potential of this paradigm for ordinary users. We study several design options and trade-offs, such as scheduling algorithms, centralization, and straggler handling, and show how they affect the utility of the paradigm. Interestingly, we conclude that whereas graphs known in the literature for high trust properties do not serve distributed trusted computing algorithms, such as Sybil defenses—for their weak algorithmic properties, such graphs are good candidates for our paradigm for their self-load-balancing features.

Index Terms—Distributed computing, trust, social computing

1 Introduction

Cloud computing is a new paradigm that overcomes restrictions of conventional computing systems by enabling new technological and economical aspects, such as elasticity and pay-as-you-go—which free users from long-term commitments and obligation towards service providers. Cloud computing is beneficial for both consumers and cloud service providers. Despite many benefits this paradigm provides, it poses several challenges that hinder its usability [2]. Examples of these challenges include the need for architectures to support various potential applications, programming models to address large scale data-centric computing [3], new applications that benefit from the architectural and programming models in the cloud, and the need for strong security and data privacy protection guarantees [4]. Indeed, both outsider and insider threats to security and privacy of data in cloud systems are unlimited [5]. Also, incentives do exist for cloud providers to make use of users' data residing in cloud for their own benefits, for the lack of regulations and enforcing policies [6].

In this paper, we oversee a new computing paradigm, called *SocialCloud*. The described paradigm enjoys parts of the merits provided by the conventional cloud and extends features of other distributed computing paradigms—namely the grid computing. Imagine the scenario of a computing paradigm where users who collectively construct a pool of resources perform computational tasks on behalf of their social acquaintance. Our paradigm and model are similar in many aspects to the conventional grid-computing paradigm. It exhibits such similarities in that users can outsource their computational tasks to peers, complementarily to using friends for storage, which is extensively studied in literature [7], [8]. Our paradigm is, however, very unique in many aspects as well. Most importantly, our paradigm exploits the trust exhibited in social networks as a guarantee for the good behavior of other ''workers'' in the system. Accordingly, the most important component in our paradigm is the social bootstrapping graph, a graph that enables recruiting workers used for performing computation tasks in *SocialCloud*.

In the recent years, social networks and their applications in enabling trust in distributed systems have seen an increasing attention in the research community [9], [10], [11], [12], [13]. Problems that are unsolvable in the cyberspace are becoming solvable using social networks, for that they possess both algorithmic properties—such as connectivity—and trust. These properties are used to reason about the behavior of benign users in the social network, and to limit the misbehavior introduced by malicious users. Also, these properties are the main feature used in the design of social network based systems and to support their efficiency and usability. Most important to the context of *SocialCloud* is the aggregate computational power provided by users who are willing to share their idle time and available compute cycles [14]. In *SocialCloud*, owners of these computing resources are willing to share their computing resources for their friends, and for a different economical model than in the conventional cloud computing paradigm—fully altruistic one. This behavior makes our work share commonalities with an existing stream of work on creating computing services through volunteers

- *A. Mohaisen is with Verisign Labs, Reston, VA 20190, USA. E-mail: amohaisen@verisign.com.*
- *H. Tran and Y. Kim are with Korea Advanced Institute of Science and Technology, Daejeon, South Korea.*
- *A. Chandra is with the University of Minnesota, Minneapolis, MN 55455, USA.*

Manuscript received 30 June 2013; revised 3 Oct. 2013; accepted 16 Nov. 2013. Date of publication 11 Dec. 2013; date of current version 17 Sept. 2014. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TSC.2013.56

[15], [16], although by enabling trust driven from social networks. Our results hence highlight technical aspects of this direction and pose challenges for designs options when using social networks for recruiting such workers and enabling trust. Assumptions in *SocialCloud* are not far-fetched, and the initial results of *SocialCloud* have attracted media attention [17], [18], [19], [20] justifying its potential use and acceptance.

1.1 Contribution

The contribution of this paper is twofold:

- First, we investigate the potential of the social cloud computing paradigm by introducing a design that bootstraps from social graphs to construct distributing computing services. We advocate the merits of this paradigm over existing ones such as the grid computing paradigm.
- Second, we verify the potential of our paradigm using simulation set-up and real-world social graphs with varying social characteristics that reflect different, and possibly contradicting, trust models. Both graphs and the simulator are made public [21] to the community to make use of them, and improve by additional features.

1.2 Organization

The organization of this paper is as follows. In Section 2 we argue for the case of our paradigm. In Section 3 we review the preliminaries of this work. In Section 4, we introduce the main design, including an intensive discussion on the design options. In Section 5, we describe our simulator used for verifying the performance aspects of our design. In Section 6 we introduce the main results and detailed analyses and discussion of the design options, their benefits, and limitations. In Section 7, we summarize some of the related work, including work on using social networks for building trustworthy computing services. In Section 8, we conclude and suggest some of the future work and directions that would be interesting to explore.

2 The Case for SocialCloud

In this paper, we look at the potential of using unstructured social graphs for building distributed computing systems. These systems are proposed with several anticipated benefits in mind. First, such systems would exploit locality of data based on the applications they are intended for, under the assumption that the data would be stored at multiple locations and shared among users represented in the social network—c.f. Section 3.3 and [15] for concrete examples of such applications. This is in fact not a far-fetched assumption. For example, consider a co-authorship social graph, like the one used in our experiments, where the *SocialCloud* is proposed for deployment. In that scenario, data on which computations are to be performed is likely to be at multiple locations; on machines of research collaborators, co-authors, or previous co-authors. Even for some online social networks, the assumption and achieved benefits are not far-fetched as well, considering that friends would have similar interests, and likely to have contents replicated across different machines, which could be potentially of interest to use in our computing paradigm. Examples of such settings include photos taken at parties, videos—for image processing applications, among others.

The second advantage of this paradigm is its trustworthiness. In the recent literature, there has been a lot of interest in the distributed computing community for exploiting social networks to perform trustworthy computations. Examples of these literature works include exploiting social networks for cryptographic signing services [22], Sybil defenses (i.e., addressing attacks in which a single node creates multiple identities and tries to act as if it is multiple nodes to tamper with the normal operation of a distributed system) [11], [23], and routing in many settings including the delay tolerant networks [24], [25]. In all of these cases, along with the algorithmic property in these social networks, the built designs exploit the trust in social networks. The trust in these networks rationalizes the assumption of collaboration in these built system, and the tendency of nodes in the network to act according to the intended protocol with the theorized guarantees. Same as in all of these applications, *SocialCloud* tries to exploit the trust aspect of the social network, and thus it is easy to reason about the behavior of nodes in this paradigm (c.f. Section 3.2).

Related to trust exhibited in the social fabric utilized in our paradigm, the third advantage is that it is also easy to reason about the recruitment of workers. In this context, workers are nodes that are willing to perform computing tasks for other nodes (tasks outsourcers). This feature, when associated with the aforementioned trust, is quite advantageous when compared to the challenge of performing trustworthy computing on dedicated workers in the conventional grid-computing paradigm, where it is hard to recruit such workers. Finally, our design oversees an altruistic model of *SocialCloud*, where nodes participate in the system and do not expect in return. Further details on this model are in Section 3.2.

2.1 Grid vs Cloud Computing

While the *SocialCloud* uses a similar paradigm to that of the grid computing paradigm—in the sense that both try to outsource computations and use high aggregate computational resources, the *SocialCloud* is slightly different. Here we use the comparison features of grid and cloud computing paradigms listed in [4]. In particular, in the *SocialCloud*, there is a pre-defined relationship between the task outsourcer and the computing worker, which does not exist in the grid-computing paradigm. We limit the computations to 1-hop neighbors, which further improve trustworthiness of computations in our model, and proves to be effective as shown in Section 6.5.3. We invasion that our system combines the application oriented paradigm from the grid computing paradigm, and the service computing paradigm from the cloud computing, although at an altruistic economical model supported by the social relationships among computations outsourcers and works. Architecture-wise, *SocialCloud* uses similar structure like grid-computing, by not having a unified infrastructure [4]. We invasion also the programming and application models of *SocialCloud* to be closer to grid computing, in which process-based computing is enabled, rather than cloud computing in which

the unified infrastructure requires dedicated programming models. Security and trustworthiness provided by *SocialCloud* is unique, and exhibits similar characteristics to trusted cloud computing paradigms than the trust used in traditional grid computing.

3 Assumptions and Settings

In this section, we review the preliminaries required for understanding the rest of this paper. In particular, we elaborate on the social networks, their popularity, and their potential for being used as bootstrapping tools for systems, services, and protocols. We describe the social network formulation at a high level, the economical aspect of our system, and finally, the attacker model.

3.1 Social Graphs—High Level Description

In this paper we view the social network as an undirected and unweighted graph $G = (V, E)$, where $V = \{v_1, \ldots, v_n\}$ is the set of vertexes, representing the set of nodes in the social graph, and correspond to users (or computing machines), and $E = \{e_{ij}\}$ (where $1 \leq i \leq n$ and $1 \leq j \leq n$) is the set of edges connecting those vertices—which implies that nodes associated with the social ties are willing to perform computations for each other. $|V| = n$ denotes the size of G and $|E| = m$ denotes the number of edges in G. In the rest of the paper, social network, network, and graph are used interchangeably to refer to both the physical computing network and the underlying bootstrapping social graph, and the meaning depends on the context. Also, we refer to computing entities associated with users in the social network as nodes.

3.2 Economics of SocialCloud

In our design we assume an altruistic model, which simplifies the behavior of users and arguments on the attacker model. In this altruistic model, users in the social network *donate* their *computing resources*—while not using them—to other users in the social network to use them for specific computational tasks. In return, the same users who donated their resources for others would anticipate others as well to perform their computations on behalf of them when needed.

One can further improve this model. Social networks are rich of trust characteristics that capture additional features, and can be used to rationalize this model in several ways. For example, trust in social networks, a well studied vein of research in this context [26], can be used to adjust this model so as users would bind their participation in computations to trust values that they assign to other users. In this work, to make use of and confirm this model, we limit outsourced computations at 1-hop.

3.3 Use Model and Applications

For our paradigm, we envision compute intensive applications for which other systems have been developed in the past using different design principles, but lacking trust features. These systems include ones with resources provided by volunteers, as well as grid-like systems, like in Condor [27], MOON [28], Nebula [16], and SETI@Home [29]. Specific examples of applications built on top of these systems, that would as well fit to our use model, include blog analysis [15], web crawling and social apps (collaborative filtering, image processing, etc) [30], scientific computing [31], among others.

Notice that each of these applications requires certain levels of trust for which social ties are best suited as a trust bootstrapping and enabling tool. Especially, reasoning about the behavior of systems and expected outcomes (in a computing system in particular) would be well-served by this trust model. We notice that this social trust has been previously used as an enabler for privacy in file-sharing systems [32], anonymity in communications systems [33], and collaboration in sybil defenses [23], [26], among others. In this work, we use the same insight to propose a computing paradigm that relies on such trust and volunteered resources, in the form of shared computing time. With that in mind, in the following section we elaborate on the attacker used in our system and trust models provided by our design, thus highlight its advantage and distancing our work from prior works in the literature.

3.4 Attacker Model

In this paper, as it is the case in many other systems built on top of social networks [23], [26], we assume that the attacker is restricted in many aspects. For example, the attacker has a limited capability of creating arbitrarily many edges between himself and other nodes in the social graph.

While this restriction may contradict some recent results in the literature [34]—where it is shown that some social networks are prone to infiltration, it can be relaxed to achieve the intended trust and attack model by considering an overlay of subset of friends of each users. This overlay expresses the trust value of the social graph well and eliminates the influence introduced by the attacker who infiltrated the social graph [26]. For example, since each user decides to which node among neighbors to outsource computations to, each user is aware of other users he knows well and those who are just social encounters that could be potential attackers. Accordingly, the user himself decides whether to include a given node in his overlay or not, thus minimizing or eliminating harm and achieving the required trust and attack model.

The description of the above attacker model might be at odds with the rest of the paper, especially that we use some online social networks that do not reflect characteristics of trust required in our paradigm. However, such networks are used in our demonstration for two reasons. First, to derive insight on the potential of such social networks, and others that share similar topological characteristics, for performing computational tasks according to the method devised in this paper. Second, we use them to illustrate that some of these social networks might be less effective than the trust-possessing social graphs, which we strongly advocate for our computing paradigm.

The restrictions of the adversary model are used when demonstrating the trust-based scheduling described in Section 4.3. In that context, we limit the adversary in the number of edges he can create with honest nodes in the social graph, thus limiting the similarity graph used for characterizing trust as a similarity. When using interactions as an indicator of trust, the adversary is also limited by the

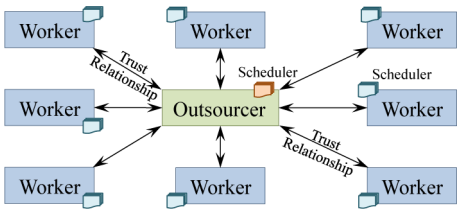


Fig. 1. Depiction of the main *SocialCloud* paradigm as viewed by an outsourcer of computations. The different nodes in the social network act as workers for their friends, who act as potential jobs/tasks outsourcers. The links between social nodes are ideally governed by a strong trust relationship, which is the main source of trust for the constructed computing overlay. Both job outsourcers and workers have their own, and potentially different, schedulers.

number of interactions he can create with honest nodes. Notice that our experimental settings avoid discussing or considering the compromise of social identify and other attacks since addressing them is orthogonal to our work. Furthermore, simple fixes can be deployed to address that issue. For example, identify compromise can be confirmed in an offline manner, or by testing the adversary against the correctness of the returned computation results—by replicating computations on other hosts in case of suspecting such compromise, among other techniques.

3.4.1 Comparison with Trust in Grid Computing Systems

While there has been a lot of research on characterizing and improving trust in the conventional grid computing paradigm [35], [36]—which is the closest paradigm to compare to ours, trust guarantees in such paradigm are less strict than what is expressed by social trust. For that, it is easy to see that some nodes in the grid computing paradigm may act maliciously by, for example, giving wrong computations, or refusing to collaborate; which is even easier to detect and tolerate, as opposed to acting maliciously [37].

4 The Design of SocialCloud

The main design of *SocialCloud* is very simple, where complexities are hidden in design choices and options. In *SocialCloud*, the computing overlay is bootstrapped by the underlying social structure. Accordingly, nodes in the social graph act as workers to their adjacent nodes (i.e., nodes which are one hop away from the outsourcer of computations). An illustration of this design is depicted in Fig. 1. In this design, nodes in the social graph, and those in the *SocialCloud* overlay, use their neighbors to outsource computational tasks to them. For that purpose, they utilize local information to decide on the way they schedule the amount of computations they want each and every one of their neighbors to take care of. Accordingly, each node has a scheduler which she uses for deciding the proportion of tasks that a node wants to outsource to any given worker among her neighbors. Once a task is outsourced to the given worker, and assuming that both data and code for processing the task are transferred to the worker, the worker is left to decide how to schedule the task locally to compute it. Upon completion of a task, the worker sends back the computations result to the outsourcer.

4.1 Design Options: A Scheduling Entity

In *SocialCloud* two schedulers are used. The first scheduler is used for determining the proportion of task outsourced to each worker and the second scheduler is used at each worker to determine how tasks outsourced by outsourcers are computed and in which order. While the latter scheduler can be easily implemented locally without impacting the system complexity, the decision used for whether to centralize or decentralize the former scheduler impacts the complexity and operation of the entire system. In the following, we elaborate on both design decisions, their characteristics, and compare them.

4.1.1 Decentralized Scheduler

In our paradigm, we limit selection of workers to 1-hop from the outsourcer. This makes it possible, and perhaps plausible, to incorporate scheduling of outsourcing tasks at the side of the outsourcer in a decentralized manner—thus each node takes care of scheduling its tasks. On the one hand, this could reduce the complexity of the design by eliminating the scheduling server in a centralized alternative. However, on the other hand, this could increase the complexity of the used protocols and the cost associated with them for exchanging *states*—such as availability of resources, online and offline time, among others. All of such states are exchanged between workers and outsourcers in our paradigm. These states are essential for building basic primitives in any distributed computing system to improve efficiency (see below for further details). An illustration of this design option is shown in Fig. 1. In this scenario, each outsourcer, as well as worker, has its own separate scheduling component.

4.1.2 Centralized Scheduler

Despite the fact that nodes may only require their neighbors to perform the computational tasks on behalf of them and that may require only local information—which could be available to these nodes in advance, the use of a centralized scheduler might be necessitated to reduce communication overhead at the protocol level. For example, to decide upon the best set of nodes to which to outsource computations, a node needs to know which of its neighbors are available, among other statistics. For that purpose, and given that the underlying communication network topology may not necessarily have the same proximity of the social network topology, the protocol among nodes needs to incur back and forth communication cost. One possible solution to the problem is to use a centralized server that maintains states. Instead of communicating directly with neighbor nodes, an outsourcer would request the best set of candidates among its neighbors to the centralized scheduling server. In response, the server will produce a set of candidates, based on the locally stored states. Such candidates would typically be those that would have the most available resources to handle outsourced computation tasks.

An illustration of this design option is shown in Fig. 2. In this design, each node in *SocialCloud* would periodically send states to a centralized server. When needed, an outsourcer node contacts the centralized server to return to it the best set of candidates for outsourcing computations, which the server would return based on the states of these candidates. Notice that only states are returned to the outsourcer, upon which the outsourcer would send tasks to these nodes on its own—Thus, the server involvement is limited to the control protocol.

The communication overhead of this design option to transfer states between a set of d nodes is $2d$, where d messages are required to deliver all nodes' states and d messages are required to deliver states of all other nodes to each node in the set. On the other hand, $d(d-1)$ messages are required in the decentralized option (which requires pairwise communication of states update). When outsourcing of computations is possible among all nodes in the graph, this translates into $O(n)$ for the centralized versus $O(n^2)$ communication overhead for the decentralized option—notice that the communication overhead in the decentralized design would be in reality $O(m)$, where $m \ll n^2$ (for an illustration, see Table 2). To sum up, Table 1 shows a comparison between both options. In the rest of this paper and through the simulation, we use the decentralized setting, accounting for the communication overhead but not requiring trust by depending on any additional entity in the paradigm, not requiring additional hardware, and not suffering from a single point of failure.

4.2 Tasks Scheduling Policy

While using distributed or centralized scheduler resolves scheduling at the outsourcer, two decisions remain untackled: how much computation to outsource to each worker, and how much time a worker should spend on a given task for a certain outsourcer. We address these two issues separately.

Any off-the-shelf scheduling algorithm can be used to schedule tasks at the outsourcer's side, which can be further improved by incorporating trust models for weighted job scheduling [26]. On the other hand, we consider several scheduling algorithms for workers scheduling, as follows. 1) **Round Robin (RR) Scheduling Policy** This is the simplest policy to implement, in which a worker spends an equal share of time on each outsourced task in a round robin fashion among all tasks he has. 2) **Shortest First (SF) Scheduling Policy** The worker performs shortest task first. 3) **Longest First (LF) Scheduling Policy** The worker performs longest task first.

Notice that we omit a lot of details about the underlying computing infrastructure, and abstract such infrastructure to ''time sharing machines'', which further simplifies much of the analysis in this work. However, in a working version of this paradigm, all of these aspects are addressed in a similar manner is in other distributed systems and paradigms. See Section 6 for details on limitations of this approach and possible extensions in the future work.

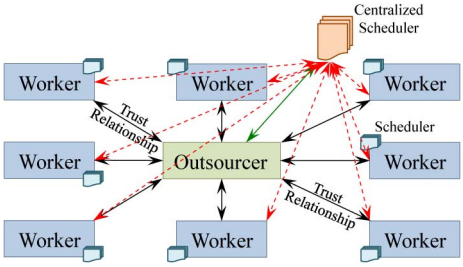


Fig. 2. Centralized versus decentralized model of task scheduling in *SocialCloud*. In the centralized model, an additional centralized entity is used for coordinating to which worker tasks are to be outsourced, whereas the decentralized model does not require this entity and rely on rounds of communication between the outsourcer and workers to coordinate outsourcing of computations.

TABLE 1
Comparison between the Centralized and Decentralized Schedulers. Compared Features are Failure, Communication Overhead, Required Additional Hardware, and Required Additional Trust

Option	Failure	Comm.	HW	Trust
Centralized	✖	$O(n)$	✖	✖
Decentralized	✔	$O(n^2)$	✔	✔

4.3 Trust-Based Scheduling

Both of the scheduling components at the worker and the outsourcer we discussed so far have considered only the availability of a host for performing computations. However, one interesting feature of social networks that can be used to bias the way according to which scheduling is done is the underlying differential trust. Oftentimes, strength of ties between nodes in the social network varies, and that strength can be measured using various ways. In a more realistic context of social network-based computing systems, nodes would capitalize on this differential trust in assigning computations to other nodes. In principle, biasing the scheduling according to the method above is similar to the ''Weighted Differential Scheduler'', where weights of scheduling (at the outsourcer) are assigned on pre-computed trust value.

4.3.1 *Defining Trust-Based Scheduling*

There are several ways used in the literature for computing the strength of ties between social nodes [26]. Two widely used notions for the strength of ties are characterized in the following:

- *Similarity-based*: the similarity-based metric for estimating the strength of ties between two nodes captures the number of common nodes between then. Formally, for two nodes v_i, and v_j, with their neighbors being $N(v_i)$ and $N(v_j)$ respectively, the similarity between node v_i and v_j is defined as $S(v_i, v_j) = \frac{|N(v_i) \cap N(v_j)|}{|N(v_i) \cup N(v_j)|}$ (where $|\cdot|$ means the cardinality of the resulting set $(\cdot)$). Every node v_x in the social graph computes its similarity with the neighboring nodes in the social graph and assigns a trust value to those nodes according to the resulting similarity. Furthermore, when scheduling tasks to be computed on those neighboring nodes, each node uses the similarity value with those neighbors for weighting the distributed portions of tasks.

TABLE 2
Social Graphs Used in Our Experiments

Dataset	# nodes	# edges	Description
DBLP	614981	1155148	CS Co-authorship
Epinion	75877	405739	Friendship network
Physics 2	11204	117649	Co-authorship
Wiki-vote	7066	100736	Voting network
Physics 1	4158	13428	Co-authorship

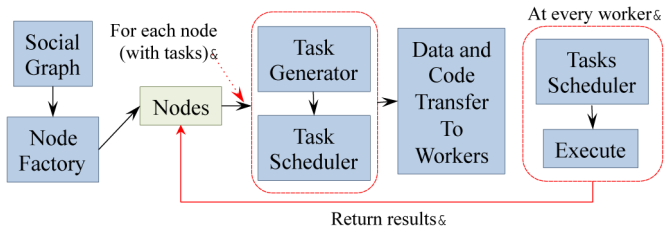


Fig. 3. Flow diagram of *SocialCloud*: social graph is used for bootstrapping the computing service and recruiting workers, nodes are responsible for scheduling their tasks, and each worker uses its local scheduler to divide compute time on neighbors' compute tasks.

- *Interaction-based*: the interaction-based metric for estimating the strength of ties between two nodes captures the volume of interactions between those nodes over time. Let the neighbors of a node v_i be $N(v_i) = \{v_{i1}, \ldots, v_{id}\}$ (where the degree of v_i is d). Let the volumes of interactions between the node v_i and its neighbors listed above be $I_{v_i} = \{I_{v_{i1}}, \ldots, I_{v_{id}}\}$ as observed over a period of time. The node v_i, when having a task to outsource, uses the interaction volumes observed with other nodes as an indicator for trust and weights the amount of computations outsourced to them accordingly. That is, for a node v_{ix} that is a neighbor of v_i, the portion of computations outsourced is $I_{ix} / \sum_{\forall j} I_{ij}$.

4.3.2 Evaluation of Trust-Based Scheduling

While evaluating the performance of *SocialCloud* under those trust-based scheduling policies using the same metric defined in Section 6.1 and used with other policies is straightforward, it is worth noting that both models have a slightly different implication on the underlying graphs, which is worth considering in the experiments. It has been observed in the literature that an adversary that introduces malicious nodes in a social graph (e.g., similar to those created in Sybil attacks, attacks in which a single node creates multiple identities and pretends as if it is multiple nodes; further details on those attacks and defenses are in [38]) still cannot afford to create many meaningful interactions with real social network nodes. Similarly, while it is potentially possible to create a single association with a node in a social network, it is quite harder to do that at scale by creating associations with that node and a large portion of its friends. Accordingly, if that adversary is to be considered in the scheduling system, it will always end up with a small portion of the computations outsourced to it. Even in case where computations are not performed in a timely manner, re-outsourcing them to a more trusted node would not substantially delay the overall computation time (with respect to the evaluation metric explained below for evaluating the performance of *SocialCloud*).

To capture this difference from scheduling policies, we assume that there is a small portion of nodes (p_d) that is controlled by the adversary. Also, given that we lack any information on a ground truth of the volume of interactions of this adversary and other honest nodes, we assume a budget of interactions distributed among all nodes controlled by this adversary and study how that impacts the evaluation metric. We study that as well as the straightforward implication of the time-to-complete evaluation metric (in Section 6.1) without an adversary, but rather with the modified graph according to the trust policies defined above.

4.4 Handling Outliers

The main performance criterion used for evaluating *SocialCloud* is the time required to finish computing tasks for all nodes with tasks in the system. Accordingly, an outlier (also called a computing straggler) is a node with computational tasks that take a long time to finish, thus increasing the overall time to finish and decreasing the performance of the overall system. Detecting outliers in our system is simple: since the total time is given in advance, outliers are nodes with computing tasks that have longer time to finish when other nodes participating in the same outsourced computation are idle. Our method for handling outliers is simple too: when an outlier is detected, we outsource the remaining part of computations on all idle nodes neighboring the original outsourcer. For that, we use the same scheduling policy used by the outsourcer when she first outsourced this task. In the simulation part, we consider both scenarios of handled and unhandled outliers, and observe how they affect the performance of the system.

Notice that our outlier handling requires an estimation of tasks timing to be able to determine the time-to-finish, and decided whether to reschedule a task or not. Alternatively, we can use similar techniques like those used in other distributed computing systems, like Mapreduce: once an outlier is detected as the task that is taking longer than the rest of other tasks to perform, it is rescheduled to other available nodes without modifying it on the original node. Depending on which node finishes the computation first, the outsourcer then sends a signal to kill the computation on the node that did not finish yet.

4.5 Deciding Workers Based on Resources

In real-world deployment of a system like *SocialCloud*, we expect heterogeneity of resources, such as bandwidth, storage, and computing power, in workers. This heterogeneity would result in different results and utilization statistics of a system like *SocialCloud*, depending on which nodes are used for what tasks. While our work does not address this issue, and leaves it as a future work (c.f. Sections 6.6 and 8). We further believe that simple decisions can be made in this regard so as to meet the design goals and achieve the good performance. For example, we expect that nodes would select workers among their social neighbors that have resources and link capacities exceeding a threshold, thus meeting an expected performance outcome.

5 Simulator of SocialCloud

To demonstrate the potential of *SocialCloud* as a computing paradigm, we implement a batch-based simulator [21] that considers a variety of scheduling algorithms, an outlier handling mechanism, job generation handling, and failure simulation.

The flow of the simulator is in Fig. 3. First, a node factory uses the bootstrapping social graph to create nodes and workers. Each node then decides on whether she has a task

or not, and if she has a task she schedules the task according to her scheduling algorithm. If needed, each node then transfers code on which computations are to be performed to the worker along with the chunks of the data for these codes to run on. Each worker then performs the computation according to her scheduling algorithm and returns results to the outsourcer. The implemented components of *SocialCloud* are described in the previous section.

5.1 Timing

In *SocialCloud*, we use *virtual time* to simulate computations and resources sharing. We scale down the simulated time by 3 orders of magnitude of that in reality. This is, for every second worth of computations in real-world, we use one millisecond in the simulation environment. Thus, units of times in the rest of this paper are in virtual seconds.

6 Results and Analysis

In this section, and to derive insight on the potential of *SocialCloud*, we experiment with the simulator described above. Before getting into the details of the experiments, we describe the data and evaluation metric used in this section.

6.1 Evaluation Metric

To demonstrate the potential of operating *SocialCloud*, we use the ''normalized finishing time'' of a task outsourced by a user to other nodes in the *SocialCloud* as the performance metric. We consider the same metric over the different graphs used in the simulation. To demonstrate the performance of all nodes that have tasks to be computed in the system, we use the empirical CDF (commutative distribution function) as an aggregate measure.

For a random variable X, the CDF is defined as $F_X(x) = P_r(X \leq x)$. In our experiments, the CDF measures the fraction (or percent) of nodes that finish their tasks before a point in time x, as part of the overall number of tasks. We define x as the factors of time of normal operation per dedicated machines, if they were to be used instead of outsourcing computations. This is, suppose that the overall time of a task is T_{tot} and the time it takes to compute the subtask by the slowest worker is T_{last}, then x for that node is defined as T_{last}/T_{tot}.

6.2 Tasks Generation

To demonestrate the operation of our simulator and the trade-off our system provides, we consider two different approaches for the tasks generated by each user. The size of each generated task is measured by virtual units of time, and for our demonstration we use two different scenarios. 1) **Constant task weight** each outsourcer generates tasks with an equal size. These tasks are divided into equal shares and distributed among different workers in the computing system. The size of each task is $\bar{T}$. 2) **Variable task weight** each outsourcer has a different task size. We model the size of tasks as a uniformly distributed random variable in the range of $[\bar{T} - \ell, \bar{T} + \ell]$ for some $\bar{T} > \ell$. Each worker receives an equal share of the task from the outsourcer. The generation of a variable task weight would result in non-uniform load among neighbors for tasks to compute, and would be an enabler for policies like shortest (or longest) first and their relative performance.

6.3 Deciding Tasks Outsourcers

Not all nodes in the system are likely to have tasks to outsource for computation at the same time. Accordingly, we denote the fraction of nodes that have tasks to compute by p, where $0 < p < 1$. In our experiments we use p from 0.1 to 0.5 with increments of 0.1. We further consider that each node in the network has a task to compute with probability p, and has no task with probability $1 - p$—thus, whether a node has a task to distribute among its neighbors and compute or not follows a binomial distribution with a parameter p. Once a node is determined to be among nodes with tasks at the current round of run of the simulator, we fix the task length. For tasks length, we use both scenarios mentioned in Section 6.2; with fixed or constant and variable tasks weights.

6.4 Social Graphs

To derive insight on the potential of *SocialCloud*, we run our simulator on several social graphs with different size and density, as shown in Table 2. The graphs used in these experiments represent three co-authorship social structures (DBLP, Physics 1, and Physics 2), one voting network (of Wiki-vote for wikipedia administrators election), and one friendship network (of the consumer review website, Epinion). Notice the varying density of these graphs, which also reflects on varying topological characteristics. Also, notice the nature of these social graphs, where they are built in different social contexts and possess varying qualities of trust that fits to the application scenario mentioned earlier. The proposed architectural design of SocialClould, however, minimally depends on these graphs, and other networks can brought instead of them. As these graphs are widely used for verifying other applications on social networks, we believe they enjoy a set of representative characteristics to other networks as well.

6.5 Main Results

In this section we demonstrate the performance of *SocialCloud* and discuss the main results of this work under various circumstances. Due to the lack of space, we delegate additional results to the technical report in [39]. For all measurements, our metric of performance and comparison is the normalized time to finish metric, as explained in Section 6.1. We note that all of the experiments, unless otherwise is mentioned, do not consider an adversary in place. We consider the adversarial model described in Section 3.4 we describing the results of trust-based scheduling.

6.5.1 Number of Outsourcers

In the first experiment, we run our *SocialCloud* simulator on the different social graphs discussed earlier to measure the evaluation metric when the number of the outsourcers of tasks increases. We consider $p = 0.1$ to 0.5 with increments of 0.1 at each time. The results of this experiment are in Fig. 4. On the results of this experiment we make several observations.

First, we observe the potential of *SocialCloud*, even when the number of outsourcers of computations in the social

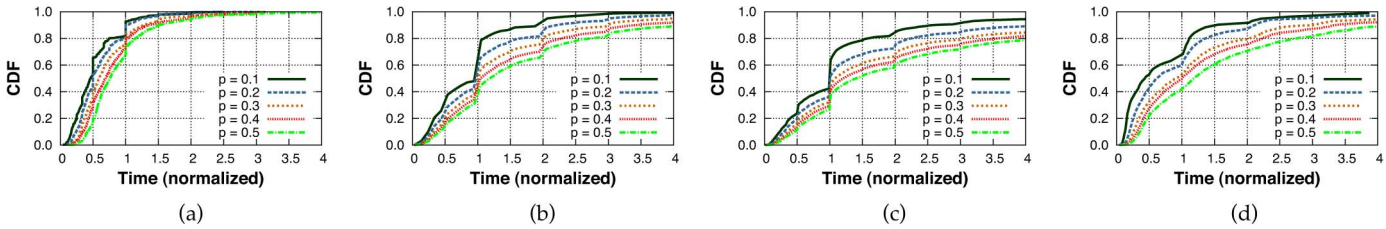


Fig. 4. Normalized time it takes to perform outsourced computations in *SocialCloud*. Different graphs with different social characteristics have different performance results, where those with well-defined social structures have self-load-balancing features, in general. These measurements are taken with round-robin scheduling algorithm that uses the outlier handling policy in Section 4.4 for a fixed task size (of 1000 simulation time units). (a) Physics 1. (b) DBLP. (c) Epinion. (d) Wiki-vote.

network is as high as 50 percent of the total number of nodes, which translates into a small normalized time to finish even in the worst performing social graphs (about 60 percent of all nodes with tasks would finish in 2 normalized time units). However, this advantage varies for different graphs: we observe that sparse graphs, like co-authorship graphs, generally outperform other graphs used in the experiments (by observing the tendency in the performance in Figs. 4a and 4b versus Figs. 4c and 4d). In the aforementioned graphs, for example, we see that when 10 percent of nodes in each case is used, and by fixing x, the normalized time, to 1, the difference of performance is about 30 percent. This difference of performance can be observed by comparing the Physics co-authorship graphs—where 95 percent of nodes finish their computations—and the Epinion graph—where only about 65 percent of nodes finish their computations at the same time.

Second, we observe that the impact of p, the fraction of nodes with tasks in the system, would depend greatly on the underlying graph rather than p alone. For example, in Fig. 4a, we observe that moving from $p = 0.1$ to $p = 0.5$ (when $x = 1$) leads to a decrease in the fraction of nodes that finish their computations from 95 percent to about 75 percent. On the other hand, for the same settings, this would lead to a decrease from about 80 percent to 40 percent, a decrease from about 65 percent to 30 percent, and a decrease from 70 percent to 30 percent in DBLP, Epinion, and Wiki-vote, respectively. This suggests that the decreases in the performance are due to an inherit property of each graph. The inherit property of each graph and how it affects the performance of *SocialCloud* is further illustrated in Fig. 5. We find that even when DBLP's size is two orders of magnitude the size of Wiki-vote, it outperforms Wiki-vote when not using outlier handling, and gives almost the same performance when using it.

6.5.2 Scheduling Policy

Now, we turn our attention to understanding the impact of the different scheduling policies discussed in Section 4.2 on the performance of *SocialCloud*. We consider the different datasets, and use $p = 0.1$ to 0.5 with 0.2 increments (the results are shown in Fig. 6). The observed consistent pattern in almost all figures in this experiment tells that shortest first policy always outperforms the round robin scheduling policy, whereas the round robin scheduling policy outperforms the longest first. This pattern is consistent regardless of p and the outlier handling policy. The difference in the performance when using different policies can be as low as 2 percent (when $p = 0.1$ in physics co-authorship; shown in Fig. 6l) and as high as 70 percent (when using $p = 0.5$ and outlier handling as in wiki-vote (Fig. 6o)). The patterns are clearer in Fig. 6 by observing combinations of parameters and policies.

One possible intuitive explanation of this behavior is understood by the fairness aspects the various policies provide. While the shortest-first policy finishes shorter tasks first, thus it is more likely to yield a set of accumulated tasks that count toward the CDF as the time progresses, the longest-first policy does the contrary by having less numbers of finished tasks when dedicating resources for those that take the longest. On the other hand, the round-robin policy establishes a middle point, by mixing longer and shorter tasks in its processing, and yielding a mix of them in the ones finished.

6.5.3 Evaluation of Trust-Based Scheduling

Now we turn our attention to the trust-based scheduling described in Section 4.3, and how they affect the performance of *SocialCloud*. We use the settings described in Section 4.3.2 for evaluating the two scheduling policies described in Section 4.3.1. In particular, for the similarity-based scheduling, we assume a fixed number of nodes under the control of the adversary.

To simulate the adversary model described in Section 4.3.2, we assume each node has a variable degree: we quantize the degree distribution of the original graph into a set of brackets (fixed to 10) and randomly select the degree of a

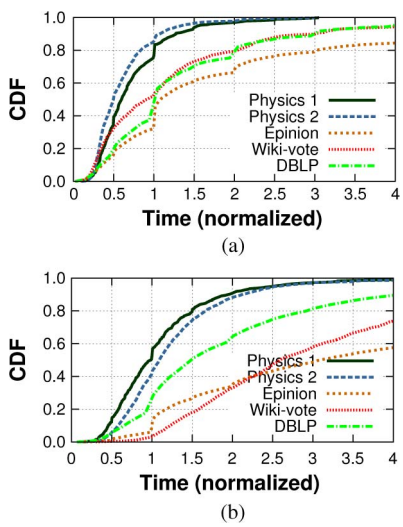


Fig. 5. Performance of *SocialCloud* on the different social graphs. These plots demonstrate the inherent differences in these social graphs. Both figures use $p = 0.3$ and the round robin scheduling algorithm. (a) Handled outliers. (b) Unhandled outliers.

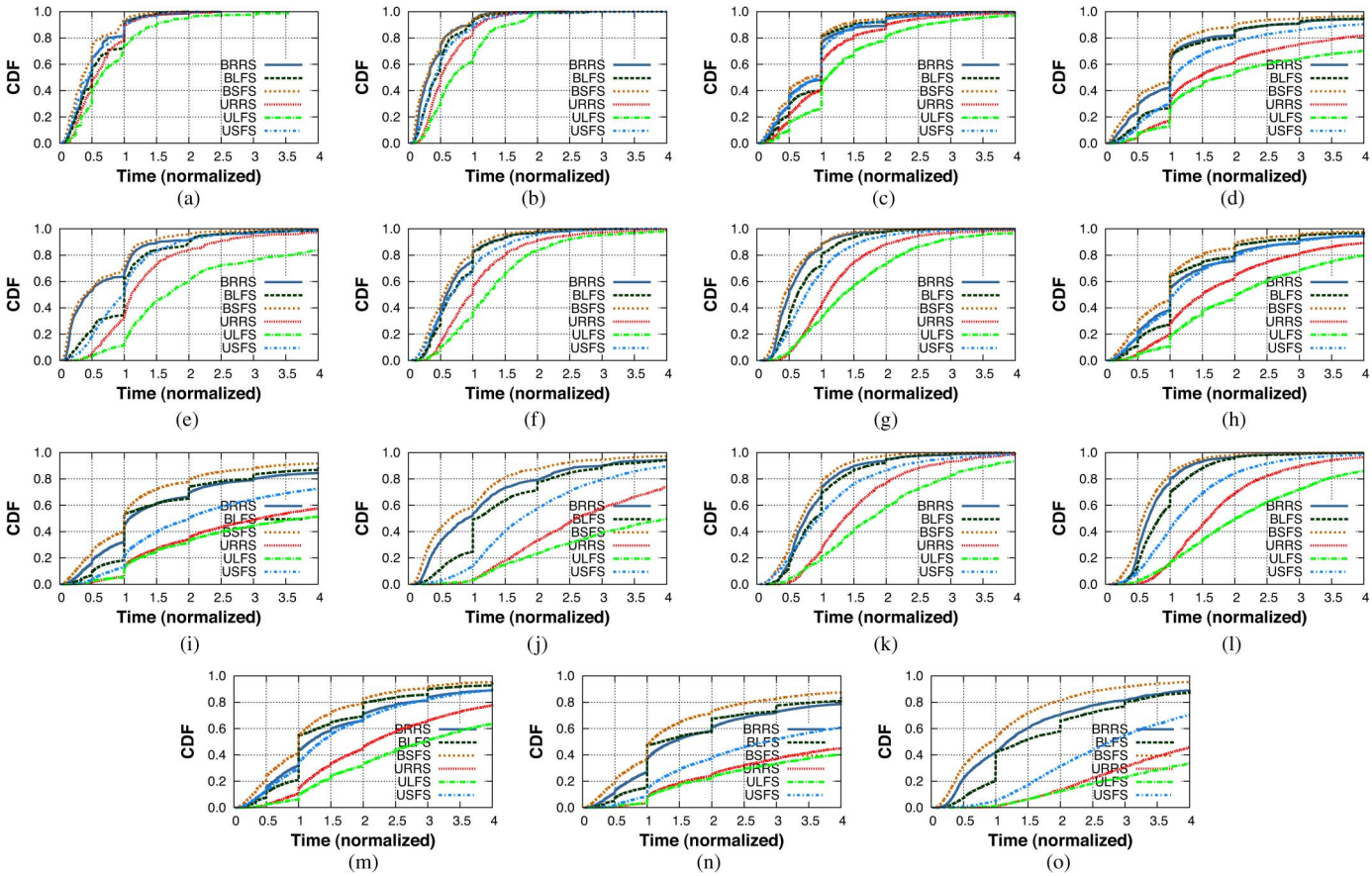


Fig. 6. Normalized time it takes to perform outsourced computations in *SocialCloud* for different scheduling policies. Naming convention: U stands for unhandled outlier and B stands for handled outliers (Balanced). RRS, SFS, and LFS stand for round-robin, shortest first, and longest first scheduling. (a) Physics 1 $(p = 0.1)$. (b) Physics 2 $(p = 0.1)$. (c) DBLP $(p = 0.1)$. (d) Epinion $(p = 0.1)$. (e) Wiki-vote $(p = 0.1)$. (f) Physics 1 $(p = 0.3)$. (g) Physics 2 $(p = 0.3)$. (h) DBLP $(p = 0.3)$. (i) Epinion $(p = 0.3)$. (j) Wiki-vote $(p = 0.3)$. (k) Physics 1 $(p = 0.5)$. (l) Physics 2 $(p = 0.5)$. (m) DBLP $(p = 0.5)$. (n) Epinion $(p = 0.5)$. (o) Wiki-vote $(p = 0.5)$.

portion of the adversary nodes to fall within that bracket. This portion of malicious nodes is proportional to the number of honest nodes falling in that bracket. Assuming a budget of interactions, we then uniformly distribute that budget on all edges controlled by the adversary. We note that this scenario is optimal for the adversary and worst for our system. For the similarity-based model, and to limit the similarity score, we connect each of those nodes to a random node in the graph, using one of its edges that contribute to its degree.

We assign the number of interactions the adversary is capable of generating as ten times the maximum number of interactions associated with an honest user in the graph. We note that meaningful interactions are hard to forge, and such simulated settings are pessimistic and correspond to a powerful adversary.

For the evaluation of these policies, we use the interaction social graph of Facebook from [40]. The final graph consists of 35,665 nodes, with 86,525 weighted edges. The weights on the edges correspond to the interactions. When using the graph for similarity, weights are omitted. The adversary is capable of plugging 1000 malicious nodes (roughly 2.8 percent of the total nodes) in the graph in both of the similarity and interaction-based models. The budget of interactions associated with the attacker is 20,000. The average node degree for the adversary is calculated and found to be 3.2, slightly more than the average degree of an honest node. The average weight on an edge controlled by the adversary is found to be 6.25, roughly a quarter of the average weight on an edge between two honest nodes. The similarity is computed as the Jaccard index [26], which is also described in Section 4.3.1.

The proportion of outsourced computations depends on the perceived trust by a node towards other nodes based on weights attributed to interaction and similarity. We assume that the adversary does not return any computation results to the outsourcer, and the outsourcer uses the outlier handling policies to perform the computations upon not hearing back, thus treating the outcomes as a failure of performing computations. The same technique is used to recover from failure due to malicious activities when not using trust models. We compare the outcomes of the trust-based policies to the unweighted graph scenario where the adversary and honest neighbors are treated equally. We use the same metric described in Section 6.1.

Fig. 7 shows the outcomes of this experiment. We notice that in both cases where the trust-based scheduling is used, *SocialCloud* operates well by outperforming the plain scenario where no trust is used. For example, we notice that while only 75 percent of the compute tasks are finished for a normalized time of 1.5 when not deploying any trust model, about 92 percent and 95 percent of tasks are finished with the

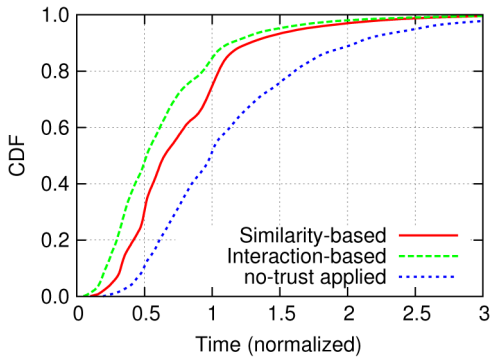


Fig. 7. Trust affects the performance of *SocialCloud*. Both of the similarity- and interaction-based models outperforms the plain model, where no trust is used (parameters: round-robin scheduling at workers, $p = 0.2$, and variable task length of mean equal 1000 units.)

similarity- and interaction-based models, respectively. The intuition behind this boost in the performance is simple: whereas the plain setting (where no trust is applied) treats neighboring nodes equally, and tasks of equal size are likely to be outsourced to a malicious neighbor—thus worsening the overall time for finishing tasks, the trust-based models described above punishes nodes with less trust. Given that both interaction and similarity are not easy to manipulate, according to the settings described earlier, the weight of the tasks outsourced to the adversary are generally small, and once the outsourcer realizes they are not completed it will take shorter to finish them using the outlier handling policy.

6.5.4 Performance with Outliers Handling

Outliers, as defined in Section 4.4, drag the performance of the entire system down. However, as pointed out earlier, handling outliers is quite simple in *SocialCloud* if accurate timing is used in the system. Providing such timing is important in understanding the time-to-finish portion and establish whether rescheduling a task is needed or not. Here we consider the impact of the outlier handling policy explained in Section 4.4. The impact of using the outlier handling policy can be also seen on Fig. 6, which is used for demonstrating the impact of using different scheduling policies as well. In this figure, we see that the simple handling policy we proposed improves the performance of the system greatly in all cases. The improvement differs depending on other parameters, such as p, and the scheduling policy. As with the scheduling policy, the improvement can be as low as 2 percent and as high as more than 60 percent. When p is large, the potential for improvement is high—see, for example, $p = 5$ in Physics 2 with the round robin scheduling policy where almost 65 percent improvement is due to outlier handling when $x = 1$.

6.5.5 Variable Task Size

In all of the above experiments, we considered computational tasks of fixed size; 1000 of virtual time units in each of them. Whether the same pattern would be observed in tasks with variable size is unclear. Here we experimentally address this concern by using variable duty size that is uniformly distributed in the interval of [500, 1500] time units. The results are shown in Fig. 8. Comparing these results to the middle row of Fig. 6 (for the fixed size tasks), we make two observations. 1) While the average task size in both scenarios is same, we observe that the performance with variable task size is worse. This performance is anticipated as our measure of performance is the time to finish that would be definitely increased as some tasks with longer time to finish are added. 2) The same patterns advantaging a given scheduling policy on another are maintained as in earlier with fixed task length.

6.5.6 Structure and Performance

We note that the performance of *SocialCloud* is quite related to the underlying structure of the social graph. We see that sparse graphs, like co-authorship graphs—which are pointed out in [26] to be slow mixing graphs—have performance advantage in *SocialCloud*. These graphs, in particular, are shown to possess a nice trust value that can be further utilized for *SocialCloud*. Furthermore, this trust value is unlikely to be found in online social networks which are prone to infiltration, making the case for trust-possessing graphs even stronger, as they achieve performance guarantees as well. This, indeed, is an interesting finding by itself, since it shows contradicting outcomes to what is known in the literature on the usefulness of these graphs—see Section 3 for more details and the work in [26] for prior literature that agrees with our findings.

6.6 Additional Features and Limitations

Our simulator of *SocialCloud* omits a few details concerning the way a distributed system behaves in reality. In particular, our measurements do not report on or experiment with failure. However, our simulator is equipped with functionality for handling failure in the same way used for handling outliers (c.f. Section 4.4). Furthermore, our simulator considers a simplistic scenario of study by abstracting the hardware infrastructure, and does not consider additional resources consumed, such as memory and I/O resources. In the future, we will consider equipping our simulator with such functionalities and see how this affects the behavior and benefits of *SocialCloud*.

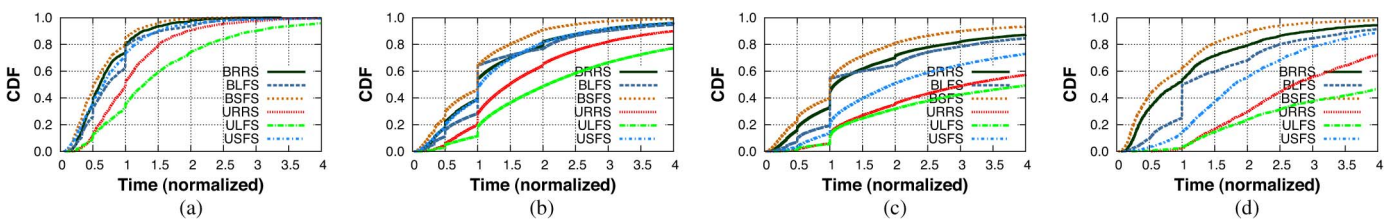


Fig. 8. Normalized time to perform outsourced computations in *SocialCloud*, for variable task size. (a) Physics 1 $(p = 0.3)$. (b) DBLP $(p = 0.3)$. (c) Epinion $(p = 0.3)$. (d) Wiki-vote $(p = 0.3)$.

For simplicity, we do not consider the heterogeneity of resources, such as bandwidth and resources, in nodes acting as workers in the system. Furthermore, we did not consider how this affects the usability of our system and what decision choices this particular aspect of distributed computing systems would have on the utility of our paradigm. While this would be mainly a future work to consider (c.f. Section 8), we expect that nodes would select workers among their social neighbors that have resources and link capacities exceeding a threshold, thus meeting an expected performance outcome.

7 Related Work

There have been many papers on using social networks for building communication and security systems. Below we highlight a few examples of these efforts and works.

Systems built on top of social networks include file sharing [32], [7], [41], [26], anonymous communication [42], [33], Sybil defenses [11], [23], [43], [44], [10], [26], routing [24], [25], [45], referral and filtering [46], content distribution [47], [48], and live streaming systems [49], among many others [50]. Most of these systems use social networks' trust and connectivity for their operation.

Concurrent to our work, and following their work in [51], Chard *et al.* [13] suggested the use of social networks to build a resource sharing system. Whereas their main realization was still a social storage system as in [51], they also suggested that the same vision can be used to build a distributed computing service as we advocate in this work. Recent realizations of this vision have been reported in [52] and [53]. In [52], Thaufeeg *et al.* devised an architecture where ''individuals or institutions contribute the capacity of their computing resources by means of virtual machines leased through the social network''. In [53] Koshy *et al.* further explored the motivations of users to enable social cloud systems for scientific computing. Caton *et al.* explored foundations of trust in social cloud computing environments [54]. Engineering incentives for social cloud have been studied in [55] and additional scientific applications based on a social network governed computing nodes are explored in [56].

With a similar flavor of distributed computing services design, there has been prior works in literature on using volunteers' resources for computations exploiting locality of data [16], [15], examination of programing paradigms, like MapReduce [57] on such paradigm [28]. Finally, our work shares several commonalities with grid and volunteer computing systems [27], [28], [16], [15], [29], [58], of which many aspects are explored in the literature. Trust of grid computing and volunteer-based systems is explored in [35], [36]. Applications built on top of these systems, that would fit to our use model, are reported in [15], [31], among others.

8 Summary and Future Work

In this section we summarize our work and conclude with directions that we would like to investigate as a future work.

8.1 Summary

In this paper we have introduced the design of *SocialCloud*, a distributed computing service that recruits computing workers from friends in social networks and use such social networks that characterize trust relationships to bootstrap trust in the proposed computing service. We further advocated the case of such computing paradigm for the several advantages it provides. To demonstrate the potential of our proposed design, we used several real-world social graphs to bootstrap the proposed service and demonstrated that majority of nodes in most cases would benefit computationally from outsourcing their computations to such service. We considered several basic distributed system characteristics and features, such as outlier handling, scheduling decisions, and scheduler design, and show advantages in each of these features and options when used in our system. To the best of our knowledge, this is the first and only work in literature that bases such design of computing paradigm on volunteers recruited from social networks and tries to bring the trust factor from these networks and use it in such systems. This characteristic distances our work from the prior work in literature that uses volunteers' resources for computations [16], [15].

Most important outcome of this study, along with the proposed design, is two findings: the relationship exposed between the social graphs and the behavior of the built computing service on top of them, and the way trust models impact the performance of *SocialCloud*. In particular, we have shown that social graphs that possess strong trust characteristics as evidenced by face-to-face interaction [26], which are known in the literature for their poor characteristics prohibiting their use in applications (such as Sybil defenses [11], [23]), have a self-load-balancing characteristics when the number of outsourcers are relatively small (say 10 to 20 percent of the overall population on nodes in the computing services). That is, the time it takes to finish tasks originated by a given fraction of nodes in such graph, and for the majority of these nodes, ends in a relatively short time. On the other hand, such characteristics and advantages are maintained even when the number of outsourcers of computations is as high as 50 percent of the nodes, contrary to the case of other graphs with dense structure and high connectivity known to be proper for the aforementioned applications. This last observation encourages us to investigate further scenarios of deployment of our design. We anticipate interesting findings based on the inherit structure of such deployment contexts—since such contexts may have different social structures that would affect the utility of the built computing overlay.

8.2 Future Work

This paper opens several future directions and call for further investigation at several fronts. To this end, in the future we will look to extend this work in directions: completing the missing design components in a simulation fashion and realizing a working real-world deployment and application.

In the first direction, we aim to complete the missing ingredient of the simulator and enrich it by further scenarios of deployment of our design, under failure, with various scheduling algorithms at both sides of the outsourcer and workers (in addition to those discussed in

this work). We also would like to consider other overhead characteristics that might not be in line with topological characteristics in the social graph. These characteristics may include the uptime, downtime, communication overhead, and I/O overhead consumption, among others.

In the second direction, we will turn our attention from the simulation settings to real-world deployment settings, thus addressing options discussed in Section 6.6. We will also implement a proof-of-concept application, among those discussed in Section 3.3, by utilizing design options discussed in this paper. We anticipate hidden complexities in the design to arise, and significant findings to come out of the deployment.

ACKNOWLEDGMENT

The work of the last author was funded by the MSIP(Ministry of Science, ICT & Future Planning), Korea in the ICT R&D Program 2013. An earlier version of this work appeared inproceedings of 8th ACM Symposium on Information, Computer and Communications Security [11].

REFERENCES

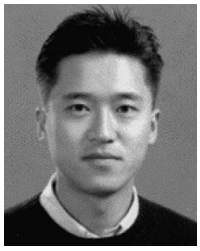
[1] A. Mohaisen, H. Tran, A. Chandra, and Y. Kim, ''Trustworthy Distributed Computing on Social Networks,'' in *Proc. ACM ASIACCS*, 2013, pp. 155-160.
[2] T.S. Dillon, C. Wu, and E. Chang, ''Cloud Computing: Issues and Challenges,'' in *Proc. IEEE AINA*, 2010, pp. 27-33.
[3] M. Esler, J. Hightower, T. Anderson, and G. Borriello, ''Next Century Challenges: Data-Centric Networking for Invisible Computing: The Portolano Project at the University of Washington,'' in *Proc. ACM MobiCom*, 1999, pp. 256-262.
[4] I. Foster, Y. Zhao, I. Raicu, and S. Lu, ''Cloud Computing and Grid Computing 360-Degree Compared,'' in *Proc. Grid Comput. Environ. Workshop*, 2008, pp. 1-10.
[5] Y. Chen, V. Paxson, and R.H. Katz, ''What's New About Cloud Computing Security?,'' Univ. California Berkeley, Berkeley, CA, USA, Tech. Rep. UCB/EECS-2010-5, Jan. 2010.
[6] S. Subashini and V. Kavitha, ''A Survey on Security Issues in Service Delivery Models of Cloud Computing,'' *J. Netw. Comput. Appl.*, vol. 34, no. 1, pp. 1-11, Jan. 2011.
[7] D. Tran, F. Chiang, and J. Li, ''Friendstore: Cooperative Online Backup Using Trusted Nodes,'' in *Proc. SNS*, 2008, pp. 37-42.
[8] M.D. Dikaiakos, D. Katsaros, P. Mehra, G. Pallis, and A. Vakali, ''Cloud Computing: Distributed Internet Computing for it and Scientific Research,'' *IEEE Internet Comput.*, vol. 13, no. 5, pp. 10-13, Sept./Oct. 2009.
[9] H. Yu, M. Kaminsky, P.B. Gibbons, and A. Flaxman, ''Sybilguard: Defending Against Sybil Attacks Via Social Networks,'' in *Proc. ACM SIGCOMM*, 2006, pp. 267-278.
[10] P. Mittal, M. Caesar, and N. Borisov, ''X-Vine: Secure and Pseudonymous Routing Using Social Networks,'' in *Proc. NDSS*, San Diego, CA, USA, 2012, pp. 1-15.
[11] G. Danezis and P. Mittal, ''Sybilinfer: Detecting Sybil Nodes Using Social Networks,'' in *Proc. NDSS*, 2009, pp. 1-15.
[12] A. Mohaisen, D.F. Kune, E. Vasserman, M. Kim, and Y. Kim, ''Secure Encounter-Based Mobile Social Networks: Requirements, Designs, and Tradeoffs,'' *IEEE Trans. Dependable Secure Comput.*, vol. 10, no. 6, pp. 380-393, Nov./Dec. 2013.
[13] K. Chard, K. Bubendorfer, S. Caton, and O. Rana, ''Social Cloud Computing: A Vision for Socially Motivated Resource Sharing,'' *IEEE Serv. Comput.*, vol. 5, no. 4, pp. 551-563, June 2011.
[14] L. Barroso and U. Holzle, ''The Case for Energy-Proportional Computing,'' *Computer*, vol. 40, no. 12, pp. 33-37, Dec. 2007.
[15] J.B. Weissman, P. Sundarrajan, A. Gupta, M. Ryden, R. Nair, and A. Chandra, ''Early Experience with the Distributed Nebula Cloud,'' in *Proc. ACM DIDC*, 2011, pp. 17-26.
[16] A. Chandra and J. Weissman, ''Nebulas: Using Distributed Voluntary Resources to Build Clouds,'' in *Proc. HotCloud*, 2010, pp. 1-5.
[17] Distributed Computing Services From Social Networks, Dec. 2011. [Online]. Available: http://bit.ly/vZvYCnmsn.com
[18] The VergeResearchers Testing a Social-Based Distributed Computing Network, Dec. 2011. [Online]. Available: http://bit.ly/sfqeo2
[19] Data NewsGrid Computing Met je Vrienden, Dec. 2011. [Online]. Available: http://bit.ly/wBatuV
[20] MIT Technology ReviewThe Imminent Rise of Social Cloud Computing, Dec. 2011. [Online]. Available: http://bit.ly/x5rY3w
[21] A. Mohaisen, H. Tran, A. Chandra, and Y. Kim, *SocialCloud*. [Online]. Available: http://bit.ly/193Z6cv 2013
[22] S. Xu, X. Li, and T.P. Parker, ''Exploiting Social Networks for Threshold Signing: Attack-Resilience vs. Availability,'' in *Proc. ACM ASIACCS*, 2008, pp. 325-336.
[23] H. Yu, P.B. Gibbons, M. Kaminsky, and F. Xiao, ''Sybillimit: A Near-Optimal Social Network Defense Against Sybil Attacks,'' in *Proc. IEEE SP*, 2008, pp. 3-17.
[24] G. Bigwood and T. Henderson, ''Social DTN Routing,'' in *Proc. ACM CoNEXT*, 2008, pp. 1-2.
[25] E.M. Daly and M. Haahr, ''Social Network Analysis for Routing in Disconnected Delay-Tolerant MANETs,'' in *Proc. ACM MobiHoc*, 2007, pp. 32-40.
[26] A. Mohaisen, N. Hopper, and Y. Kim, ''Keep Your Friends Close: Incorporating Trust into Social Network-Based Sybil Defenses,'' in *Proc. IEEE INFOCOM*, 2011, pp. 1943-1951.
[27] M. Litzkow, M. Livny, and M. Mutka, ''Condor-A Hunter of Idle Workstations,'' in *Proc. ICDCS*, 1988, pp. 104-111.
[28] H. Lin, X. Ma, J. Archuleta, W.-C. Feng, M. Gardner, and Z. Zhang, ''Moon: Mapreduce on Opportunistic Environments,'' in *Proc. HPDC*, 2010, pp. 95-106.
[29] D. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer, ''Seti@home: An Experiment in Public-Resource Computing,'' *Commun. ACM*, vol. 45, no. 11, pp. 56-61, Nov. 2002.
[30] M. Cardosa, C. Wang, A. Nangia, A. Chandra, and J. Weissman, ''Exploring Mapreduce Efficiency with Highly-Distributed Data,'' in *Proc. ACM MapReduce*, 2011, pp. 27-34.
[31] L. Wang, J. Tao, M. Kunze, A. Castellanos, D. Kramer, and W. Karl, ''Scientific Cloud Computing: Early Definition and Experience,'' in *Proc. IEEE HPCC*, 2008, pp. 825-830.
[32] T. Isdal, M. Piatek, A. Krishnamurthy, and T. Anderson, ''Privacy-Preserving P2P Data Sharing With Oneswarm,'' *ACM SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 4, pp. 111-122, Oct. 2010.
[33] S. Nagaraja, ''Anonymity in the Wild: Mixes on Unstructured Networks,'' in *Proc. PETS*, 2007, pp. 254-271.
[34] L. Bilge, T. Strufe, D. Balzarotti, and E. Kirda, ''All Your Contacts are Belong to us: Automated Identity Theft Attacks on Social Networks,'' in *Proc. WWW*, 2009, pp. 551-560.
[35] F. Azzedin and M. Maheswaran, ''Towards Trust-Aware Resource Management in Grid Computing Systems,'' in *Proc. IEEE/ACM CCGRID*, 2002, pp. 1-6.
[36] S. Kamvar, M. Schlosser, and H. Garcia-Molina, ''The Eigentrust Algorithm for Reputation Management in P2P Networks,'' in *Proc. WWW*, 2003, pp. 640-651.
[37] A. Chakrabarti, *Grid Computing Security*. Berlin, Germany: Springer-Verlag, 2007.
[38] A. Mohaisen, A. Yun, and Y. Kim, ''Measuring the Mixing Time of Social Graphs,'' in *Proc. IMC*, 2010, pp. 383-389.
[39] A. Mohaisen, H. Tran, A. Chandra, and Y. Kim, ''Socialcloud: Distributed Computing on Social Networks,'' Univ. Minnesota, Minneapolis, MN, USA, Tech. Rep., 2011.
[40] A. Mohaisen and Y. Kim, ''Dynamix: Anonymity on Dynamic Social Structures,'' in *Proc. ACM ASIACCS*, 2013, pp. 167-172.
[41] B. Viswanath, M. Mondal, K.P. Gummadi, A. Mislove, and A. Post, ''Canal: Scaling Social Network-Based Sybil Tolerance Schemes,'' in *Proc. 7th EuroSys Conf. Comput.*, Bern, Switzerland, Apr. 2012, pp. 309-322.
[42] E. Vasserman, R. Jansen, J. Tyra, N. Hopper, and Y. Kim, ''Membership-Concealing Overlay Networks,'' in *Proc. ACM CCS*, 2009, pp. 390-399.
[43] P. Mittal, M. Wright, and N. Borisov, ''Pisces: Anonymous Communication Using Social Networks,'' in *Proc. NDSS*, 2013, pp. 1-18.
[44] A. Mohaisen, H. Tran, N. Hopper, and Y. Kim, ''On the Mixing Time of Directed Social Graphs and Security Implications,'' in *Proc. ASIACCS*, 2012, pp. 36-37.
[45] J. Davitz, J. Yu, S. Basu, D. Gutelius, and A. Harris, ''ILINK: Search and Routing in Social Networks,'' in *Proc. ACM KDD*, 2007, pp. 931-940.
[46] H.A. Kautz, B. Selman, and M.A. Shah, ''Referral Web: Combining Social Networks and Collaborative Filtering,'' *Commun. ACM*, vol. 40, no. 3, pp. 63-65, Mar. 1997.

[47] F. Zhou, L. Zhang, E. Franco, A. Mislove, R. Revis, and R. Sundaram, ''WebCloud: Recruiting Social Network Users to Assist in Content Distribution,'' in *Proc. 11th IEEE Int'l Symp. NCA*, Cambridge, MA, USA, Aug. 2012, pp. 10-19.
[48] K. Chard, S. Caton, O. Rana, and D.S. Katz, ''A Social Content Delivery Network for Scientific Cooperation; Vision, Design, and Architecture,'' in *Proc. DataCloud*, 2012, pp. 1058-1067.
[49] W. Lin, H. Zhao, and K. Liu, ''Incentive Cooperation Strategies for Peer-to-Peer Live Multimedia Streaming Social Networks,'' *IEEE Trans. Multimedia*, vol. 11, no. 3, pp. 396-412, Apr. 2009.
[50] P. Mittal, C. Papamanthou, and D. Song, ''Preserving Link Privacy in Social Network Based Systems,'' in *Proc. NDSS*, San Diego, CA, USA, 2013, pp. 1-16.
[51] K. Chard, S. Caton, O. Rana, and K. Bubendorfer, ''Social Cloud: Cloud Computing in Social Networks,'' in *Proc. IEEE CLOUD*, 2010, pp. 99-106.
[52] A.M. Thaufeeg, K. Bubendorfer, and K. Chard, ''Collaborative Eresearch in a Social Cloud,'' in *Proc. IEEE eScience*, 2011, pp. 224-231.
[53] J. Koshy, K. Bubendorfer, and K. Chard, ''A Social Cloud for Public Eresearch,'' in *Proc. IEEE eScience*, 2011, pp. 363-370.
[54] S. Caton, C. Dukat, T. Grenz, C. Haas, M. Pfadenhauer, and C. Weinhardt, ''Foundations of Trust: Contextualising Trust in Social Clouds,'' in *Proc. 2nd Int'l Conf. CGC*, 2012, pp. 424-429.
[55] C. Haas, S. Caton, and C. Weinhardt, ''Engineering Incentives in Social Clouds,'' in *Proc. 11th IEEE/ACM Int'l Symp. CCGrid*, 2011, pp. 572-575.
[56] W. Wu, H. Zhang, and Z. Li, ''Open Social Based Collaborative Science Gateways,'' in *Proc. 11th IEEE/ACM Int'l Symp. CCGrid*, 2011, pp. 554-559.
[57] J. Dean and S. Ghemawat, ''Mapreduce: A Flexible Data Processing Tool,'' *Commun. ACM*, vol. 53, no. 1, pp. 72-77, Jan. 2010.
[58] H. Wang, F. Wang, J. Liu, and J. Groen, ''Measurement and Utilization of Customer-Provided Resources for Cloud Computing,'' in *Proc. IEEE INFOCOM*, A.G. Greenberg and K. Sohraby, Eds., 2012, pp. 442-450.

Aziz Mohaisen received the MS and PhD degrees in computer science from the University of Minnesota, Minneapolis, MN, USA, both in 2012. In 2012, he joined Verisign Labs where he is currently a Research Scientist. Before pursuing graduate studies at Minnesota, he was a Member of Engineering Staff at the Electronics and Telecommunication Research Institute, a large research and development institute in South Korea. His research interests are in the areas of networked systems, systems security, data privacy, and measurements. He is a member of the IEEE.

Huy Tran received the BS degree in computer science from the University of Minnesota, Minneapolis, MN, USA, in 2013, where he spent the last semester of his study at Korea Advanced Institute of Science and Technology in Daejeon, South Korea. He is a Software Engineer at Microsoft Corp. His research interests include complex networks analysis, security, and privacy.

Abhishek Chandra received the BTech degree in computer science and engineering from IIT Kanpur, India, and the MS and PhD degrees in computer science from the University of Massachusetts Amherst, Amherst, MA, USA. He is an Associate Professor in the Department of Computer Science and Engineering at the University of Minnesota. His research interests are in the areas of Operating Systems and Distributed Systems, with current focus on performance and resource management in large-scale systems such as Clouds, Grids, and Data centers. He is a recipient of the NSF CAREER Award and IBM Faculty Award, his PhD dissertation was nominated for the ACM Dissertation Award, and he has been a co-author on two Best Paper/Student Paper Awards. He is a member of the IEEE.



Yongdae Kim received the PhD degree from the computer science department at the University of Southern California, Los Angeles, CA, USA, under the guidance of Gene Tsudik. He is a Professor in the Department of Electrical Engineering at KAIST. Prior to joining KAIST, he was a faculty member in the Department of Computer Science and Engineering at the University of Minnesota—Twin Cities. He received NSF career award and McKnight Land-Grant Professorship Award from University of Minnesota, in 2005. Currently, he is serving as a steering committee member of NDSS (Network and Distributed System Security Symposium). His current research interests include security issues in various systems such as cyber physical systems, mobile/ad hoc/sensor/cellular networks, social networks, storage systems, and anonymous communication systems. He is a member of the IEEE.