SocialCloud: Using Social Networks for Building Distributed Computing Services

Abedelaziz Mohaisen
University of Minnesota
Minneapolis, MN 55455
mohaisen@cs.umn.edu

Huy Tran
University of Minnesota
Minneapolis, MN 55455
huy@cs.umn.edu

Abhishek Chandra
University of Minnesota
Minneapolis, MN 55455
chandra@cs.umn.edu

Yongdae Kim
University of Minnesota
Minneapolis, MN 55455
kyd@cs.umn.edu

***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, Security, Trust, Social Computing, Performance.**

I. Introduction

Cloud computing is a new paradigm of computing that overcomes the restriction of conventional computing paradigms 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. While it meets customers and users technological demands, the cloud computing paradigm is also a rich field of profit to cloud providers [3].

For users, cloud computing overcomes several shortcomings as opposed to using conventional computing paradigms; where the used infrastructure and software are owned by the user. For example, cloud computing enables users of the cloud—who also can be providers of services—to virtually locate their contents closers to their consumers and reduce latency of serving such contents, a challenging issue in conventional computing settings. Also, considering the return on investment, cloud computing has its appealing economical benefits and incentives, which make it a desirable option to many users. These incentives can be seen in the long run as a reduced overall cost resulting from hardware and software liabilities and maintenance costs in alternative paradigms [3]. As for providers, benefits are also economical in the absolute sense.

The current conventional cloud computing paradigm has many benefits, despite posing several challenging issues that need to be addressed before wider adoption by many potential users [22]. Examples of these issues include the need for concrete and clear business model that outlines clearer service level agreements (SLA) and guarantees the rights of users [29], [28], [15], the need for architectures that consider the variety of potential applications demanded by users, the need for programming models that consider the large scale of data in the cloud, and the need for new applications that benefit from the architectural and programming models in the cloud, among other issues. While many of these issues are being constantly addressed in ongoing research efforts; where several architectures [19], [9], [51], programming models [21], [16], [47], and applications [43], [54], [27], [51], [10], [29], [28] are proposed, security and data privacy are chief among other issues to be considered before this paradigm is widely accepted. Indeed, both outsider and insider threats to security and privacy of data in cloud systems are unlimited. 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.

In this paper, we oversee a new type of computing paradigm, called SocialCloud, that enjoys parts of the merits provided by the conventional cloud. 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. 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 ingredient to our paradigm is the social bootstrapping graph, a graph that is used for recruiting workers for a social network.

Indeed, social networks are very popular (c.f. §III-A). This popularity of social networks has opened the door wide for investigating the potential of these networks for many applications. Problems that are unsolvable in the cyberspace are easily solvable using social networks, for that they possess both algorithmic properties—such as connectivity—and trust, which are used to reason about the behavior of honest users in the social network, and limit the misbehavior introduced by other malicious users supported by efficiency features. Most important to the context of our paradigm is the aggregate computational power of nodes in the social network. Indeed, beyond the nodes and social links, the social networks consist of users with computing machines that are idle for most of the time [6]. Furthermore, owners of these computing machines 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 [53], [14]. 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.

A. Contributions

To this end, our contribution in this paper is mainly 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 [40] to the community to make use of them, and improve by additional features.

B. Organization

The organization of this paper is as follows. In §II we argue for the case of our paradigm. In §III we review the preliminaries of this work. In §IV, we introduce the main design, including an intensive discussion on the design options. In §V, we describe our simulator used for verifying the performance aspects of our design. In §VI we introduce the main results and detailed analyses and discussion of the design options, their benefits, and limitations. In §VII, we summarize some of the related work, including work on using social networks for building trustworthy computing services. In §VIII, we conclude and suggest some of the future work and directions that would be interesting to explore.

II. 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—see §III-D and [53] 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 [55], Sybil defenses [58], [18], [57], and routing in many settings including the delay tolerant networks [7], [17]. 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. §III-C).

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 §III-C.

Grid 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. 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.

III. 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.

A. Social Networks and Systems Bootstrapping

Social networks are so popular. Nine of the twenty most popular sites on the web are for social networking [24]. The top ten online social networking websites have more than 650 million of unique visitors per month in total. The most popular social network, Facebook [25] alone serves 250 million unique visitors per month, with more than 96 unique visitors per second. Such popularity of social networks has motivated so many designs, protocols, and applications on top of social networks. Examples include routing [7], [17], [20], [37], social gossip [1], [26], [12], and Sybil defenses [58] (c.f. §VII). While they are different in the details of their operation, all of these designs and protocols weigh algorithmic properties (connectivity), trust, and collaboration in the underlying social networks, which are used for bootstrapping such systems.

B. 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.

C. 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 [38], 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, in order to make use of and confirm this model, we limit outsourced computations at 1-hop.

While we do not consider that in this paper, another model using interests and groups is worth mentioning for its popularity and potential as a future work. The incentives model can be further relaxed by enabling "interest" based model of computation where workers do computation to other nodes in the graph that only share some interest with them. This interest can be publicly identified by the membership of a node in a group. Investigating this model is left as a future work.

D. 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; where trust is needed in such applications and provided by our paradigm. These systems include ones with resources provided by volunteers, as well as grid-like systems, like in Condor [36], MOON [34], Nebula [14], [53], and SETI@Home [2].

Specific examples of applications built on top of these systems, that would as well fit to our use model, include blog analysis [53], web crawling and social-network applications (collaborative filtering, image processing, etc) [11], scientific computing [52], 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 [30], anonymity in communications systems [42], and collaboration in sybil defenses [33], [57], [38], 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.

E. Attacker Model

In this paper, as it is the case in many other systems built on top of social networks [57], [58], [49], 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 [8]—where it is shown that some legitimate users befriend random users in the social network who are potentially attackers, it can be relaxed to achieved 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 [38]. For example, since each user decides on to which node among his adjacent nodes 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, when used, are used 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.

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 [4], [5], [46], [31]—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 [13].

IV. 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 Figure 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.

A. Design Options: Scheduling Entity

In the 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.

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 Figure 1. In this scenario, each outsourcer, as well as worker, has its own separate scheduling component.

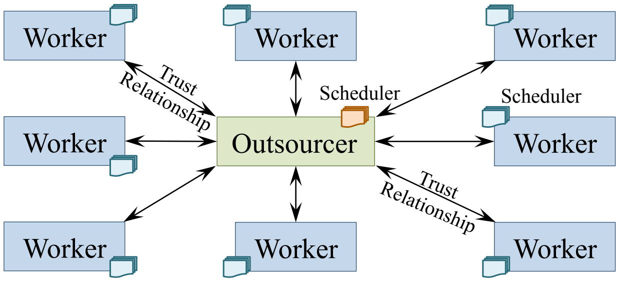


Fig. 1. A depiction of the main SocialCloud paradigm as viewed by an outsourcer of computations. The different nodes in the social network act as workers for their friends, who act as potential jobs/tasks outsourcers. The links between social nodes are ideally governed by a strong trust relationship,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

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, in order 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 of the different nodes. 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 the outsourced computation task.

An illustration of this design option is shown in Figure 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.

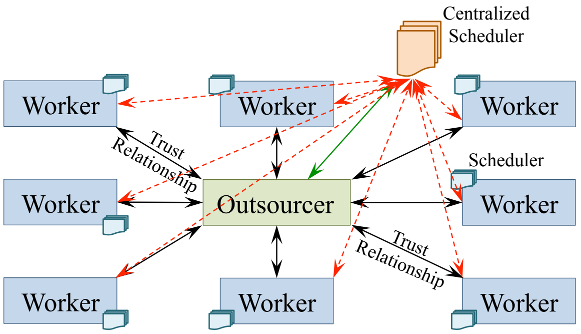


Fig. 2. The decentralized model of task scheduling in SOCIALCLOUD.

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. To sum up, Table I shows a comparison between both options.

TABLE I
A COMPARISON BETWEEN THE CENTRALIZED AND DECENTRALIZED SCHEDULER OPTIONS. COMPARED FEATURES ARE RESISTANCE TO FAILURE, COMMUNICATION OVERHEAD, REQUIRED ADDITIONAL HARDWARE, AND REQUIRED ADDITIONAL TRUST.

Option	Failure	Communication	Hardware	Trust
Centralized	✖	$O(n)$	✖	✖
Decentralized	✔	$O(n^2)$	✔	✔

B. Tasks Scheduling Policy

While the use of distributed or centralized scheduling entity resolves the issue of scheduling at the outsourcer side, two decisions remain unsolved: how much computation to outsource to each node (worker), and how much time a node among these workers should spend on a given task for a certain outsourcer. We handle these two issues separately.

As mentioned earlier, any off-the-shelf scheduling algorithm can be utilized to decide the right scheduling policy at the side of the outsourcer, which can be further improved by incorporating trust characterization models for weighted job scheduling [38]. On the other hand, for workers scheduling, we consider several scheduling options as follows (notice that all of these policies are applied with respect to "computing time". This further requires estimating the time required for each task as a first step for using these policies).

- **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.
- **Shortest First (SF) Scheduling Policy.** The worker performs shortest task first.
- **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. In the results, we experiment with the three scheduling policies.

C. 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.

D. 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. §VI-F and §VIII). 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.

V. SIMULATOR OF SOCIALCLOUD

To demonstrate the potential of SOCIALCLOUD as a computing paradigm, we implement a batch-based simulator [40] that considers a variety of scheduling algorithms, an outlier handling mechanism, job generation handling, and failure simulation. A flow diagram of the simulator is in Figure 3.

The flow of the simulator, which represents the flow of the system, is depicted in Figure 3. First, the node factory uses the bootstrapping social graph to create nodes and their workers. Each node then decides on whether she has a task or not, and if she has a task she schedule 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 splits of the data for these codes to run on. Each worker then performs the computation according to the scheduling algorithm of the worker and returns the results of the computations to the outsourcer.

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.

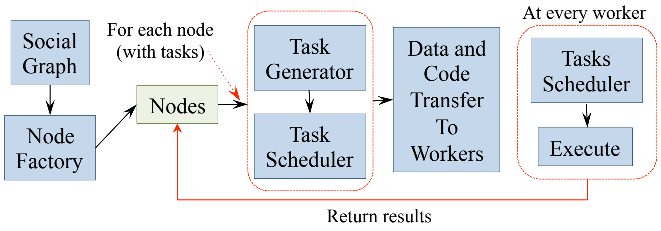


Fig. 3. The flow diagram of SocialCloud: social graph is used for bootstrapping the computing service and recruit workers, nodes are responsible for scheduling their tasks by determining the amount of work each of its neighbors would process, and each worker (node) uses its local scheduler to determine how much time is allowed for each sub-task by its neighbors.

VI. Results and Analysis

In this section, in order 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.

A. 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 for the population 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}.

B. Tasks Generation

Also for demonstrating the operation of our simulator, and the trade-off that such operation 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:

- **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 $\bar{T}$.
- **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.

C. Deciding Tasks Outsourcers

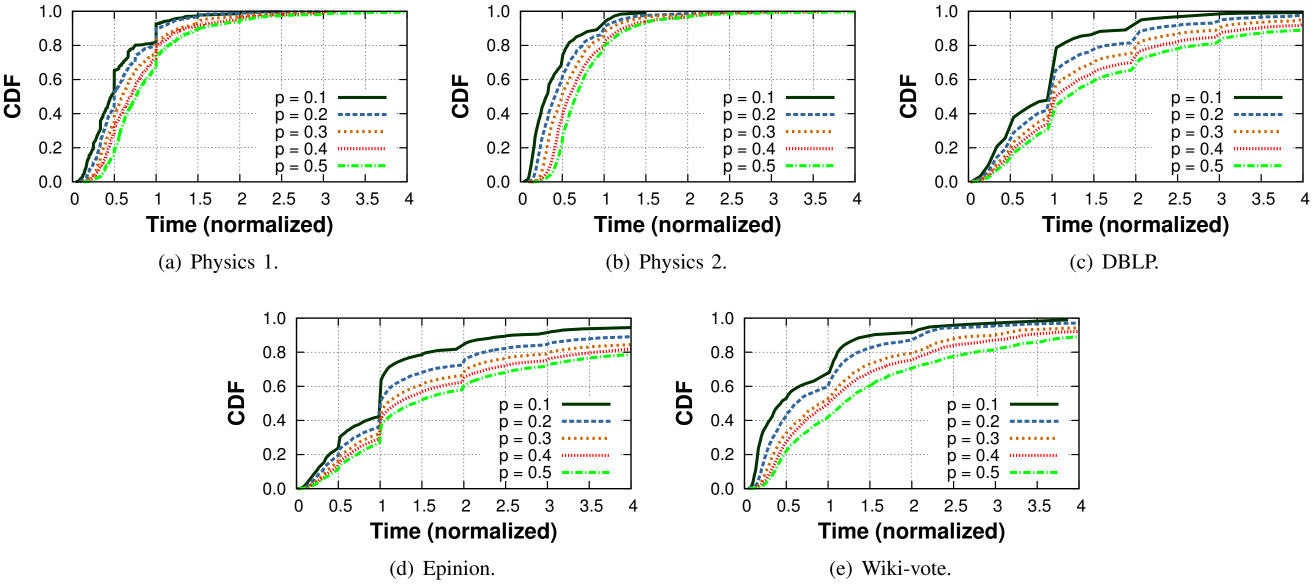
Not all nodes in the system are likely to have tasks to outsource for computation at the same time. Accordingly, we denote the fraction of nodes that have tasks to compute by p, where $0 < p < 1$. In our experiments we use p from 0.1 to 0.5 with increments of 0.1. We further consider that each node in 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 §VI-B; with fixed or constant and variable tasks weights.

D. 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 II. 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). All of these graphs are made undirected, if they are not already, which rationalizes their use in our system. 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 [38].

E. Main Results

In this section we demonstrate our paradigm and discuss the main results of this work. 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, explained in section VI-A.



(a) Physics 1.

(b) Physics 2.

(c) DBLP.

(d) Epinion.

(e) Wiki-vote.

Fig. 4. The 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 §IV-C for a fixed task size (of 1000 simulation time units).

TABLE II
SOCIAL GRAPHS USED IN OUR EXPERIMENTS.

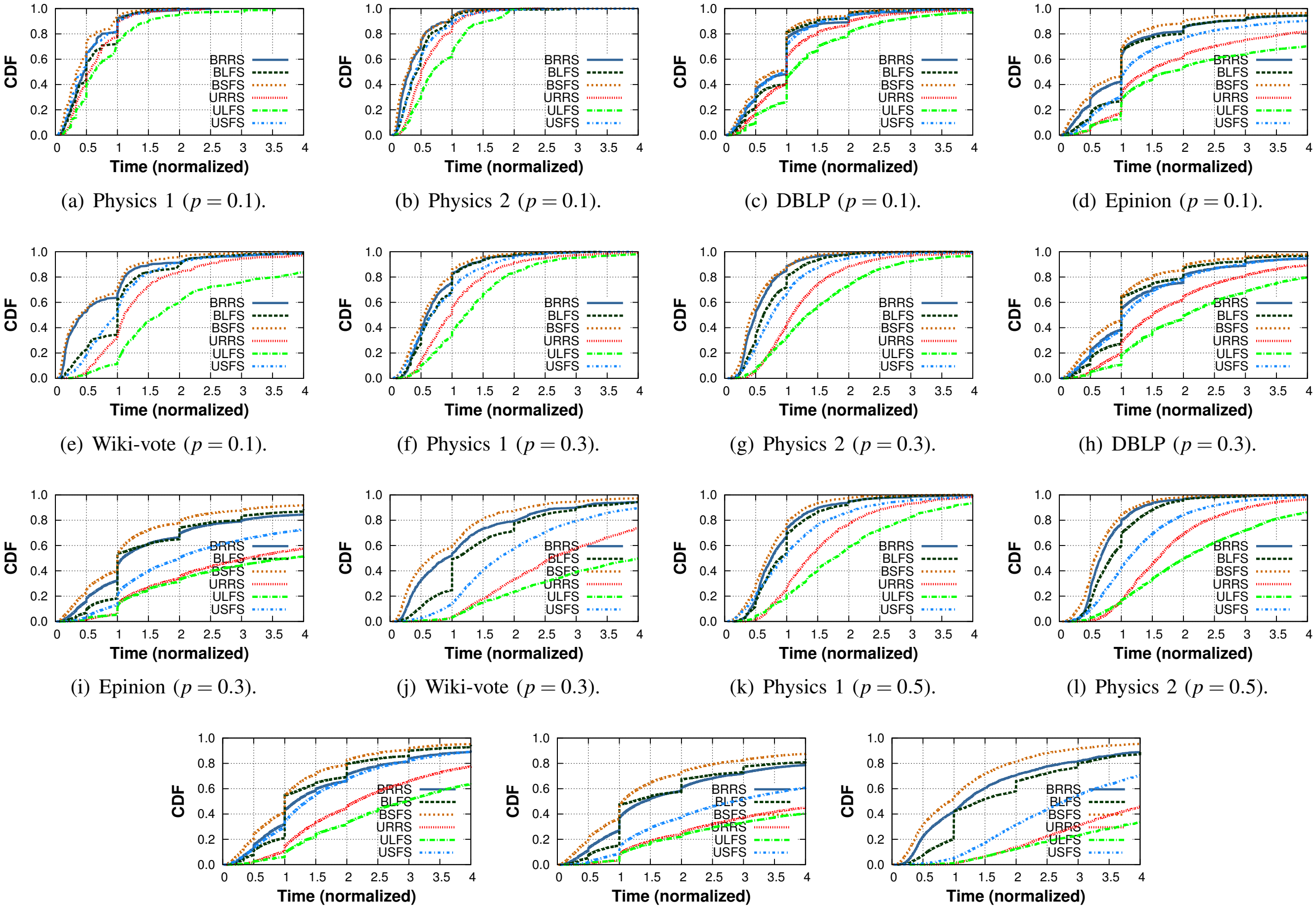
Dataset	# nodes	# edges	Description
DBLP	614981	1155148	CS Co-authorship
Epinion	75877	405739	Friendship network
Physics 2	11204	117649	Co-authorship
Wiki-vote	7066	100736	Voting network
Physics 1	4158	13428	Co-authorship

1) Performance When Varying the 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 Figure 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 network is as high as 50% of the total number of nodes, which translates into a small normalized time to finish even in the worst performing social graphs (about 60% 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 figures 5(b) through 4(c) versus figures 4(d) and 4(e)). In the aforementioned graphs, for example, we see that when 10% of nodes in each case is used, and by fixing x, the normalized time, to 1, the difference of performance is about 30%. This difference of performance is observed between the Physics co-authorship graphs—where 95% of nodes finish their computations—and the Epinion graph—where only about 65% of nodes finish their computations.

Second, we observe that the impact of p, the fraction of nodes with tasks in the system, would depend on the graph rather than p alone. For example, in Figure 5(b), 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% to about 75%. On the other hand, for the same settings, this would lead to a decrease from about 80% to 40%, a decrease from about 65% to 30%, and a decrease from 70% to 30% 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 Figure 5. Interestingly, we find that even if DBLP is almost two orders of magnitude the size of Wiki-vote, for example, it outperforms Wiki-vote when not using outlier handling, and gives almost the same performance when using outliers handling.

2) Performance with different scheduling policies: Now, we turn our attention to understanding the impact of the different scheduling policies discussed in §IV-B 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 Figure 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% (when $p = 0.1$ in physics co-authorship; shown in figure 7(b)) and as high as



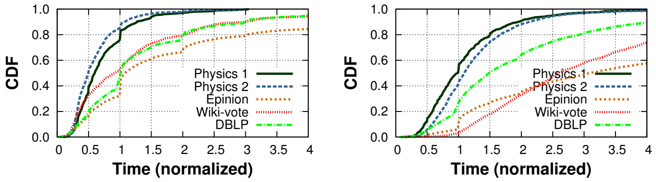
(a) Physics 1 ($p = 0.1$). (b) Physics 2 ($p = 0.1$). (c) DBLP ($p = 0.1$). (d) Epinion ($p = 0.1$).

(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 ($p = 0.5$).

Fig. 6. The 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) Handled outliers (b) Unhandled outliers

Fig. 5. The performance of SocialCloud on the different social graphs used for our experiments, demonstrating the inherent differences in the different social graphs. Both figures use $p = 0.3$ and the round robin scheduling algorithm. Left figure is when handling outliers, whereas the right figure without handling the outliers.

70% (when using $p = 0.5$ and outlier handling as in wiki-vote (figure 6(o))). The patterns are made clearer in Figure 6 by observing combinations of parameters and policies.

3) Performance with Outliers Handling: Outliers, as defined in §IV-C, 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. Here we consider the impact of the outlier handling policy explained in §IV-C. The impact of using the outlier handling policy can be also seen on figure 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% and as high as more than 60%. 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% improvement is due to outlier handling when $x = 1$.

4) Performance with 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 Figure 7. Comparing these results to the middle row of Figure 6 (for the fixed size tasks), we make two observations. (i) 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. (ii) The same patterns advantaging a given scheduling policy on another are maintained as in earlier with fixed task length.

5) Relationship Between Structure and Performance: It is worth noting that the performance of SOCIALCLOUD is quite related to the underlying structure of the social graph. For example, sparse graphs such as co-authorship graphs—which are pointed out in [38] to be slow mixing graphs—are the graphs with 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 opposite outcomes to what is known in the literature on the usefulness of these graphs—see §III and more details, see [38].

F. Additional Features and Limitations of Experiments

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. §IV-C). 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.

One last concern related to our demonstration of our paradigm is that 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. §VIII), 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.

VII. RELATED WORK

There have been many papers on the use of social networks for building communication and security systems, studying the performance of such designs on top of social networks, and analyzing the assumptions used in these designs as well. Below we highlight a few examples of these efforts and works.

Systems built on top of social networks include file sharing systems [30], anonymous communication systems [50], [42] Sybil defenses [18], [33], [56], [58], referral and filtering systems [32], [44], and live streaming [35]. Most of these applications weigh the trust in social graph, and an algorithmic property that makes the operation of these systems on top of social network effective. Another set of applications that exploit social networks' trust is routing [7], [17], [20], [37]—in several settings, where it has been shown that connectivity in social graphs can be of benefit in disconnected networks. Finally, assumptions of social network-based systems are explored recently, where Sybil defenses and their assumptions are studied in [41], and trust is challenged in [38].

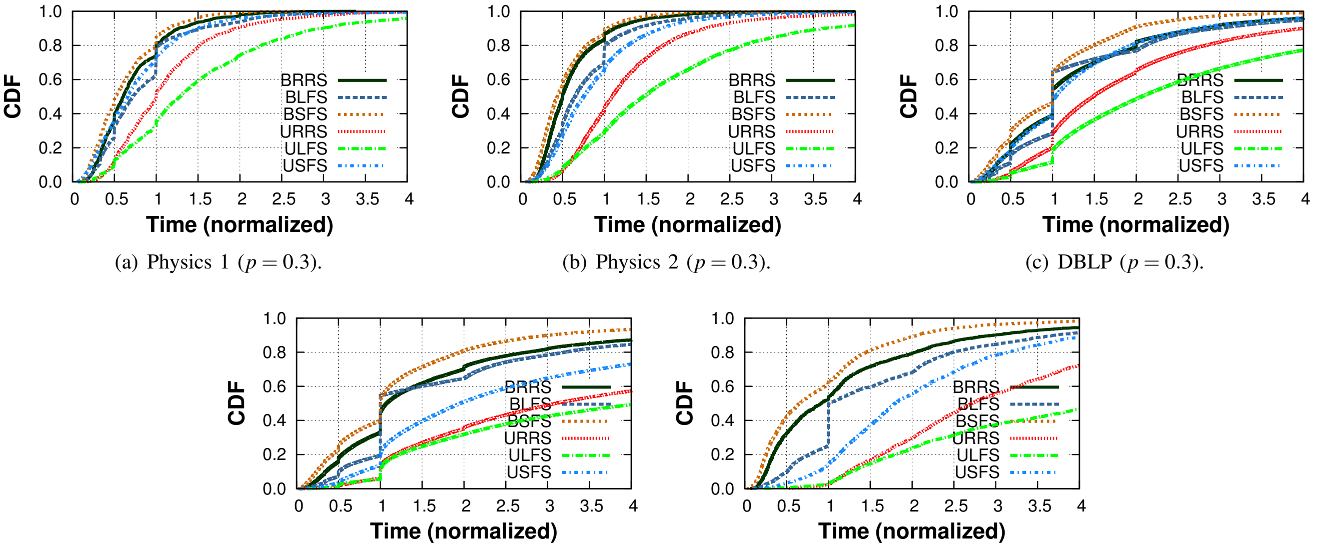
Perhaps the closest vein of related work in the literature to our work is on the use of social networks for building computing services. Until the time of writing this work, most of the prior research work has been solely focused on providing storage services, but not a platform of computations. Such storage services use slightly different economical model from SOCIALCLOUD's model, where payment per Megabyte per month rates are used as opposed to our eco-system. Examples of such efforts are reported by Sato [45] and Tran et al. [48]). Xu et al. [55] have further explored a first step in the direction of building cloud computing platforms on top of social networks where by considering the access control model in this domain with preferred access control guarantees. The results of this work can be used as a building block in our work to improve the quality of access control and authorization.

With similar flavor of distributed computing services design, there has been prior works in literature on using volunteers' resources for computations exploiting locality of data [14], [53], examination of programing paradigms, like MapReduce [21] on such paradigm [34], [11]. Finally, our work shares several commonalities with the grid and volunteer computing systems [36], [34], [14], [53], [2], of which many aspects are explored in the literature. Trust of grid computing and volunteer-based systems is explored in [4], [5], [46], [31], [23]. Applications built on top of these systems, that would fit to our use model, are reported in [53], [11], [52], among others.

VIII. SUMMARY AND FUTURE WORK

A. 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



(a) Physics 1 ($p = 0.3$).

(b) Physics 2 ($p = 0.3$).

(c) DBLP ($p = 0.3$).

(d) Epinion ($p = 0.3$).

(e) Wiki-vote ($p = 0.3$).

Fig. 7. The normalized time it takes to perform outsourced computations in SOCIALCLOUD, for variable task size.

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 [14], [53].

Most important outcome of this study, along with the proposed design, is the relationship exposed between the social graphs and the behavior of the built computing service on top of them. In particular, we have shown that social graphs that possess strong trust characteristics as evidenced by face-to-face interaction [38], which are known in the literature for their poor characteristics prohibiting their use in applications (such as Sybil defenses [18], [57], [58]), 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% 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.

B. Future Work

In the future we will look at two directions. 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 different scheduling algorithms at both sides of the outsourcer and workers (in addition to those discussed in this work), and 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. One interesting feature that we will consider is trust-based scheduling, benefiting from the prior work in [38].

In the second direction, we will turn our attention from the simulation settings to real-world deployment settings, thus addressing options discussed in §VI-F, and to implement a proof-of-concept application, among those discussed in §III-D, by utilizing design options discussed in this paper. We anticipate a lot of hidden complexities in the design to arise, and significant findings to come out of the deployment that we will report on in the future work.

ACKNOWLEDGEMENT

This work is supported by a research grant from Korea Advanced Institute of Technology (KAIST). A. Mohaisen is partly supported by a Doctoral Dissertation Fellowship of the graduate school of the University of Minnesota and H. Tran is partly UROP (undergraduate research opportunity program of the University of Minnesota).

REFERENCES

[1] S. M. A. Abbas, J. A. Pouwelse, D. H. J. Epema, and H. J. Sips, "A gossip-based distributed social networking system," in *Proc. of WETICE*, 2009, pp. 93–98.

[2] D. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer, "Seti@ home: an experiment in public-resource computing," *Communications of the ACM*, vol. 45, no. 11, pp. 56–61, 2002.

[3] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *CACM*, vol. 53, no. 4, pp. 50–58, 2010.

[4] F. Azzedin and M. Maheswaran, "Evolving and managing trust in grid computing systems," in *IEEE Canadian Conference on Electrical and Computer Engineering*, vol. 3. IEEE, 2002, pp. 1424–1429.

[5] ——, "Towards trust-aware resource management in grid computing systems," in *Proc. of CCGRID*. IEEE, 2002, pp. 452–452.

[6] L. Barroso and U. Holzle, "The case for energy-proportional computing," *Computer*, vol. 40, no. 12, pp. 33–37, 2007.

[7] G. Bigwood and T. Henderson, "Social dtn routing," in *Proc. of ACM CoNEXT*, 2008, pp. 1–2.

[8] 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. of WWW*. ACM, 2009, pp. 551–560.

[9] N. Botts, B. Thoms, A. Noamani, and T. A. Horan, "Cloud computing architectures for the underserved: Public health cyberinfrastructures through a network of healthatms," in *Proc. of HICSS*, 2010, pp. 1–10.

[10] R. Buyya, R. Ranjan, and R. N. Calheiros, "Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services," in *Proc. of ICA3PP*, 2010, pp. 13–31.

[11] M. Cardosa, C. Wang, A. Nangia, A. Chandra, and J. Weissman, "Exploring mapreduce efficiency with highly-distributed data," in *Proc. of ACM MapReduce*, 2011.

[12] A. Chaintreau, P. Fraigniaud, and E. Lebhar, "Opportunistic spatial gossip over mobile social networks," in *Proc. of SNS*, 2008.

[13] A. Chakrabarti, *Grid computing security*. Springer Verlag, 2007.

[14] A. Chandra and J. Weissman., "Nebulas: Using distributed voluntary resources to build clouds," in *Proc. of HotCloud*, 2010.

[15] R. Clarke, "User requirements for cloud computing architecture," in *Proc. of IEEE CCGRID*, 2010, pp. 625–630.

[16] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, J. Gerth, J. Talbot, K. Elmeleegy, and R. Sears, "Online aggregation and continuous query support in mapreduce," in *ACM SIGMOD*, 2010, pp. 1115–1118.

[17] E. M. Daly and M. Haahr, "Social network analysis for routing in disconnected delay-tolerant manets," in *Proc. of ACM Mobihoc*, 2007.

[18] G. Danezis and P. Mittal, "Sybilinfer: Detecting sybil nodes using social networks," in *Proc. of NDSS*, 2009.

[19] A. V. Dastjerdi, S. G. H. Tabatabaei, and R. Buyya, "An effective architecture for automated appliance management system applying ontology-based cloud discovery," in *Proc. of IEEE CCGRID*, 2010.

[20] J. Davitz, J. Yu, S. Basu, D. Gutelius, and A. Harris, "ilink: search and routing in social networks," in *KDD*. ACM, 2007, pp. 931–940.

[21] J. Dean and S. Ghemawat, "Mapreduce: a flexible data processing tool," *Communications of the ACM*, vol. 53, no. 1, pp. 72–77, 2010.

[22] T. S. Dillon, C. Wu, and E. Chang, "Cloud computing: Issues and challenges," in *Proc. of IEEE AINA*, 2010, pp. 27–33.

[23] P. Domingues, B. Sousa, and L. Moura Silva, "Sabotage-tolerance and trust management in desktop grid computing," *Future Generation Computer Systems*, vol. 23, no. 7, pp. 904–912, 2007.

[24] Ebizmba, "Ebizmba," www.ebizmba.com/, 2009.

[25] Facebook, "Facebook," www.facebook.com, 2009.

[26] Y. Fernandess and D. Malkhi, "On spreading recommendations via social gossip," in *Proc. of SPAA*. ACM, 2008, pp. 91–97.

[27] M. Hagiwara, "Development procedure of the cloud-based applications," in *Proc. of DASFAA*, 2010, pp. 320–326.

[28] W. Iqbal, M. N. Dailey, and D. Carrera, "Sla-driven dynamic resource management for multi-tier web applications in a cloud," in *Proc. of IEEE CCGRID*, 2010, pp. 832–837.

[29] W. Iqbal, M. N. Dailey, D. Carrera, and P. Janecek, "Sla-driven automatic bottleneck detection and resolution for read intensive multi-tier applications hosted on a cloud," in *Proc. of GPC*, 2010, pp. 37–46.

[30] T. Isdal, M. Piatek, A. Krishnamurthy, and T. Anderson, "Privacy-preserving p2p data sharing with oneswarm," in *Proc. of ACM SIGCOMM*, vol. 40, no. 4, 2010, pp. 111–122.

[31] S. Kamvar, M. Schlosser, and H. Garcia-Molina, "The eigentrust algorithm for reputation management in p2p networks," in *Proc. of WWW*. ACM, 2003, pp. 640–651.

[32] H. A. Kautz, B. Selman, and M. A. Shah, "Referral web: Combining social networks and collaborative filtering," *Communications of the ACM*, vol. 40, no. 3, pp. 63–65, 1997.

[33] C. Lesniewski-Laas, "A Sybil-proof one-hop DHT," in *Proc. of the workshop on Social network systems*. ACM, 2008, pp. 19–24.

[34] H. Lin, X. Ma, J. Archuleta, W.-c. Feng, M. Gardner, and Z. Zhang, "Moon: Mapreduce on opportunistic environments," in *Proc. of HPDC*. New York, NY, USA: ACM, 2010, pp. 95–106.

[35] W. Lin, H. Zhao, and K. Liu, "Incentive cooperation strategies for peer-to-peer live multimedia streaming social networks," *IEEE Transactions on Multimedia*, vol. 11, no. 3, pp. 396–412, 2009.

[36] M. Litzkow, M. Livny, and M. Mutka, "Condor-a hunter of idle workstations," in *Proc. of ICDCS*. IEEE, 1988, pp. 104–111.

[37] S. Marti, P. Ganesan, and H. Garcia-Molina, "Dht routing using social links," in *Proc. of IPTPS*. Springer, 2004, pp. 100–111.

[38] A. Mohaisen, N. Hopper, and Y. Kim, "Keep your friends close: Incorporating trust into social network-based sybil defenses," in *Proc. of INFOCOM*, 2011, pp. 1943–1951.

[39] A. Mohaisen, H. tran, A. Chandra, and Y. Kim, "Socialcloud: Using social networks to build distributed computing services," University of Minnesota, Tech. Rep., 2011.

[40] A. Mohaisen, H. Tran, A. Chandra, and Y. Kim, "SocialCloud," http://socialcloud.cypriv.com, July 2011.

[41] A. Mohaisen, A. Yun, and Y. Kim, "Measuring the mixing time of social graphs," in *Proc. of IMC*. 11: ACM, 2010.

[42] S. Nagaraja, "Anonymity in the wild: Mixes on unstructured networks," in *Proc. of PETS*, 2007, pp. 254–271.

[43] S. Pandey, L. Wu, S. M. Guru, and R. Buyya, "A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments," in *Proc. of AINA*, 2010, pp. 400–407.

[44] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl, "Grouplens: An open architecture for collaborative filtering of netnews," in *Proc. of ACM CSCW*, 1994, pp. 175–186.

[45] M. Sato, "Creating next generation cloud computing based network services and the contributions of social cloud operation support system (oss) to society," in *Proc. of IEEE WETICE*, 2009, pp. 52–56.

[46] S. Song, K. Hwang, and Y. Kwok, "Trusted grid computing with security binding and trust integration," *Grid computing*, vol. 3, no. 1, 2005.

[47] M. Stonebraker, D. J. Abadi, D. J. DeWitt, S. Madden, E. Paulson, A. Pavlo, and A. Rasin, "Mapreduce and parallel dbmss: friends or foes?" *Communications of the ACM*, vol. 53, no. 1, pp. 64–71, 2010.

[48] D. Tran, F. Chiang, and J. Li, "Friendstore: cooperative online backup using trusted nodes," in *Proc. of SNS*, 2008, pp. 37–42.

[49] N. Tran, B. Min, J. Li, and L. Subramanian, "Sybil-resilient online content voting," in *Proc. of USENIX NSDI*, 2009, pp. 15–28.

[50] E. Vasserman, R. Jansen, J. Tyra, N. Hopper, and Y. Kim, "Membership-concealing overlay networks," in *Proc. of ACM CCS*, 2009, pp. 390–399.

[51] M. Wallis, F. Henskens, and M. Hannaford, "Expanding the cloud: A component-based architecture to application deployment on the internet," in *Proc. of IEEE CCGRID*, 2010, pp. 569–570.

[52] L. Wang, J. Tao, M. Kunze, A. Castellanos, D. Kramer, and W. Karl, "Scientific cloud computing: Early definition and experience," in *Proc. of IEEE HPCC*. Ieee, 2008, pp. 825–830.

[53] J. B. Weissman, P. Sundarrajan, A. Gupta, M. Ryden, R. Nair, and A. Chandra, "Early experience with the distributed nebula cloud," in *Proc. of ACM DIDC*, 2011, pp. 17–26.

[54] B. Wickremasinghe, R. N. Calheiros, and R. Buyya, "Cloudanalyst: A cloudsim-based visual modeller for analysing cloud computing environments and applications," in *Proc. of IEEE AINA*, 2010, pp. 446–452.

[55] S. Xu, X. Li, and T. P. Parker, "Exploiting social networks for threshold signing: attack-resilience vs. availability," in *Proc. of ACM ASIACCS*, 2008, pp. 325–336.

[56] H. Yu, P. B. Gibbons, and M. Kaminsky, "Toward an optimal social network defense against sybil attacks," in *Proc. of PODC*. ACM, 2007, pp. 376–377.

[57] H. Yu, P. B. Gibbons, M. Kaminsky, and F. Xiao, "Sybillimit: A near-optimal social network defense against sybil attacks," in *Proc. of IEEE S&P*, 2008, pp. 3–17.

[58] H. Yu, M. Kaminsky, P. B. Gibbons, and A. Flaxman, "Sybilguard: defending against sybil attacks via social networks," in *Proc. of ACM SIGCOMM*, 2006, pp. 267–278.