# Trustworthy Distributed Computing on Social Networks

Abedelaziz Mohaisen
Verisign Labs, VA, USA

Huy Tran
KAIST, Daejeon, South Korea

Abhishek Chandra
University of Minnesota, MN, USA

Yongdae Kim
KAIST, Daejeon, South Korea

## ABSTRACT

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.

## Categories and Subject Descriptors

C.2.0 [**Computer Communication Networks**]: General – *Security and Protection*; C.4 [**Performance of Systems**]: Design studies

## General Terms

Security, Design, Experimentation

## Keywords

Distributed computing, Trust, Social Computing.

## 1. INTRODUCTION

Cloud computing is a paradigm that overcomes restrictions of conventional computing systems by enabling elasticity and pay-as-you-go, which free users from long-term commitments and obligation towards providers. Cloud computing is beneficial for both consumers and cloud service providers. Despite such benefits, this paradigm also poses several challenges, including the need for architectures to support various potential applications, programming models to address large scale data-centric computing, and the need for strong security and data privacy protection guarantees. Indeed, both outsider and insider threats to security and privacy of data in cloud systems are unlimited. Also, there are many incentives for providers to 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 SocialCloud. Most important to the context of SocialCloud is the aggregate computational power provided by users willing to share their idle time compute cycles [3]. 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 [20, 5], although by enabling trust. 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.

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 to the community to make use of them, and improve by additional features [14].

**Organization.** In §2 we review preliminaries followed by the design in §3. In §4, we describe our simulator followed in §5 by preliminary results, analyses and discussion. In §6, we summarize some of the related work followed by concluding remarks in §7.

## 2. 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.

**Social Graphs—High Level Description.** 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 $1 \le i \le n$ and $1 \le j \le n$) is the set of edges connecting those vertices. $|V| = n$ denotes the size of $G$ and $|E| = m$ denotes the number of edges in $G$.

**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 model, users *donate* their *computing resources* while not using them. One can further improve this model by incorporating a differential trust in scheduling [13]. In this work, and in order to make use of and confirm this model, we limit outsourced computations at 1-hop.

**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 [11], MOON [10], Nebula [5], and SETI@Home [1]. Specific examples of applications built on top of these systems, that would as well fit to our use model, include blog analysis [20], web crawling and social apps (collaborative filtering, image processing, etc) [4], scientific computing [19], among others.

**Attacker Model.** In this paper, as it is the case in many other systems built on top of social networks, 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. For understanding the rationale of this model, see the related literature; e.g., [13]

**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 [2]—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.

## 3. 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.

### 3.1 Design Options: Scheduling Entity

In SocialCloud two schedulers are used; one 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.

**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 and worker has its own separate scheduler.

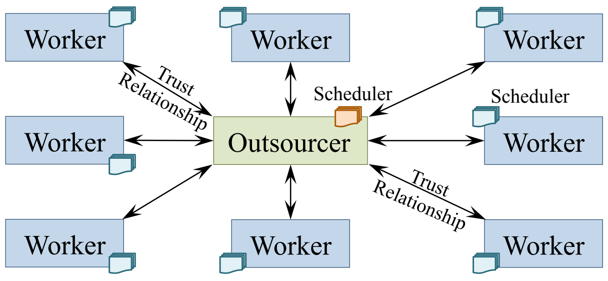


**Figure 1: A depiction of the main SocialCloud paradigm as viewed by an outsourcer of computations.**

**Centralized Scheduler.** Despite 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

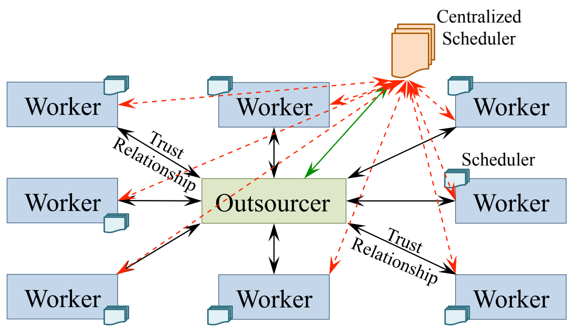


**Figure 2: Decentralized model of scheduling in SocialCloud.**

**Table 1: A comparison of design options**

| Option | Failure | Comm. | Hardware | Trust |
|---|---|---|---|---|
| Centralized | ✖ | $O(n)$ | ✖ | ✖ |
| Decentralized | ✔ | $O(m)$ | ✔ | ✔ |

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 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.

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 1 shows a comparison between both options.

## 3.2 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 at outsourcer's side, which can be further improved by incorporating trust models for weighted job scheduling [13]. On the other hand, we consider several scheduling algorithms for workers scheduling, as follows. (*i*) 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. (*ii*) Shortest First (SF) Scheduling Policy The worker performs shortest task first. (*iii*) 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 §5 for details on limitations of this approach and possible extensions in the future work.

## 3.3 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.

## 3.4 Deciding Workers Based on Resources

In real-world deployment of 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. 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.

## 4. SIMULATOR OF SOCIALCLOUD

To demonstrate the potential of SocialCloud as a computing paradigm, we implement a batch-based simulator [16] 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 is in Figure 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.

**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.

## 5. RESULTS AND ANALYSIS

In this section, and 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.

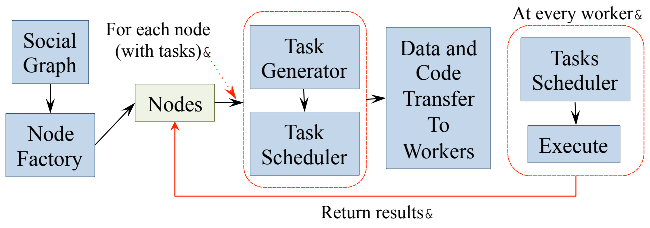


**Figure 3: The flow diagram of SocialCloud.**

## 5.1 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. For that, we use the empirical commutative distribution function (CDF) as an aggregate measure among all nodes with tasks to compute. For a random variable $X$, the CDF is $F_X(x) = P_r(X \leq x)$. In our experiments, the CDF measures the fraction of nodes that finish their tasks before 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 $T_{last}/T_{tot}$.

## 5.2 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. (*i*) **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 $\bar{T}$. (*ii*) **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.

## 5.3 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 §5.2; with fixed or constant and variable tasks weights.

## 5.4 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.

**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 |

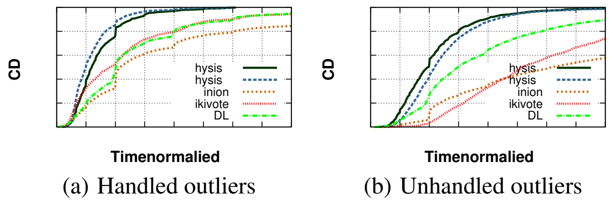
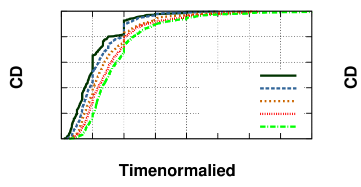


**Figure 5: The performance of SocialCloud on the different social graphs. Both figures use $p = 0.3$ and the RRS.**

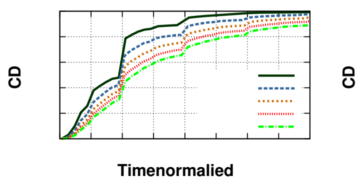
## 5.5 Main Results

**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 4(a) through 4(b) versus figures 4(c) and 4(d)). 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 can be observed by comparing 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.

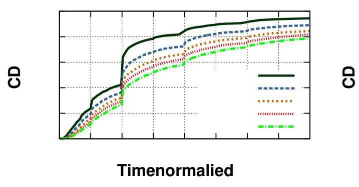
Second, we observe that the impact of $p$, the fraction of nodes with tasks in the system, would depend greatly on the underlying graph rather than $p$ alone. For example, in Figure 4(a), we observe that moving from $p = 0.1$ to $p = 0.5$ (when $x = 1$) leads to a decrease in the fraction of nodes that 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 de-



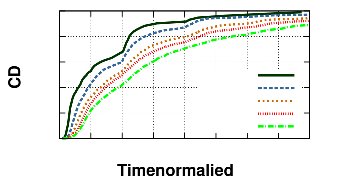
(a) Physics 1.



(b) DBLP.

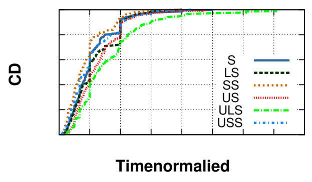


(c) Epinion.

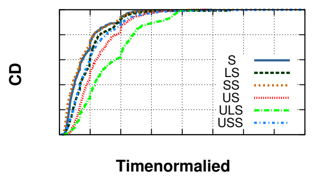


(d) Wiki-vote.

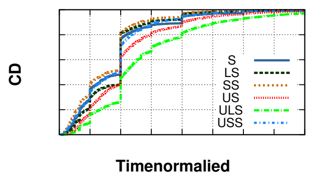
**Figure 4: The normalized time it takes to perform outsourced computations in SocialCloud.**



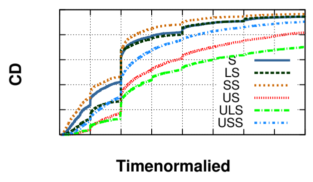
(a) Physics 1 ($p = 0.1$).



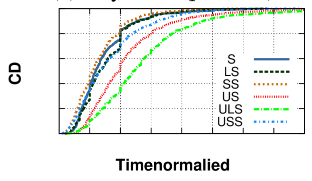
(b) Physics 2 ($p = 0.1$).



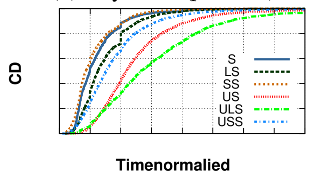
(c) DBLP ($p = 0.1$).



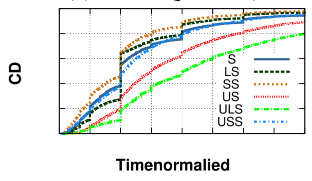
(d) Epinion ($p = 0.1$).



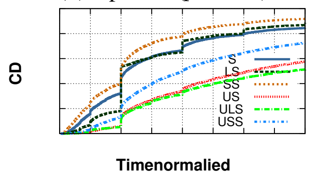
(e) Physics 1 ($p = 0.3$).



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



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



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

**Figure 6: The normalized time it takes to perform outsourced computations in SocialCloud for different scheduling policies; U and stand for unhandled and handled (balanced) outlier. RRS, SFS, and LFS are the scheduling policies.**

creases 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. 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.

**Scheduling Policy.** To understand the impact of policies on the performance, we use $p = 0.1$ to $0.5$ with $0.2$ increments (partly shown in Figure 6). The observed consistent pattern in all figures in this experiment tells that shortest first policy always outperforms the round robin policy, whereas the round robin 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 6(f)) and as high as 70% (when using $p = 0.5$ and outlier handling as in wiki-vote; not shown for the lack of space). The patterns are made clearer in Figure 6 by observing combinations of parameters and policies.

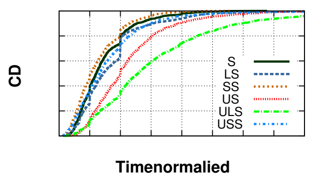
**Performance with Outliers Handling.** Outliers drag the performance of the entire system down. However, handling outliers is quite simple in SocialCloud if accurate timing is used. Here we consider the impact of the outlier handling policy in §3.3. In this figure 6, we see that the simple handling 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 $x = 1$.

**Variable Task Size.** In all experiments so far, we considered 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.

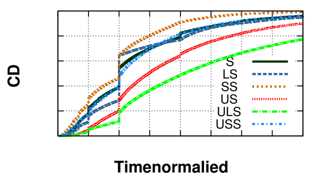
**Structure and Performance.** We note that the performance of SocialCloud is quite related to the underlying structure of the social graph. We see that sparse graphs, like co-authorship graphs, have a performance advantage in SocialCloud. These graphs are shown to possess a nice trust value that can be further utilized in SocialCloud. Furthermore, this trust value is unfound 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 §2 for more details and the work in [13] for prior literature that agrees with our findings.

## 5.6 Additional Features and Limitations

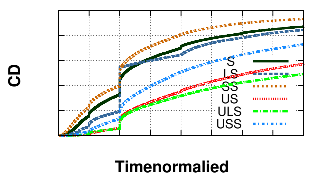
Our simulator of SocialCloud omits a few details concerning the way a distributed system behaves. For example our measurements do not consider failure. Furthermore, our simulator considers a simplistic scenario by abstracting the hardware, and does not consider additional resources consumed, such as memory and I/O resources. We also do not consider the heterogeneity of resources, such as bandwidth and memory. Last, 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 work would be pursued in the future, we expect that nodes would select workers among their



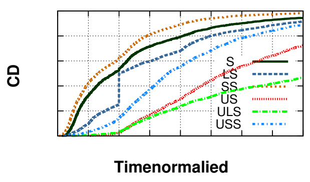
(a) Physics 1 ($p = 0.3$).



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



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



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

**Figure 7: The normalized time to perform outsourced computations in SocialCloud, for variable task size.**

social neighbors that have resources and link capacities exceeding a threshold, thus meeting an expected performance outcome.

## 6. RELATED WORK

Systems built on top of social networks include file sharing [8, 13], anonymous communication [18], Sybil defenses [21, 12, 15], routing [7], and content distribution [22], among many others. Most of these systems use social networks' trust and connectivity for their operation. Concurrent to our work, and following their work in [6], Chard et al. suggested the use of social networks to build a resource sharing system. Whereas their main realization was still a social storage system as in [6], 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 [17] and [9]. In [17], 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 [9] Koshy et al. further explored the motivations of users to enable social cloud systems for scientific computing. 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 [5, 20], examination of programing paradigms on such system [10]. Finally, our work shares several commonalities with grid and volunteer computing systems [11, 10, 5, 20, 1], of which many aspects are explored in the literature. Trust of grid computing and volunteer-based systems is explored in [2]. Applications built on top of these systems, that would fit to our use model, are reported in [20], among others.

## 7. SUMMARY AND FUTURE WORK

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 a computing service. To demonstrate the potential of our design, we used several social graphs to show that the majority of nodes in most graphs would benefit 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.

In the future we aim to complete the missing features 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. We will 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. Second, we will implement a proof-of-concept application 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.

**Acknowledgement.** The last author was supported by the KCC (Korea Communications Commission), Korea, under the R&D program supervised by the KCA (Korea Communications Agency), KCA- 2013-12911-05003.

## 8. REFERENCES

[1] D. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. Seti@ home: an experiment in public-resource computing. *Communications of the ACM*, 45(11):56–61, 2002.

[2] F. Azzedin and M. Maheswaran. Towards trust-aware resource management in grid computing systems. In *Proc. of CCGRID*, 2002.

[3] L. Barroso and U. Holzle. The case for energy-proportional computing. *Computer*, 40(12):33–37, 2007.

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

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

[6] K. Chard, S. Caton, O. Rana, and K. Bubendorfer. Social cloud: Cloud computing in social networks. In *CLOUD*. IEEE, 2010.

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

[8] T. Isdal, M. Piatek, A. Krishnamurthy, and T. Anderson. Privacy-preserving p2p data sharing with oneswarm. In *ACM SIGCOMM*, 2010.

[9] J. Koshy, K. Bubendorfer, and K. Chard. A social cloud for public eresearch. In *eScience*, pages 363–370. IEEE, 2011.

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

[11] M. Litzkow, M. Livny, and M. Mutka. Condor-a hunter of idle workstations. In *Proc. of ICDCS*, pages 104–111. IEEE, 1988.

[12] P. Mittal, M. Wright, and N. Borisov. Pisces: Anonymous communication using social networks. In *NDSS*, 2013.

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

[14] A. Mohaisen, H. Tran, A. Chandra, and Y. Kim. Socialcloud: distributed computing on social networks. Technical report, University of Minnesota, 2011.

[15] A. Mohaisen, H. Tran, N. Hopper, and Y. Kim. On the mixing time of directed social graphs and security implications. In *ASIACCS*, 2012.

[16] SCloud Team. Anonymized for submission. Anonymized for submission, 2012.

[17] A. M. Thaufeeg, K. Bubendorfer, and K. Chard. Collaborative eresearch in a social cloud. In *eScience*, pages 224–231. IEEE, 2011.

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

[19] 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*, pages 825–830. Ieee, 2008.

[20] 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*, pages 17–26, 2011.

[21] 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*, pages 3–17, 2008.

[22] F. Zhou, L. Zhang, E. Franco, A. Mislove, R. Revis, and R. Sundaram. WebCloud: Recruiting social network users to assist in content distribution. In *IEEE NCA*, 2012.