# Reliability Enhanced Social Crowdsourcing

Wei Chang and Jie Wu
Department of Computer and Information Sciences
Temple University, Philadelphia, PA 19122
Email: {wei.chang, jiewu}@temple.edu

***Abstract*—Social crowdsourcing is a special outsourcing platform, on which a complex job is outsourced onto a social network of workers. By interactively recruiting friends of friends, and assigning small pieces of works to them, eventually a large job can be completed. However, due to the workflows presenting a tree-like structure, the reliability of the current social crowdsourcing system is poor. The absence of a relay worker will cause disconnection between the job owner and sub-workers. For increasing the reliability, we explore two special social structures. One is based on a triangle relation among any three consecutive workers on the workflows; the other one is based on a quadrilateral relation between two physically intersected workflows. Based on these structures, we propose several social crowdsourcing return rules. Theoretical analysis and simulation results show that our proposed schemes can significantly improve the performance and reliability of social crowdsourcing.**

***Index Terms*—Crowdsourcing, failure recovery, reliability, social crowdsourcing, social tie.**

## I. Introduction

Crowdsourcing are increasingly used to solve human intelligence-related problems, such as proofreading. In crowdsourcing, a large work is partitioned into smaller pieces by its owner, and outsourced onto a crowdsourcing platform. Independent freelances search and take up some subworks. After finishing sub-works, they return results to the platform. In Amazon Mturk, a subwork is called a Human Intelligence Tasks (HIT). Although crowdsourcing brings more knowledge diversity and a large amount of labor force [1], the independent feature of workers causes the problem that it can only process simple and independent works [2].

In this paper, we propose a new outsourcing system, called *social network-based crowdsourcing* (SC), which outsources works onto a social network of workers. Unlike the existing systems, workers of SC are not independent: they have some level of awareness and trust, and therefore, cooperation locally exists. A job can be completed, via iterative recruitment of workers through social ties. Instead of directly interact with each worker, a job owner only needs to outsource his workloads to friends, and leaves them to further propagate the works, collect results, and post-process the results.

SC works as follows: when user $v$ possesses a job, he first creates social-HITs, and forwards them to his friends. Social-HIT is a special data structure, which records job description, results' requirements, and the identity of the requester who send the social-HIT to its current holder. When $u$ accepts a social-HIT, he is required to check whether the social-HIT should continue to be forwarded to his friends, according to the requirements of social-HITs. Each participant is also responsible for collecting its subtrees' results. After the amount of collected results satisfies the given return requirements, $u$ will process these results, such as reduce or aggregate them, and send them back to the requester. Physically, the job's workflows form a directed acyclic graph, but *logically, the flows of social-HITs form a virtual tree*.

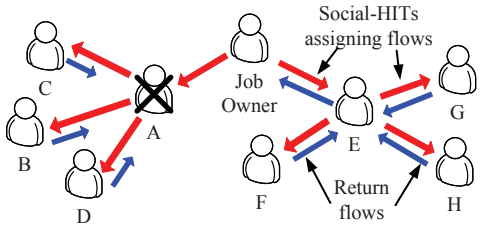



Fig. 1. An example about the absence of a relay worker. Worker $A$ was on-line during the propagation of social-HITs, and then left. The return flows from its subtrees are disrupted by the absence of $A$.

However, the tree-like structure is unreliable for returning results: the off-line or long-term unresponsiveness of a relay worker will cause the disconnection between the sub-workers and the job owner, as shown in Fig. 1. Redundant preplanned return paths can be adopted to improve the reliability. But, inappropriate selection and using backup paths may even worsen the situation. For example, for controlling the waiting time of collecting subtrees' results, the return requirements could be: 1) $u$ has collected results from $c$ or more than $c$ descendants; 2) $u$ is awake (an on-line status). Suppose we adopted a recovery scheme that, when a worker withdraws, his children are allowed to directly give the results back to a pre-selected worker. If this is the case, not only the backup collector's children may lose the chance to return when the missing worker's children use up the $c$ return slots early, but also the backup collector itself may be overflowed by the data when multiple failures happens. Moreover, the problem is more complex, due to the facts: (1) workers in SC are distributed and self-organized; they only possess some status information of their friends; (2) workers may be sleeping or awake, which leads to the uncertainty of return flows' delay and order.

In this paper, we focus on the design of redundant return paths and the rules for using them. We first design a distributed *dual return paths* scheme by exploring the relationships among any three consecutive workers on the workflows. The proposed scheme is local, lightweight, and can be directly implemented on the current crowdsourcing platform. It is proven that the scheme can tolerate any non-consecutive node failure or any non-common-source link failure. Moreover, the negative effects of using the backup paths are controlled. Through the qualitative and quantitative analysis, we show that the proposed scheme significantly improves the performance, especially the reliability, of SC systems. Considering the fact that most social-HITs are propagated via the popular users (who have a large node degree), once such a worker misses, his single backup collector will be overflowed quickly. In order to avoid this problem, we propose an extension scheme,

**Algorithm 1** P-SNCA

1: Current node $u$ accepts a social-HIT from $v$: $\langle JobId, v, LifeTime, Hop, Instruct\rangle$
2: **if** $Hop > 0$ **then**
3: Generate a new social-HIT: $\langle JobId, u, LifeTime, Hop\text{-}1, Instruct\rangle$
4: Send the new social-HIT to every friend of $u$ except $v$
5: Complete the work described in $Instruct$
6: **while** Waiting for the returns from $u$'s friends **do**
7: **if** The collected results satisfy the return requirements described in $Instruct$ **then**
8: Process the results, and send them back to $v$; **break**
9: **if** Current time $t > LifeTime$ **then**
10: Destroy the social-HIT at $u$; **break**
11: **else**
12: Complete the work described in $Instruct$
13: Return the work's result to $v$

by letting the second social-HIT's requester, from the same 'grandfather,' be the backup collector. Theoretical analysis and simulation results show that our proposed schemes greatly increase the number of collected results at the job owner. The main contributions of our work are as follows: 1) We find two special local structures for selecting of the backup returning nodes: GFC structure-based scheme and Second Requester-based scheme; 2) Three rules are designed for the coordinated use of the return paths; 3) Extensive simulations and theoretical analysis are performed to testify our solutions.

## II. Related work

The organization of a workplace affects the characters of works and outcomes [3]. The commonly used one is the layered structure, where people in a higher position fully control those that are lower [4]. But this structure lacks flexibility, and hurts the willingness of workers. Crowdsourcing [5] is a new workplace structure, where one can obtain the needed services from self-employed workers. However, the independence and the uncertainty of workers lead to new problems, such as low quality assurance. In this paper, we propose a new outsourcing system called SC, where works are outsourced onto a social network of workers. Fault tolerance is an important issue in distributed systems. Typically, recovery schemes create several backup paths to bypass the fault. Due to the specific features of a system, the recovery scheme may be implemented by establishing two spanning trees [6], creating link/node disjointed paths [7], or dynamically maintaining a breadth first search tree of living nodes [8]. Unlike traditional distributed systems, the nodes on crowdsourcing are real humans. They are not always available, nor sharing status information with strangers. Our scheme explores the local features of participants, and can be implemented by Mturk's APIs.

## III. System Model of Social Crowdsourcing

SC models the job's outsourcing procedure via the process of iteratively recruiting friends' friends. Since each user serves as a computing entity in a distributed system, we also refer to users as 'nodes.' Whenever a user has a job, the user will create social-HITs and send them to his friends. The user is regarded as the job owner. Unlike traditional crowdsourcing, where workers search for works, in SC, a social-HITs is directly sent from a user to his friend. In crowdsourcing,

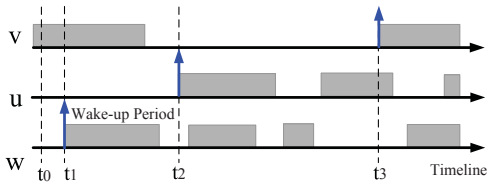



Fig. 2. Sleep/Wake-up period-related delay. Assume that $v$ and $w$ are the father and son of $u$, respectively. At time $t_0$, all of them collected $c-1$ returns from their subtrees, and $w$ got another returns at $t_1$. Since $w$ is awake, it immediately returns the results to $u$, who is in sleep status. When $u$ waked up at $t_2$, he found that he had satisfied returning conditions, and then, he forwarded the results to $v$. Although $v$ had collected enough returns, due to the longer sleep period of $v$, the results are returned at $t_3$.

workers are always alive and work, but in SC, workers may be online, offline, or stop working. We name an online worker $Awake$, and an offline user $Sleep$. During the propagation of social-HITs, some nodes may sleep; the platform will store the social-HITs for them. When they wake up, if the social-HITs have not expired, the workers can still accept them. Basically, the system just provides a communication platform for the peer workers. Due to the existence of sleeping nodes, communication delays may exist. For example, in Fig. 2, there is a $t_2 - t_1$ time delay between $u$ and $v$. Since SC's workers are friends, a node is able to have extra information about his friends, such as the distribution of sleep/awake status' duration, the real-time sleep/awake status, and the number of children, who have returned the results to him. From security and privacy concerns, such information should only be shared between friends.

## IV. Social Network-based Crowdsourcing

A social-HIT is defined as a quintuple:

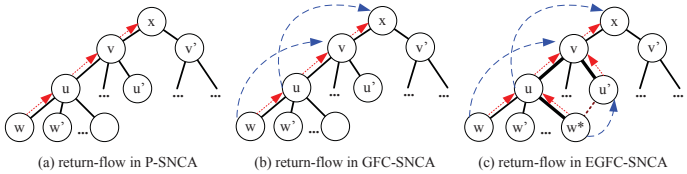
$$\langle JobId, Father, LifeTime, Hop, Instruct\rangle.$$

$JobId$ is a unique identity of the original job. As a social-HIT is transmitted from user to user, $Father$ is used to indicate the one, who gave the social-HIT. Every job is associated with a life time, $LifeTime$, in order to timely clean-up the starved job. When time is up, any related social-HITs will be destroyed, no matter whether their results have been submitted. For controlling the scope of participants on a social network, a propagation counter, named $Hop$, is used to record the number of remaining hops; when $Hop = 0$, the job will not be transmitted anymore. The job's description, results' return requirements, and other details are recorded in $Instruct$. After obtaining a social-HIT from a friend, the receiver is able to decide whether to accept it, or not; for the same job, a node may reject it when it comes from one friend, yet may accept it when it is from another. Algorithm 1 shows *Primitive Version of Social Network-based Crowdsourcing Algorithm* (P-SNCA).

Algorithm 1 creates a self-organized system, and each pair of friends exchanges at most two rounds of messages: forwarding social-HITs and collecting results. The overall logic flow of social-HITs presents a tree-like structure (Fig. 3 (a)), where the job owner is the root. Suppose that the initial value of $Hop$ equals $H$, the average communication delay between nodes is $d$, and the average number of friends is $r$. Ideally, the job owner communicates with $r$ users, but there are $O(r^H)$ users working on this job, which will be

TABLE I. DECREASING PATTERN OF $P_H$

| $P_H$ | $c=6$ | $c=7$ | $c=8$ | $c=9$ | $c=10$ | $c=11$ |
|---|---|---|---|---|---|---|
| $H=5$ | .7999 | .7994 | .7962 | .7736 | .1828 | 0 |
| $H=6$ | .7999 | .7994 | .7962 | .7724 | $4e^{-5}$ | 0 |
| $H=7$ | .7999 | .7994 | .7962 | .7716 | 0 | 0 |




(a) return-flow in P-SNCA (b) return-flow in GFC-SNCA (c) return-flow in EGFC-SNCA

Fig. 3. The return-flows in a social network-based crowdsourcing system.

completed within $2dH$ time (by ignoring the job's processing time). Because works and results are propagated and collected hop-by-hop, SC is *a native privacy-preserved system*: whether a node having or participating in a work is only known by its direct friends. The system successfully hides the identity of the job owner to sub-workers. However, due to the tree-like structure, P-SNCA system is not reliable: if a relay node $u$ fails ($0 < Hop(u) < H$), all results on its subtrees will be lost. Note that different return requirements make a little difference on the reliability. But, from the aspects of severity, failure conditions, and recovery schemes, they are the same. In this paper, we focus on a specific return requirement: *when a node is awake and receives $c$ replies, the node immediately returns his result; if the node is in sleep and receives more than $c$ replies during sleeping, it should return all of them when it wakes up.*

**Successful Return Rate Analysis:** Suppose that the work-flows of P-SNCA form a complete $r$-ary tree with height $H+1$ (including the original job owner) and every node has the same reliability $R$. Let $P_i$ be the probability that a node with $Hop = i$ successfully returns its sub-trees' results to its father node. $P_i$ can be recursively computed by: $P_i = R \cdot \sum_{j=c}^{r} \binom{r}{j} \cdot P_{i-1}^{j} \cdot (1-P_{i-1})^{r-j}$ where $1 \le i \le H$, and $P_0 = 1$. $P_i$ greatly drops at certain $H$ and $c$. For example, if $r=15$, $R=0.8$, we have Table II.

**System Reliability Analysis:** In SC, a node failure means that user $u$ sends a package to a non-root user $v$, but $v$ fails to forward the package to the next user. The failure of nodes may appear in the form of *absent nodes*, who are in the sleep status, or *incompetent nodes*, who cannot fulfill the return requirements. A link failure indicates the case, where $u$ sends a package to $v$, but $v$ does not receive it. In realistic, the communication between nodes is implemented by a reliable platform, such as Amazon Mturk. The occurrence rate of link failure is much less than that of the node failure. Let $v$ be $u$'s father, and $x$ be $u$'s grandfather. For $u$, there are three conditions, which cause $v$'s failure: (1) $v$ had already returned results to $x$, before $u$ satisfies the return requirements. (2) $u$ returned results to $v$, but $v$ is in the sleep status; $x$ leaves before $v$'s next wake-up time. (3) $u$ submitted results to $v$, and $v$ is awake; $x$ leaves before $v$ satisfies the return requirements. These three are the general conditions; many cases may cause them. Moreover, one node's failure may lead to the failure of others, since the accomplishment of a node depends on the fulfillment of its subtrees.

## V. RELIABILITY ENHANCED SC

### A. Grandpa, Father, Current node (GFC) structures

For the purpose of increasing reliability and resource utility, we need to provide alternative return paths for P-SNCA. Due to the pure-distributed feature, the new return path generating scheme must be lightweight, only using local information, and can be implemented on the current crowdsourcing platform. We found that, by exploring the relationships between a node and its grandfather/sibling, the performance of P-SNCA can be significantly improved.

***Definition** 1:* GFC represent a triangle relation in which a non-root node records the identities of its father (a primary return node) and grandfather/sibling (a backup return node).

Given a SC system with $Hop \in [0, H]$, for $\forall u$, if $Hop(u) \in [0, H-2]$, $u$ records the identities of its father and grandfather; if $Hop(u) = H-1$, it records the identities of father and one of the siblings. Assume that the original job owner is always available before $LifeTime$ expires.

***Property** 1:* The proposed GFC structure can tolerate any non-consecutive node failure.

***Property** 2:* The proposed GFC structure can tolerate any non-common-source link failure, if there is at least one return flow unbroken between the root and its children.

### B. GFC Structure-based SNCA (GFC-SNCA)

Assume that worker $w$ is ready to return. He needs to determine whether to send results to its father $u$, or to grandfather $v$, such that the job owner eventually can collect as many results as possible. If $w$ decides to give the results directly to $v$, he also needs to determine the time of returning. If $w$ replies too late, both $u$ and $v$ could leave, while, if it returns too early, many potential results will be dropped off. For example, let $w$ be a node in our complete $r$-ary tree analysis model, $Hop(w) = i$. Suppose that $v$ has collected $c - \Delta c$ replies, $1 \le \Delta c < c$. If $w$ returns to $v$ immediately, $v$ will collect $(c-\Delta c) \cdot c^{i+2} + c^{i+1} = c^{i+3} - (c \cdot \Delta c - 1)c^{i+1}$ units of data. However, other children of $v$ may return results in a later time; $v$ could collect $c^{i+3}$ units of data if $w$ did not return to him. Approximately, there is a $\Delta c \cdot c^{i+2}$ difference.

Obviously, the optimal return node selection depends on real-time success returning probabilities of the paths, from $u$ to root, and from $v$ to root. But, due to the lack of global information on the number of collected replies of each node and the uncertainty of future sleep/awake statuses, computing such probabilities is impossible. Since each node has only two options and early using the backup one may cause results dropping, GFC-SNCA adopts the following idea: *a node does not use its backup return node, as long as the primary one have enough opportunity to return data to a higher-level node.* GFC-SNCA locally predicts father's failure, and gives the return time when a node needs to use the backup return node.

*1) Predicting of Father's Failure:* The success of a recovery scheme depends on the accuracy of failures' prediction. In SC, a non-root node needs to know two things about its father and grandfather nodes: 1) the probability of their sleep/awake status at the next decision time $t + \Delta t$; 2) the probability that a given node will collect the required number of returns at

$t+\Delta t$. By knowing a node's current status and its sleep/awake pattern, every node can locally estimate the first probability of its friends. Note that, in SC, how many results will be collected by a node during a $\Delta t$ interval is determined by the number of newly wake-up nodes on its subtrees, who have collected greater or equal to $c$ returns. Therefore, by gradually estimating the sleep/awake status from leave to root, one can compute the probability about the number of returns that a node may collect. Although the predicting results are accurate, this method destroys the distributed feature of SC. So, we need to approximate the second probability by local information.

Consider that the majority of new incoming returns during $\Delta t$ comes from the children, who were in the sleep status but had collected the required number of results at $t$. Therefore, we approximate the second probability that $v$ will collect enough replies at $t+\Delta t$ by only using the information of $v$'s children. For the ease of description, let $|\cdot|$ represent the number of elements in a given set, $N(v)$ be the responded friend set who accepts $v$'s social-HITs, $ACC(v,t)$ be the number of replies collected by $v$ at time $t$, $Father(v)$ gives the identity of $v$'s father, and $S(v,t)$ gives $v$'s status at $t$. There are four possible statuses associated with $S(v,t)$: $Sleep$, $Awake$, $Done$, and $Dead$. Before $v$ returns results, it could be in either $Sleep$ or $Awake$. When $u$ submits results to a node $v$, then the status of $u$ becomes $Done$. When the statuses of $u$'s both return paths are either in $Done$ or $Dead$, $u$ is $Dead$.

Estimator on the probability of satisfying return requirements (EPSRR) is given by Algorithm 2, which contains two parts. Part one (lines 1-5) computes the wake-up probability $P_w(\cdot)$ of node $v$'s children, who have already collected the required amount of replies, but are in the sleep status. How to compute the exact value of conditional probability $Pro\{S(u,t+\Delta t) = Awake \mid S(u,t) = Sleep\}$ is determined by the sleep/awake function of nodes. Part two (lines 6-10) estimates $v$'s replying probability at $t+\Delta t$. $\{P_w > 0\}$ gives a set of $v$'s children, who may wake up at $t+\Delta t$. Suppose that $v$ has collected $ACC(v,t)$ replies. Part two checks whether it is possible for $v$ to collect another $c - ACC(v,t)$ replies during the $\Delta t$ time interval, by using the nodes in $\{P_w > 0\}$. If it is possible, EPSRR returns with the corresponding probability.

*2) Returning Rules in GFC-SNCA:* Based on the local status information and the estimated replying probability of children, we can detect and predict nodes' failure, and accordingly select a returning node for a worker. In this part, we propose three returning rules for enhancing the reliability.

Ideally, the complete time of a node should be earlier than that of its father. However, due to the unbalanced tree structure and the sleep/awake status, a father node may leave before its children. Here are the rules that a node should directly submit its results to the grandfather node.

***Rule 1:*** For node $u$ satisfied $S(u,t) = Awake$, $ACC(u,t) \geq c$, and either (1) $S(v,t) = Done/Dead$ or (2) $ACC(x,t) > C_1$, $ACC(v,t) < C_2$, if its grandfather node $x$ satisfies one of the following three conditions, then $u$ should give the results to $x$ instead of $v$.

1) If $S(x,t) = Sleep$, $ACC(x,t) \geq c$, then $u \to x$ at $t$.
2) If $Pro\{ACC(x,t+\Delta t) \geq c \mid ACC(x,t+\Delta t) < c\} \cdot Pro\{S(x,t+\Delta t) = Sleep \mid S(x,t) = Sleep\} \geq P_s$, then $u \to x$ at time $t$.
3) If the number of collected results of $x$ does not change in the next $k$ consecutive time intervals, and $ACC(x,t+k\cdot\Delta t) > C_1$, then $u \to x$ at $t+k\cdot\Delta t$.

$P_s$, $C_1$, and $C_2$ are three thresholds, where $0 < P_s \leq 1$ and $C_1 > C_2 > 0$. Conditions $S(u,t) = Awake$ and $ACC(u,t) \geq c$ indicate that $u$ is ready to return. There are two possible problems that could be associated with its father $x$. First, $x$ has left, $S(v,t) = Done/Dead$. Second, it has only collected a small number of replies by the time the grandfather node $x$ is almost done. In both cases, it is unlikely for $v$ to forward $u$'s results to $x$. But, as we mentioned before, directly returning results to $x$ at an inappropriate time will quickly use up $x$'s return slots. Note that, even if a node has collected enough replies, it will not return them until it is in awake status. So, when the grandfather node has collected, or will collect enough returns while sleeping, $u$ can directly send the results to $x$. For the situation that node $x$ has not received any reply for a long time since the last return, it is possible that the rest return flows are broken. If we allow $u$ to forward its results to $x$ in such situation, the return flows of both $u$ and $x$ can keep going.

Consider the users with less than $c+1$ friends. Although they can propagate social-HITs, it is impossible for them to meet the return requirements. The root definitely will never get the results from the subtrees of these users.

***Rule 2:*** If $|N(u)| = 0, 0 < Hop(u) < H, S(v,t) = Sleep$, and either (1) $ACC(v,t) \geq c$, or (2) $Pro\{ACC(v,t+\Delta t) \geq c \mid ACC(v,t+\Delta t) < c\} \cdot Pro\{S(v,t+\Delta t) = Sleep \mid S(v,t) = Sleep\} \geq P_s$, then $u$ could return its results to the father $v$, $u \to v$, at time $t$.

In Rule 2, node $u$'s social-HITs are not accepted by any of its friends. If its father $v$ is in sleep status and has already collected the required number of replies, whether $u$ gives results to $v$ will not affect the returning of any other node. Therefore, we let $u$ directly send the result to $v$. Another similar condition is that $v$ will keep sleeping and collect the required number of results with a high probability. We also allow $u$ to directly communication with $v$.

For the GFC structures containing lower degree grandfather or father nodes, if $u$ satisfies one of the following conditions in Rule 3, it is allowed to return results directly to its grandfather.

***Rule 3:*** Node $u$ is ready to return results to its father $v$, $S(u,t) = Awake$, $ACC(u,t) \geq c$. Let $x$ be $u$'s grandfather.

1) If $|N(v)| \leq c$ and $|N(x)| \leq c$, then $u \to x$ at time $t$.
2) If $|N(v)| \leq c$, $|N(x)| > c$, $S(x,t) = Sleep$, and either (1) $ACC(x,t) \geq c$, or (2) $Pro\{ACC(x,t+\Delta t) \geq c \mid ACC(x,t+\Delta t) < c\} \cdot Pro\{S(x,t+\Delta t) = Sleep \mid S(x,t) = Sleep\} \geq P_s$, then $u \to x$ at $t$.
3) If $|N(v)| > c$, $|N(x)| \leq c$, $ACC(x,t) > C_1 \cdot |N(x)|/c$, and $ACC(v,t) < C_2$ then $u \to x$ at $t$.

In Rule 3 case 1, both father and grandfather nodes do not get enough social-HIT acceptance; if $x$ can collect more than $c$ returns from its grandchildren, then it still can give its results back. Rule 3 case 2 shows the condition where the father node does not have enough acceptance. Similar to Rule 2, if the sleeping grandfather node has collected, or will collect $c$, or more than $c$ results, $u$ can directly submit to $x$ without the disturbances of others' submission. For the last case, if the

**Algorithm 2** Estimator on Probability of Satisfying Return Requirements (EPSRR)

**for** $\forall u, u \in N(v), Father(u) = v$ **do**
  **if** $S(u,t) = Sleep$ and $ACC(u,t) \geq c$ **then**
    $P_w(u) = Pro\{S(u, t+\Delta t) = Awake | S(u,t) = Sleep\}$
  **else**
    $P_w(u) = 0$
**if** $|\{P_w > 0\}| \geq c - ACC(v,t)$ **then**
  $Pro\{ACC(v, t+\Delta t) \geq c\}$ = $Pro\{$combination of $\{P_w\}\}$
**else**
  $Pro\{ACC(v, t+\Delta t) \geq c\} = 0$
**Return** $Pro\{ACC(v, t+\Delta t) \geq c\}$

collecting progress of $v$ is too slow while grandfather node $x$ has collected a majority of the returns from its children, $u$ will be allowed to send results to $x$.

### C. Performance Analysis of GFC-SNCA

**Successful Return Rate Analysis:** By exploring the friendships within 2-hops, the successful return rate will depend on the completing of both children and grandchildren nodes. Let $P(u)$ and $Q(u)$ be the probability that node $u$ completes the jobs and returns to its father node by using P-SNCA and GFC-SNCA, respectively. $N^2(u)$ represents a set of nodes, who accept the social-HITs from $u$'s children.

***Theorem** 1:* In P-SNCA, if $|N(u)| < c$, the return rate of node $u$ must be zero, $P(u) = 0$; in GFC-SNCA, if $|N(u)| < c$ but $|N^2(u)| \geq c$, node $u$ may be able to return its subtrees' results, $Q(u) \geq 0$.

***Theorem** 2:* For any given workflow graph, if node $u$ satisfies $2 \leq Hop(u) \leq H$, and the prediction of father nodes' failure is accurate, then the successful return rate at $u$ always has $Q(u) > P(u)$.

Again, let us consider our simpler model, the workflows of which form a complete $r$-ary tree with height $H+1$. Let $Q_i$ be the probability that a node with $Hop = i$ completes the jobs and returns to its father node, function $g(y,z,q) = C_y^z \cdot q^z \cdot (1-q)^{y-z}$. $C_y^z$ means the number of $z$-combinations from $y$ elements. We have:$Q_i = R \cdot \sum_{j=0}^{c-1} \left( g(r,j,Q_{i-1}) \cdot \sum_{k=c-j}^{(r-j)(c-1)} g\big((r-j)(c-1),k,Q_{i-2}\big) \right)$
$+ R \cdot \sum_{j=c}^{r} \left( g(r,j,Q_{i-1}) \cdot \sum_{k=0}^{(r-j)(c-1)} g\big((r-j)(c-1),k,Q_{i-2}\big) \right)$
where $Q_0 = P_0 = 1, Q_1 = P_1 = R$. $Q_i$ is an approximation of GFC-SNCA's return rate: we ignore the case where both $u$ and its father $v$ directly send results to $x$. This equation contains two parts. The first part computes the return probability when less than $c$ nodes from the $i-1$ level return results to the node in the $i$ level. The second one computes the same probability when there are equal or greater than $c$ replied from the $i-1$ level. $Q_j$ partially depends on $Q_{j-1}$. If a level $j$ node $v$ fails, the number of successively returned $v$'s children must be less than $c$. Let $H = 10, r = 15, c = 14$ and $R = 0.1$. By the above approximation, we have $Q_{10} = 0.0713$. But, under the same setting, $P_{10} = 0$.

**System Reliability Analysis:** As Property 1 shows, the GFC structure can tolerate any non-consecutive node failure. Consider the fact that the failure of a node is usually caused by the failure of another one. If the proposed GFC-SNCA can timely and accurately detect the existence of node failures, then the probability of having consecutive node failure can also be reduced. From the returning node selection rules in GFC-SNCA, a reader can see that the children of a failure node only return their results to the grandfather node, who has satisfied the return requirements but is in sleep status, or who has not collected any more results in a while. Suppose that node $u$ is ready to submit its results. If $u$'s father or grandfather is in one of the three failure conditions in Section IV.B, $u$ just returns its results to the alternative returning node.

### D. Extension

When using GFC-SNCA, if $u$ fails, plenty of results from $u$'s children will flock to the $u$'s father, $v$, which may cause $v$'s return slots being quickly used up. In this section, we present an extension version of GFC-SNCA, called EGFC-SNCA, which explores the second-requester path to bypass the return flow. During the propagation of social-HITs, $u$ may receive the same social-HITs multiple times from different friends. Some of them may have the same grandfather but a different father. We define a second requester as follows:

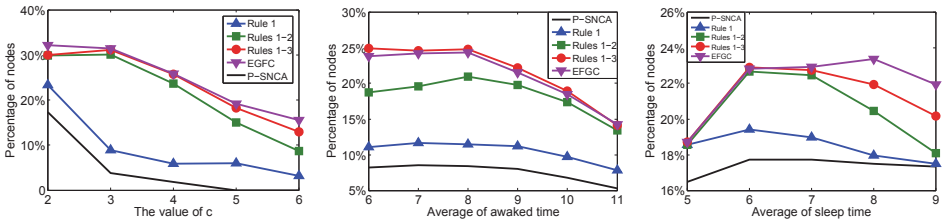
***Definition** 2:* According to the receiving time, the sender of the second received social-HIT, who has the same grandfather as the first social-HIT, is called the Second Requester.

When $u$ detects the failure of its father, instead of returning results to the grandfather, $u$ should first attempt to submit them to the second requester $y$. If $y$ fails, then $u$ can use its grandfather $x$ as the backup return node. The return rules for the second requester are the same as the ones for grandfather nodes, by simply replacing $x$ with $y$ in rules. Due to the page limitation, we will not restate the rules. Note that, by using the second requester, the return flow from $u$ can be controlled under the subtree of $x$; therefore, the potential negative effects for using the backup paths are limited in a local scope.

## VI. Evaluation and Performance Analysis

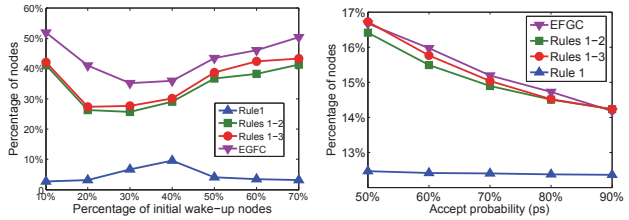
We synthetically create a social network, which follows the power law distribution. Each node is associated with an initial wake/sleep status, which is randomly assigned and follows uniform distribution. The length of a sleep/awake period follows normal distribution. In order to simulate the preference of a node, we assign a random weight on each pair of friendships. Assume that each node can generate one result. We use the number of collected results out of the total number of participated nodes as our evaluation metric.

Fig. 5 compares P-SNCA and GFC-SNCA. When using P-SNCA, only $1.8\%$ of nodes can successively return their results. Since the social-HITs are propagated based on their arriving time and users' preference, the number of nodes with different $Hop$ does not increase monotonously. Instead, it has a fusiform distribution, which causes the situations in which many nodes have no children; even if they have children nodes, the amount of them does not stratify the returning requirement. We check the number of nodes, which may satisfy one of our proposed rules. When the root node cannot collect results any longer, we count these numbers. There are $1.2\%$ of nodes, who have collected the required amount of replies after their father node submitted; about $25.4\%$ nodes have a non-zero $Hop$ value, but have no responded child; there are $1.6\%$ nodes




(a) Return requirement $c$ (b) Ave. length of awake time (c) Ave. length of sleep time

Fig. 4. The impacts of simulation setting-related parameters




(a) Initial status (b) Parameter $P_s$ (c) Waiting time $k$ in Rule 1

Fig. 6. The impacts of returning rule-related parameters

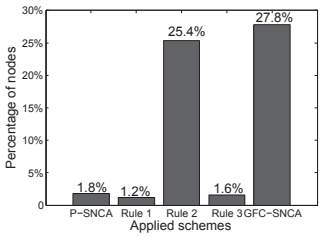



Fig. 5. Failure types' distribution

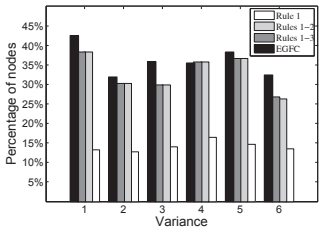



Fig. 7. Impacts of sleeping variance

whose father or grandfathers have less than $c$ children. After adopting our proposed algorithm, the root can collect results from approximately 28% of nodes.

The value of the returning requirement $c$ affects the total number of results that the root can collect. In Fig. 4 (a), with the growth of $c$, the percentage of collected results sharply decreases. Next, we test the impacts of the length of awake/sleep status. In Fig. 4 (b), we fixed the average length of sleeping time, while gradually raising that value of the awake time. The percentage of collected results slightly increases first, and then drops. It seems that there is some kind of balance between the average length of sleep status and awake status, and Fig. 4 (c) confirms our observation. The percentage of replied nodes also goes up first, and then drops down. Since the length of sleep/awake status follows normal distribution, we further check the impacts of the variance in Fig. 7. However, there is no obvious changing pattern: they are all influenced by the variance with different amplitudes.

In Fig. 6 (a), we gradually increase the number of awake nodes. For Rule 1, when there are about equal numbers of sleeping and awake nodes, the root node collects the maximum number of replies. But, for the others, the amount of collected replies drops first and then goes up. One possible explanation is that the number of nodes satisfying Rule 1 reaches its maximum when there are an equal number of sleeping and awake nodes, while the number of nodes satisfying the others rules gets its extremum at the opposite conditions. In Fig. 6 (b), we test the impacts of $P_s$, which determines whether or not to give results to a backup returning node. The percentage of replied nodes slowly decreases with the growth of $P_s$. Since only a few nodes satisfy Rule 1, the decreasing speed is relatively slower. EGFC beats all others. In Fig. 6 (c), we test parameter $k$, which controls the waiting time in Rule 1. With the growth of $k$, the number of collected results forms a concave shape. When $k$ is small, the returning slots may be quickly occupied by the grandchildren; if $k$ is too large, the father of the backup node may leave. Therefore, the number of collected results first increases, and then drops.

## VII. Conclusion

SC explores the social relations among workers: recruiting a worker not only means hiring the worker himself, but also means hiring the worker's friends, and friends of friends, who may be able to help the worker complete the job. However, due to the tree-like structure of social-HITs' logic flows, the SC system is not reliable. For increasing the reliability, nodes need redundant return paths. But inappropriate assignment or use of these paths will even worsen the situations of failures. Due to the unique features of SC, recovery schemes of traditional distributed systems are either ineffective or unrealistic. In this paper, we proposed several rules for locally selecting and using the backup return paths. The basic idea is that, by allowing a node to monitor and predict the statuses of its primary and backup return nodes, the node is able to return the results to the higher-level nodes while avoiding the backup node's premature return. Theoretical analysis and extensive simulations prove the significant performance of our schemes.

## References

[1] G.-J. Qi, C. C. Aggarwal, J. Han, and T. Huang, "Mining collective intelligence in diverse groups," in *IW3C2 WWW*, 2013.

[2] A. Kittur, B. Smus, S. Khamkar, and R. E. Kraut, "Crowdforge: Crowdsourcing complex work," in *ACM UIST*, 2011.

[3] T. F. Bresnahan, E. Brynjolfsson, and L. M. Hitt, "Information technology, workplace organization, and the demand for skilled labor: Firm-level evidence," *The Quarterly Journal of Economics*, 2002.

[4] R. Ashkenas, D. Ulrich, T. Jick, and S. Kerr, "The boundaryless organization: Breaking the chains of organizational structure," *Economica*, 2008.

[5] W. Chang and J. Wu, "Progressive or conservative: Rationally allocate cooperative work in mobile social networks," *IEEE TPDS*, 2014.

[6] G. Xue, L. Chen, and K. Thulasiraman, "Quality-of-service and quality-of-protection issues in preplanned recovery schemes using redundant trees," *IEEE J-SAC*, 2003.

[7] C. S. Ou, J. Zhang, H. Zang, L. H. Sahasrabuddhe, and B. Mukherjee, "New and improved approaches for shared-path protection in WDM mesh networks," *IEEE JLT*, 2004.

[8] E. Gafni and D. Bertsekas, "Distributed algorithms for generating loop-free routes in networks with frequently changing topology," *IEEE Transactions on Communications*, 1981.

## APPENDIX

### A. Proof of Property 1:

*Proof:* We regard 'tolerate' to be that the failure of any node will not break off the return flow between its father and subtrees. There are two possible cases with a failure node $u$. First, $u$ is a leaf on the workflow tree, $Hop(u) = 0$. Since $u$ has no children, return flows will not be interrupted by the absence of $u$. Second, $u$ is a relay node, $0 < Hop(u) < H$. Suppose $v$ is the father of $u$, and $w$ is one of $u$'s children, as shown in Fig. 3 (b). When $u$ fails, $w$ just needs to return his results to $v$. However, for the consecutive node failure, assuming $u$ and $v$, GFC structure cannot tolerate it, because both of $w$'s return paths are broken off. ■

### B. Proof of Property 2:

*Proof:* We regard any two return flows with the same source node as common-source links. For broken links, there are two cases: (1) purely relay node-related link failure, and (2) the root-related link failures. In case 1, consider any three successive nodes $v$, $u$, and $w$, as shown in Fig. 3 (b). $Hop(v) = Hop(u) + 1 = Hop(w) + 2$, $Hop(w) \geq 0$, and $Hop(v) < H$. Since there are always two return paths from a node to its ancestors, for any single link failure , the source node of a failure link can always return its result via the alternative path. For any non-common-source link failure, the broken links could also be a pair: (I) $w \nrightarrow v$ and $u \nrightarrow v$, or (II) $w \nrightarrow u$ and $u \nrightarrow v$. Assume that $y$ is $w$'s child and $x$ is $v$'s father. In subcase (I), the alternative path will be $w \rightarrow u \rightarrow x$; in subcase (II), the alternative paths will become $y \rightarrow u \rightarrow x$ and $w \rightarrow v$. In case 2, let $x$ be the root, where $Hop(x) = H$, and nodes $u$, $v$, and $w$ be its children. In GFC structure, $x$'s children will create the backup paths, assuming they are $u \rightarrow v$, $v \rightarrow w$, and $w \rightarrow u$. Again, for any single link failure, the source node of a failure link can always use the backup path. If both $u \rightarrow x$ and $v \rightarrow x$ break off, the results can still be returned via $u \rightarrow v \rightarrow w \rightarrow x$; if both $u \rightarrow x$ and $v \rightarrow w$ fail, the second path will be $u \rightarrow v \rightarrow x$; if $u \rightarrow v$ and $v \rightarrow w$ are off, we still have $u \rightarrow x$, $v \rightarrow x$, and $w \rightarrow x$ to the job owner. ■

### C. Proof of Theorem 1:

*Proof:* In P-SNCA, node $u$ will never satisfy the return requirements if it has less than $c$ responded subtrees. However, in GFC-SNCA, $u$ may be able to receive $c$, or more than $c$, replies from its grandchildren, unless $u$ keeps sleeping. ■

### D. Proof of Theorem 2:

*Proof:* Let the node conditions making $P(u) > 0$ be set $\{P(u) > 0\}$, and the conditions leading $Q(u) > 0$ be set $\{Q(u) > 0\}$. According to both algorithms, if P-SNCA is able to return, then under the same condition, GFC-SNCA can also successively give results back, but not vice versa. So, $\{P(u) > 0\} \subsetneqq \{Q(u) > 0\}$ for $\forall u$, $Hop(u) \in [2, H]$. Hence, for $\forall u$, if $2 \leq Hop(u) \leq H$, then $Q(u) > P(u)$. ■