Progressive or Conservative: Rationally Allocate Cooperative Work in Mobile Social Networks

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

***Abstract*—There are plenty of idle computational resources on the Internet, which could potentially be used for accomplishing huge tasks. More and more applications are being designed for exploring those idle resources. In this paper, we focus on the idle computational resources, including both human intelligence and machine computing abilities, in mobile social networks (MSNs). Based on the unique features of MSN, we design a new cooperative system, called social-crowdsourcing. The distributed and infrastructure-free features of the system make it more attractive than traditional crowdsourcing platforms. In the proposed system, a huge work is gradually partitioned into smaller pieces, and is propagated from node to node. However, how to partition and allocate these segments is a critical problem, which directly affects the work's completion time and system throughput. Due to the lack of global information, independent relay nodes are likely to make conflicted decisions, which will cause an unbalanced workload distribution on participating nodes. In this paper, we find that, for a work at different processing stages, one should adopt distinct workload exchanging schemes, moving from a progressive method to a conservative one. Based on this observation, we propose an adaptive workload allocation scheme, in which a participating node can gradually switch his decision strategy according to the workload statuses of neighboring nodes. By using our approach, system throughput can be significantly improved, and large works can finish within a nearly optimal time. Unlike in traditional scheduling problems, we take a human's rejection, contact delay, and social similarity into consideration. Extensive simulation results show that our proposed algorithms can successfully make full use of the idle resources in MSNs.**

***Keywords*—Mobile social networks, outsourcing, potential resource, social-crowdsourcing, work partition.**

I. Introduction

Around us, plenty of computing resources have been wasted, including both human intelligence and machine computational abilities. For electronic devices, such as smartphones or tablet computers, we may leave them turned on for a whole day without doing anything to them; even when we are using them, only a portion of the machine's computational resources have been utilized, let alone the human intelligence that is wasted each day. In order to make better use of these idle resources, several centralized crowdsourcing projects have been implemented, such as Boinc [1], Folding@home [2], and Amazon MTurk [3]. In these projects, a project owner uploads a large and time-consuming task onto a server in advance [4], and volunteers participate in certain parts of the task when they are idle. The tasks may relate to human intelligence, such as seeking an object from images, or the tasks may simply use the computational resources of idle machines, such as training a model or solving complex mathematic functions.

However, there are two constraints, which restrict the development of crowdsourcing systems. First, the existing crowdsourcing platforms, such as Amazon MTurk, lack an advertising mechanism to timely recruit participants (also known as workers): It is hard for a newly created task to attract enough participants in a relatively short time, unless the task owner gives a very attractive payment. Many off-line workers, who are eager to do certain types of tasks, are not able to be timely aware the existence of the new tasks. Moreover, plenty of people do not even know of the existence of certain platforms, let alone the tasks on them. Secondly, the current system is centralized and platform-specified, and therefore, it is not flexible and robust enough. The workers, accustomed to one platform, are less likely to participate in another platform's tasks, and the unavailability of these platforms will completely destroy most of the existing crowdsourcing applications.

In this paper, we create a distributed and self-organized crowdsourcing scheme within mobile social networks (MSNs), called *social-crowdsourcing*. The main idea of the social-crowdsourcing scheme is that, after recruiting a worker, the task not only gets the workers' abilities, but also the resources potentially contributed by the worker's related people. Our scheme creates a multilayered outsourcing structure, and explores the idle computing resources within *social domains*. Instead of waiting for others to log in to a crowdsourcing platform, select your tasks, and work on them, our system directly sends the task to the potential workers via multi-hop social contacts. By using the scheme, users can self-organizedly build up a crowdsourcing system for a task, or it can also be used as an extension of workers in conventional crowdsourcing systems.

Example task in social-crowdsourcing: One may meet the following situation: when thinking about a problem, we recall that we have read a paper mentioning an idea, which could be used. But we do not remember which article it is. Although we have saved all of the papers in electronic documents, finding the article is still hard since that idea may be semantically described. Solely going through all of documents' contents is extremely time-consuming; instead, we can recruit friends for seeking the article.

The procedure of social-crowdsourcing: A task owner (a mobile device's user) first creates a social-crowdsourcing task by including both the electronic documents and job description. Once it is done, the owner becomes the first worker and begins to locally search the article. Note that the task owner could be the owner of the documents, or a worker, who undertakes a portion of a task from the conventional crowdsourcing platform. In social-crowdsourcing, any participant can

further recruit new workers via any kind of social contacts. For example, when a worker physically comes across his social contactor, such as friends or colleague, the participant's device will automatically send a message to the contactor and ask his willingness to participate in the task. If the answer is yes, a portion of the documents will be transferred from the worker to the contactor via shortwave radio, and then, both of them can go through the papers in parallel. When a participant is able to access free Internet, such as in vicinity of a free WiFi access point, or the participant agrees to use cellular networks to transmit social-crowdsourcing's workload, he can directly send the helping message or transfer a certain amount of workloads to his friends via network-storages, such as email, instant messaging (IM), or dropbox. Later, when his friends get Internet connection (e.g. 3G/4G or free WiFi), their devices will automatically fetch the data. When a participant finds the result, he will return the article's id to the task owner via the cellular network. Note that only a few participants need to return, and the size of the package is very small.

Remuneration: For workers' payment, we adopt MIT DARPA Challenge Team's scheme: the winner who finds the article gets half of the total bonus; the worker, who invited the winner, gets one fourth of the total; the people who invited the inviter gets one eighth, and so on. The main advantage of this incentive scheme is that users, who do not want to personally seek the paper from the documents, are still willing to participate and propagate the task to their social contactors.

During work segments' dissemination processes, the estimation of assigned workloads to each social contactor is a critical problem. Essentially, a selected allocation algorithm directly affects not only the whole work's completion time, but also the resource utilization rate of the social-crowdsourcing system. In this paper, we address the following question: given huge works originating at random nodes, by what strategy can the works' segments be appropriately disseminated during stochastic contacts, such that the system's throughput can be maximized? Due to the distributed feature of our system, this problem is not trivial. Take Fig. 1 as an example. When A meets C, A should leave some extra workload to C, such that C can forward these extra work segments to his future encounter D at an earlier time. When A transmits work segments to B, he must also count B's potential workloads given by B's other friends. In short, a worker has to take others' future potential contactors, accepting probabilities, contacting delays, current workloads, processing speed, and the impacts of common friends into account.

To solve the problem, we first propose a distributed workload allocation algorithm based on the historical information of worker's 1-hop neighbors. Consider the fact that, no matter who owns the work and how large the work is, a worker's overall computing ability should be a fixed value; we further propose a distributed algorithm for pre-estimating each node's future computing ability. By comparing the work-completing progress patterns of these two algorithms with those of the optimal ideal case, we finally propose an adaptive solution. The adaptive scheme automatically switches the work allocation methods according to the work segments' propagation status. Our solutions not only work under the single work, single source condition, but also are applicable in the situations of multiple works and multiple sources. Extensive simulation results show that our proposed schemes can significantly increase the system's overall throughput.

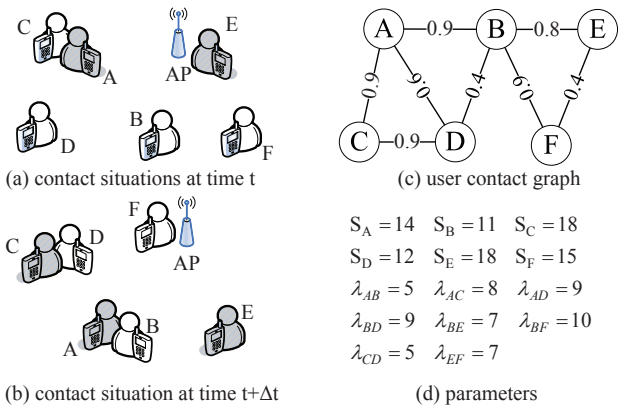


Fig. 1. Workload allocation in social-crowdsourcing. Shadowed users indicate the ones carrying work segments. Idle users are represented by the color white.

II. System Model and Problem Formulation

A. System Model

We consider a typical MSN, which consists of M mobile users (also named nodes). Let V_i represent a node. Each node is associated with a constant processing speed S_i, and we assume that, once a node participates in a task, it will not stop until the task has been finished. Social-crowdsourcing involves multiple rounds of recruiting via social contacts: a user's workload will be outsourced to his friends, friends' friends, and so on. Essentially, a huge work is gradually partitioned and propagated from friend to friend. In our model, a work consists of several work segments, and a work segment is the smallest uncleavable data unit. The workload of a node is the total number of work segments that have been assigned to it. Any user could be a task's owner (also called a source node), and more than one task could exist in the system.

In social-crowdsourcing, information exchanges during any kind of social contacts: locally or remotely. For local contacts, two nodes should physically encounter each other (like nodes' contacts in DTN), and data is transmitted via shortwave radio; remote contacts indicate the situations that two remote friends communicate with one another via cellular network or free WiFi-based Internet. In practice, data transmission of remote contacts can be implemented by using network storages, such as email, IM, or dropbox. When a worker contacts his friend for the first time after participating in a task, the friend can determine whether to join in the task. We use P_{ij} to represent the accepting probability. Once agreed, their devices will automatically adjust the workloads whenever they contact other participants of the task, until it is completed. For any worker, he may physically encounter other participants, stochastically. For the participants, who agree to use cellular networks at any time, they may query the working progress of their friends and adjust the workloads, once in a while. As for the workers, who use free Internet to transmit workload, they also can do same things whenever they get free Internet. Clearly, the contacting interval between a pair of workers is a random variable, and we use λ to represent the average inter-contacting time. Table I summarizes the common symbols used in this paper.

TABLE I. COMMON NOTATIONS

Notation	Description
G	nodes' social contact graph, $G =< V, E >$
V_i	a participant (user/node)
M	total number of nodes
S_i	user V_i's local computing speed
$S_i^{(p)}$	user V_i's future potential ability
P_{ij}	the accepting probability between V_i and V_j
$Q_{ij}(t)$	encounter (contact) probability between V_i, V_j within t
λ_{ij}	average inter-meeting time between V_i, V_j
$N(\cdot)$	the neighbor set of V_i on contact graph
$E(\cdot)$	the expected value
W	wasted computing resources
ΔW	exchanged workload
Φ_i	the assigned workload on node V_i
A	workload transition matrix $A = [A_{ij}]$
L	the size of the original work

Note that we cannot let all friends of a worker share one network storage "box" due to the security and trust reasons. Social-crowdsourcing has a concept of commitment: at any time, a work segment only belongs to one worker. If we put the workload in one box and let the worker's friends to fetch the data based on their dynamic needs, some workers may maliciously modify others' work segments in order to prevent them from winning the bonus (the friends of a user may not be friends). In order to avoid using complex security mechanisms, we do not let multiple users share a network storage.

B. Problem Formulation and Challenges

Social-crowdsourcing is a distributed system. From a source node, how to allocate the segments of a given work during stochastically contacts becomes a fundamental problem, which directly influences the completion time of the work. *The goal of the paper is to opportunistically maximize the overall throughput of a social-crowdsourcing system.* Since work segments are transmitted hop-by-hop, at each instance of segment relay, a relay node must estimate how many workloads he will keep, and how many workloads should be forwarded to the social contactors. The optimal allocation happens when (1) each participating node begins to work as early as possible and (2) all participants complete their assigned work at the same time. However, due to the lack of real-time global information and the accurate future encountering times, it is impossible for nodes to provide an optimal allocation.

The social-crowdsourcing system can be abstracted as a stochastic bucket network: each node is a huge bucket and there is a hole at the bottom of each bucket. The size of the holes may be different (which means the computing speed of each node may be different). Stochastically, we can pour some amount of water from one bucket to another (which models the workload reassignments during social contacts). Initially, only one bucket has water (the original large job), and we want to find a local scheme to determine the amount of transmitted water such that the water in the whole system can be drain as quickly as possible. For the rest of the paper, we will not discriminate how a social contact is conducted (locally or remotely), since all types of social contacts can be abstracted as stochastic data exchange between friends.

In order to estimate the amount of transmitted workloads during social contacts, the relay nodes must be able to answer the following two questions: 'how to describe a node's local computing ability and the overall abilities of his future encountered nodes?' and 'how to estimate the carrying workloads of the future potential contactors?' Since segments are disseminated based on multi-hops relay, the participants must carry an appropriate workload for their future contactors.

The estimation of future encountered nodes' capacities is difficult, not only because of the uncertainty of their contact times and acceptance decisions, but also the double counting of the abilities of their common friends. For instance, Fig. 1 (c) is a possible contacting graph among six users. When V_A forwards workloads to V_B, he should not only consider V_B's capacity, but also the potential workloads given by V_D, V_E, and V_F; otherwise, V_B may overload. Also, when V_A forwards workloads to V_C, he should make a decision on whether to give V_D's work segments to V_C for further relay. Obviously, V_A could also keep the segments and directly forward them to V_D later, or give them to V_B. Even if V_A selected a relay path to V_D with the highest probability, the realistic shortest contact path could be another one. Note that our problem is essentially different from the conventional delay-tolerant networks (DTN) studies. In our problem, any idle node could be a work segment's consumer. Since the optimal result happens when all workers simultaneously complete their workloads, we should avoid the unbalanced workload distribution among workers, which has never been considered in DTN.

III. SOLUTION OVERVIEW

A. Key Observations

Compared with traditional online social networks and delay-tolerant networks, MSNs have several unique features, which influence the performance of a solution. In this part, we present some important features of MSNs on our problem.

The number of participants, in the initial phase, highly influences the whole work's completion time. Roughly speaking, the number of participants at this phase affects the speed of work segment propagation. This observation is consistent with the research results in the field of maximizing social influence. In our model, work segments are disseminated to participating nodes via multiple relays. If a node does not carry enough work segments, he may locally process all the segments before contacting other idle nodes. Progressively estimating the abilities of a new participant and its future contactors could reduce the likelihood of this condition happening, but could also cause the imbalanced distribution of work segments.

In social-crowdsourcing, it is inevitable to have multiple segment-relaying flows pass the same node. However, whether the confluence is beneficial or not is determined by whether the situation could potentially reduce the work's completing time. During the initial phase of segment propagation, flow convergence may result in overloads, while, at a later time, it also provides chances for re-balancing the workload.

Due to the lack of real-time global information, it is extremely hard to estimate the future potential encounter nodes' carrying workloads. However, the amount of transferred workloads essentially relates to the workloads' differences between a pair of contactors, instead of their absolute workloads.

B. Main Idea

In this paper, we propose an adaptive scheme for allocating work segments. Instead of assigning the accurate workload to a

node on the first try, we *fuzzily* divide the whole process into three phases. Phase one intends to assign an approximately corrected amount of workload onto each region; phases two is used for dynamically re-balancing the workloads within small regions, and the final phase ends the work by opportunistically transferring the unbalanced workloads to idle nodes.

In the first phase, since a majority of nodes have not obtained the work yet, we use *an aggressive scheme to estimate nodes' overall computing abilities*, including the potential abilities from the future contactors. Therefore, for each node, we consider a *local computing speed* and a *future potential speed*, which indicates how much help a node could potentially obtain from future opportunistic contactors. Since there are time delays and rejecting probabilities at each time of transmission, computing the exact value of the potential speed is inappropriate and impossible. Considering that the aggressive scheme cannot provide a fine adjustment of workloads, instead, we take the overall speed that a node could get via the opportunistic contacting paths as the criteria. This metric can integrate transmission delay and accepting rate together, and avoids the double-counting problems caused by common friends. Although the work segments are physically propagated via random paths, our metric essentially computes the expected sum of speed that the whole system could provide to the node.

As for the second phase, when two users contact each other, they will use their stored neighbor's historical workload status information to estimate the work's finishing time, and will further make workloads' adjustments based on it. The second phase is the transition phase. The intuition behind this mechanism is that, once each node's neighbors have the same completion time, the workloads are balanced and distributed among all participating nodes. By exploring the two contactors' direct neighborhood information, one extends his local view to 2-hop. Based on an estimated completion time distribution, a node proportionally assigns weights to the aggressive scheme and the conservative scheme.

The locally stored historical information used in phase two is collected during a pairwise contact. It may have already become outdated before making a workload allocation decision; some nodes could complete their assignments earlier than others. At the final phase, nodes simply transfer some unfinished work segments to the randomly encountered contactors. The method, used in this phase, purely considers each node's 1-hop neighborhood status and that of the current contactors.

IV. Solution Details

A. Contact Frequency and Contact Graph

We assume that the inter-contacting time t between any two nodes V_i and V_j follows exponential distribution with the pairwise contact rate λ_{ij} ($\lambda_{ij} > 0$). The contacting probability density functions (PDF) are represented as follows:

$$Q_{ij}(t) = \lambda_{ij} \times e^{-\lambda_{ij} \cdot t} \tag{1}$$

where $Q_{ij}(t)$ stands for the contacting probability between nodes V_i and V_j within time interval t. Assume that there is a r-hop transmitting path from node V_i to node V_r with edge weights $\lambda_{i,1}, \lambda_{1,2}, \cdots, \lambda_{r-1,r}$. According to [5], the total transmitting time follows hypoexponential distribution:

$$Q_{ir}(t_1 + t_2 + \cdots + t_r) = \sum_{s=1}^{r} C_{s,r} \times \lambda_{s,s+1} e^{-\lambda_{s,s+1} \cdot t} \tag{2}$$

where $C_{s,r} = \prod_{u \neq s, s=1}^{r} \lambda_{u,u+1} / (\lambda_{u,u+1} - \lambda_{s,s+1})$ and $t = t_1 + t_2 + \cdots + t_r$.

After iteratively exchanging information with friends, nodes learn the system parameters within r-hop, such as computing speed, average accepting rate, and average inter-contacting time. Based on this information, a contact graph will be locally created on each node, which records the identities of a node's contactors, together with the system parameters. Besides the system parameters, each node may also store certain auxiliary information about its direct contactors, such as a node's potential ability (which will be introduced in Section IV. D.) Note that, in our system, a node may also record others' expected completion times. However, such values are not in real-time, and the node can only learn distant nodes' expected completion times via multi-hop contacts. For instance, in Fig. 1 (c), node V_A never contacts V_F, but it can learn V_F's expected completion time via node V_B's historical estimation. After encountering with V_F, V_B may estimate its finishing time based on the carrying workload at the contacting time, and record the value together with a time stamp. When V_A meets V_B, he could update his record about V_F by using the newest historical estimation.

B. Neighbor Status-based Finishing Time Estimation (NSFT)

In our system, after each instance of pairwise contacts, a node is able to estimate the work's finishing time within its 1-hop neighborhood. The estimation of finishing time is not trivial, due to the fact that nodes may adjust their workloads among other neighbors at any time. Therefore, the recorded carrying workload of a neighbor node may have already been changed at the time of estimation.

Algorithm 1 shows our procedure for approximating the finishing time. Without loss of generality, we assume that Algorithm 1 is running on node V_u, and node V_v is the current contactor of V_u. Since a user V_i could be the common contactor of both nodes V_u and V_v, the first step of Algorithm 1 is to eliminate the double counting problem. We virtually split the common contactor V_i into two virtual nodes, and use its accepting probabilities to weight each virtual node, as shown by Algorithm 1, line 5. Essentially, line 5 splits the carrying workload and the local computing speed according to the corresponding accepting probability, and exclusively virtually assigns a portion of the carrying workloads and speed to V_u and V_v, respectively. The advantage of this approach is that, after virtually splitting, the completion times of the two virtual nodes are still equal to those of V_i. Via this way, node V_u is able to rationally estimate the real possible impacts of his neighbors' workloads and computing abilities.

Although workload adjustments may happen at any time, here, *we are only interested in a set of time periods, which could potentially come at the cost of wasting computing resources*. More specifically, within node u's 1-hop neighborhood, computing resources will be wasted if workload adjustment happens during the condition that some neighbors have finished all of their workloads while others are still

Algorithm 1 Neighbor Status-based Finishing Time Estimation (NSFT)

1: /*Assume NSFT is running on u and the other encounter is v*/
2: Find u and v's common neighbors $N(u) \cap N(v)$
3: **for** $\forall i \in N(u) \backslash v$ **do**
4: **if** $i \in N(u) \cap N(v)$ **then**
5: Split the recorded workload Φ and speed S of the common neighbor of u and v: $\Phi_i \leftarrow \Phi_i \times \frac{P_{iu}}{P_{iu}+P_{iv}}$, $S_i \leftarrow S_i \times \frac{P_{iu}}{P_{iu}+P_{iv}}$ /*Eliminate the double counting problem of the common neighbors of u and v*/
6: Based on u's local record, estimate each neighbor's finishing time $\{\Phi_i/S_i\}$, and Sort $\{\Phi_i/S_i\} \cup \Phi_u/S_u$ in ascending order
7: **for** $\forall i \in N(u) \backslash v$ **do**
8: **if** $\Phi_i/S_i < \Phi_u/S_u$ **then**
9: Compute the expected wasted computing resource W according to Equation (3)
10: /*Wasted resource exists when u is working while i is idle*/
11: **if** $\Phi_i/S_i > \Phi_u/S_u$ **then**
12: Compute the wasted resource W_u by Equation (3), and update a temporary variable tp: $tp \leftarrow \min\{tp, W_u/S_u\}$ /*Resource is wasted when u is idle while the neighbors of u are working*/
13: $W \leftarrow W + (\sum_{\forall i, i \in N(u)\backslash v, \Phi_i/S_i < \Phi_u/S_u} S_i + S_u) \times tp$
14: Estimate the finish time t_F: $t_F \leftarrow [\sum_{\forall i \in N(u)\backslash v} \Phi_i + \Phi_u + W]/[\sum_{\forall i \in N(u)\backslash v} S_i + S_u]$

working. After getting an approximated value of the wasted computing resources, u can estimate his finishing time by first summing it up with the real workloads, and then dividing the result by the summation of weighted speeds, as shown by Algorithm 1, line 14.

From the viewpoint of V_u, he only needs to care about the workload adjustment between him and his neighbors. For the neighbor node $\{V_i\}$, who will finish its current workload before V_u (Algorithm 1 line 8), we can estimate the wasted computing resource at a node V_i, W_i, as the product of V_i's computing speed and the expected time delay after V_i finishes the work. Assume t_{next} is the next contacting time between nodes V_u and V_i, then we have:

$$\begin{aligned} W_i &= S_i \times E[(t_{next} - \Phi_i/S_i), \Phi_i/S_i < t_{next} < \Phi_u/S_u] \\ &= S_i \times \int_{\Phi_i/S_i}^{\Phi_u/S_u} (t - \Phi_i/S_i)\, \lambda e^{-\lambda t}\, dt \end{aligned} \quad (3)$$

For each node V_i, who completes the corresponding works before V_u, we need to calculate a W_i and sum them up (Algorithm 1 line 9). However, as for any node V_j, who finishes his works after V_u (Algorithm 1 line 11), only V_u's wasted computing resources will be considered (since V_u records only 1-hop information, the wasted resources during the workload propagation directly between V_u's 1-hop neighbors are ignored.) Therefore, for the node group with a completion time larger than V_u, we only need to find the earliest completing time: after this time, V_u may get new work segments again (Algorithm 1, line 12.) One may notice that, when V_u finishes his workloads and is in idle status, the neighbors with earlier completing times must also be idle. So, we also estimate their wasted resources during the period from V_u completing his workload to getting new segments (Algorithm 1, line 13).

Algorithm 2 Local 1-hop Status-based Workload Adjustment

1: /*Assume it is running on u and the other encounter is v*/
2: Get v's 1-hop information from v
3: $\Delta W \leftarrow |\Phi_u - \Phi_v|/2$
4: **while** $NSFT(u, \Delta W) \neq NSFT(v, \Delta W)$ **do**
5: **if** $NSFT(u, \Delta W) > NSFT(v, \Delta W)$ **then**
6: **if** $\Phi_u > \Delta W$ **then**
7: $\Phi_u \leftarrow \Phi_u - \Delta W$, $\Phi_v \leftarrow \Phi_v + \Delta W$
8: **else**
9: $\Delta W \leftarrow \Phi_u$
10: **else**
11: **if** $\Phi_v > \Delta W$ **then**
12: $\Phi_u \leftarrow \Phi_u + \Delta W$, $\Phi_v \leftarrow \Phi_v - \Delta W$
13: **else**
14: $\Delta W \leftarrow \Phi_v$
15: $\Delta W \leftarrow \Delta W/2$
16: Send/receive ΔW amount of workload to the other contactor

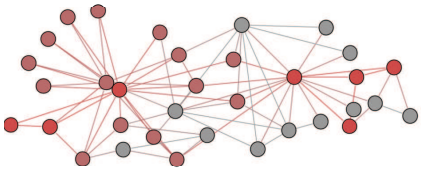


Fig. 2. Contacting graph of example data set.

C. *Local* 1*-hop Status Information-based Workload Allocation*

Whenever two users contact each other, they could reallocate their workloads such that the work can be completed at the same time. Intuitively, after exchanging and updating their recorded neighborhood information, each node can figure out an ideal size of the transferred workloads by using NSFT algorithm. Here, we adopt the binary search algorithm to estimate the size. Note that, in our model, both contactors could run the NSFT algorithm and get the same value of the transferred workload, ΔW. Algorithm 2 gives the procedure.

In order to check the effectiveness of this 1-hop information-based algorithm, we compare it with the optimal result and the naive result, respectively. In the naive case, after two nodes meet each other, workloads are reallocated according to their local processing speed. The optimal result is based on posteriori knowledge: after events have happened, all nodes' contacting times and accepting decisions are recorded and known. Based on this information, the optimal scheme simply finds out the fastest relay paths from the work owner node to any other participants. The optimal scheme calculates the earliest work segment arriving time of each node, and then, assigns an appropriate amount of workloads to the participants, such that they will finish the job at the same time.

Figs. 3 to 7 illustrate the workloads' average diffusion patterns on a regular contacting graph (as shown by Fig. 2) with the growth of observation time. The contacting graph is based on karate club social graph [6]. When generating these figures, we keep each node's acceptance decision fixed, and repeat the simulation 10 times by randomly creating different encountering times. On these graphs, the intensity of the gray color at position (x, y) represents the carrying workload on node y at time x. We use the absolute work amount to represent the unfinished workloads being carried on each node. The darker a line's color is, the more workloads that are held

by a node. If a node becomes idle, we use white color to represent it. The work's processing progression is shown in Fig. 8. In order to more accurately describe the contributions of different schemes, we also consider an extreme social contacting graph, where the nodes form a chain and each node only has contact with the previous two nodes and next two nodes. The simulation results of this special contacting graph are given by Figs. 9 to 14.

From Fig. 3, we can see that, under the naive scheme, the work segments do not fully spread. Many nodes barely have work segments, and for some other nodes (such as nodes 1 to 5), they have not obtained appropriate amount of workload: there are many white-colored intervals between the gray one. From the x-axis, we can see that it takes almost 1200 units of time to finish the work. In Fig. 8, the slope of the brown-diamond curve indicates the overall processing speed by using the naive scheme. In this example, since nodes fail to transfer an appropriate amount of the workload to other nodes, only a limited amount of computing resources is used. Moreover, during the simulation, we also find that the completion time of this scheme is unpredictable, and is influenced by the contacting sequences. As for the optimal result, shown by Fig. 7, we can see that, during the initial phase (the first 200 units of observation time), work segments are quickly spread among nodes. As we have mentioned, in the optimal result, all participants complete their work at the same time.

Compared with the naive scheme, the 1-hop status information-based algorithm (Fig. 4) can quickly disseminate the workloads during the initial phase; however, from a long-term perspective, the transferred amount at this phase is inappropriate, and therefore, some nodes are out of work quickly. By further checking the size of assigned workloads, we find that the scheme underestimates certain nodes' potentially idle abilities, especially during the initial phase. From the view of the percentage of completed work, as shown in Fig. 7, we observe a better result in which, the slop of the green-star curve is larger than the brown-diamond one. The simulation result on the chain-style contacting graph gives us a similar result. By comparing Figs. 11 and 12, we can clearly see that the 1-hop status information-based algorithm can better propagate work segments than the naive scheme. However, by observing the locations of the darkest points between Figs. 12 and 9, it seems the initial participants should transfer more work segments to the contactors.

D. Potential ability-based Workload Allocation

Influenced by the HITS algorithm [7], [8], which is used for rating Web pages, we propose an algorithm for measuring and ranking a node's potential future computing ability that the node could obtain by recruiting other nodes. Essentially, we let each node be associated with two features: a local computing speed and a potential speed. However, unlike the traditional link analysis problems, the links in our model are associated with both accepting rate and random contacting delay. For addressing this problem, we design a special weight on each link, which can successfully integrate the number of friends, contacting delays, and accepting decisions together.

1) Weights of Contacting Edges: For a node V_u, we consider the following situation: if all neighbors of V_u are idle and only V_u has a huge work with size L, what is the expected amount of workload that each of V_u's neighbors, V_v, can obtain? We use the NSFT algorithm, which is discussed in Section IV.B. However, the NSFT assumes that a pair of nodes has already met. In the current problem, nodes V_u and V_v may or may not come across each other. Similar to previous analysis, we need to measure the expected amount of wasted computing resources, $E[W_v]$, before V_u meets V_v and transfers workload to it. Here, we estimate the wasted workloads as $E[W_v] = S_v \times E[\text{inter-contacting time}] = S_v/\lambda_{uv}$. Let Φ_u and Φ_v be the workloads on nodes V_u and V_v after encounter. They should satisfy the following two conditions:

$$L = \Phi_u + \Phi_v + S_v/\lambda_{uv} \tag{4}$$

$$NSFT(\Phi_u) = NSFT(\Phi_u) \tag{5}$$

We use ρ_{uv} to represent the expected percentage of workloads that node V_v can get from V_u.

$$\rho_{uv} = P_{uv} \times \frac{\Phi_v}{L} \tag{6}$$

where P_{uv} is the accepting probability between V_u and V_v. Without loss of generality, we define ρ_{uu} as the following:

$$\rho_{uu} = \frac{E[L - \Phi_v]}{L} = \frac{\sum_{v \in N(u)} [P_{uv} \times (L - \Phi_v)]}{L \times \sum_{v \in N(u)} P_{uv}} \tag{7}$$

By normalizing ρ_{uv} for all of u's neighbors, we get a workload transition matrix $A = [A_{uv}]$.

$$A_{uv} = \frac{\rho_{uv}}{\sum_{v \in N(u)} \rho_{uv}} \tag{8}$$

For each node V_u, the computation of A_{uv} for each neighbor node V_v only requires node V_u's 1-hop information, and therefore, each node can locally compute the transition weight A_{uv} to each neighbor. Note that the transition matrix A is not a symmetric matrix, and $\sum_{v \in N(u) \bigcap V_u} A_{uv} = 1$.

***Theorem** 1:* The transition matrix, $A = [A_{uv}]_{M \times M}$, has a stationary distribution, $\pi_{1 \times M}$, such that $\lim_{k \to \infty} A^k = \mathbf{1}\pi$, where $\mathbf{1}$ is the column vector with all entries equal 1.

Proof: We can model the propagation process of a single work segment by a time-homogeneous Markov chain. Each user (node) is a state. The state transition probability is given by the transition matrix A, which is introduced in Section IV. D. According to book [5] (Theorem 4.1), a Markov chain has a unique stationary distribution if it is irreducible and aperiodic.

According to our scheme (Algorithm.3 lines 3 to 6) for generating the transition matrix A, all considered states are accessible by the algorithm executing node V_u. Since the communication between nodes is bidirectional, any state in A can access any other state by following a transition path through node V_u. Therefore, the Markov chain is irreducible.

In our model, the average inter-contacting time between any pair of nodes is greater than zero. Therefore, the value of A_{uu} is non-zero. A state in a Markov chain is aperiodic if the state's transition probability to itself is greater than zero. Since all states in our model have the self-loop, the Markov chain is aperiodic. Based on the above two features of the transition matrix A, the corresponding Markov chain has a stationary distribution $\pi = [\pi_1 \quad \pi_2 \cdots \pi_M]$. $\pi_j = \sum_{i=0}^{\infty} \pi_i \times A_{ij}$ $(j \geq 0)$, and $\sum_{j=0} \pi_j = 1$. ■

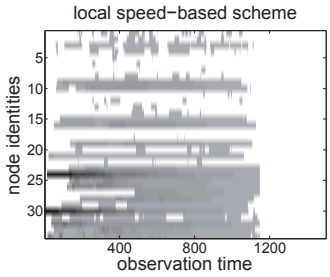


Fig. 3. Single work: case 1

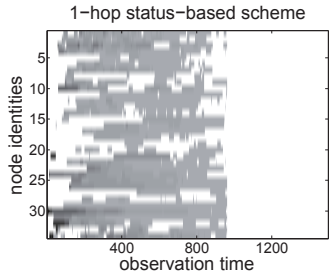


Fig. 4. Single work: case 2

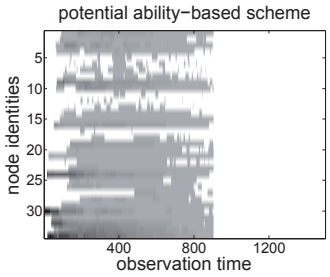


Fig. 5. Single work: case 3

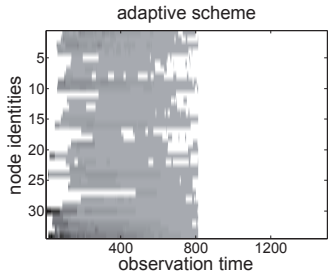


Fig. 6. Single work: case 4

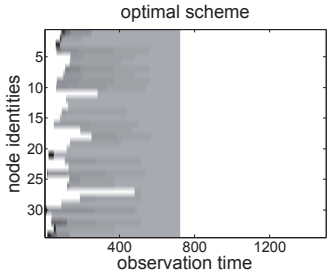


Fig. 7. Single work: case 5

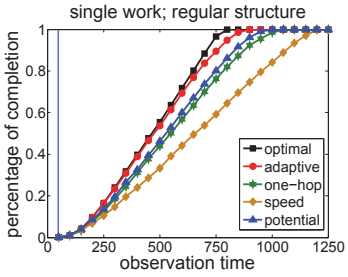


Fig. 8. Work progress comparison

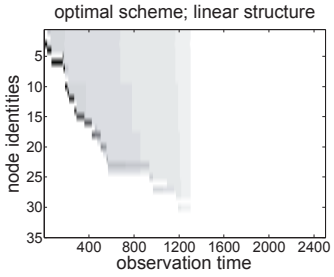


Fig. 9. Special structure: case 5

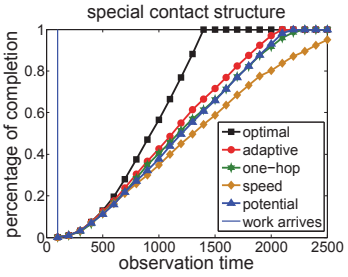


Fig. 10. Work progress comparison

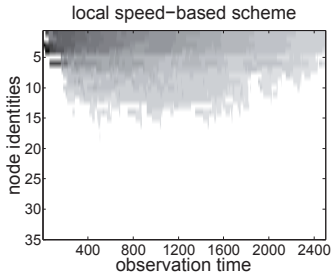


Fig. 11. Special structure: case 1

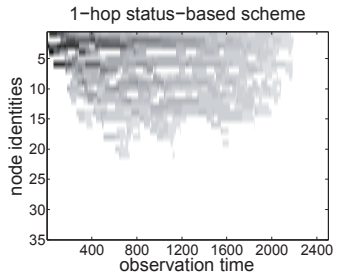


Fig. 12. Special structure: case 2

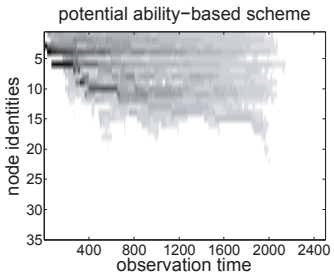


Fig. 13. Special structure: case 3

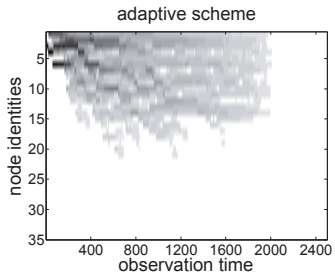


Fig. 14. Special structure: case 4

2) Estimation of Potential Ability: During the computation of node V_u's potential ability, both the local computing ability of a neighbor node V_v and its potential ability should be regarded as V_u's potential ability. However, on a social contacting graph, since each node has more than one contactor, a node must equally split its abilities among the neighbors. Here, we adopt A_{uv} as the weights for splitting. Let $S_u^{(p)}$ represent V_u's potential ability, then it can be recursively defined as following:

$$S_u^{(p)} = \sum_{v \in N(u)} A_{vu} \times \left[S_v^{(p)} + S_v \right] \tag{9}$$

where S_v is the local computing speed of a node, and A_{vu} is the transition probability from node V_v to V_u. One may note that the value of A_{uv} is related to the selected value of L. In reality, instead of computing a single value, we calculate a series of different values of $\langle L \rangle = \langle L_1, L_2, \cdots, L_k \rangle$, and weigh the corresponding results.

There are two common ways to compute the potential ability. The first approach is similar to the computing procedure of the HITS algorithm: each node may simply store a temporary score $S_u^{(p)}(t)$, iteratively exchange the recorded score with neighbors, and update the score by using $S_u^{(p)}(t+1) = \sum_{v \in N(u)} A_{vu} \times \left[S_v^{(p)}(t) + S_v \right]$. Ideally, after several rounds of exchange, $S_u^{(p)}(t)$ becomes stable. However, at the end of each iteration, the approach must normalize each node's recorded score by the total of the current scores; otherwise, the scores will not converge.

The second approach is based on random walk. By analyzing the definition function of $S_u^{(p)}$, we can see that the value of $S_u^{(p)}$ is essentially the sum of the computing speeds that each node can contribute to V_u along the opportunistic contacting paths. In the second approach, each node sends out several random walkers. Each walker is associated with a score, which indicates the amount of speed that the originated node can contribute to the walkers' current resident node. Whenever a walker passes a node, the resident node records the current score of the walker, and the walker randomly selects its next destination from the direct neighbors of the resident node. Note that, with each transition from node to node, the walk's carrying score will be reduced according to the edge weight, A_{uv}. A walker stops moving when its score becomes less than a pre-defined threshold. The potential ability is approximated by the sum of a node's recorded scores. Although the second approach is fully distributed, in reality, the whole process may take a long time and involve a lot data packages.

For simplifying the problem, we let each node gradually learn other nodes' A_{uv} and speeds via several rounds of node pairwise contacting. Each node locally creates the whole social contacting graph and computes the potential ability. One can

Algorithm 3 Potential ability-based Workload Allocation

1: /*Assume the algorithm is running on V_u*/
2: /*Generation of the edge weights in matrix A*/
3: **for** Every node $v \in N(u)$ **do**
4: **for** Every L_i in $\langle L_1, L_2, \cdots, L_K \rangle$ **do**
5: Compute $A_{uv}^{(L_i)}$ for each sampling workload size
6: Compute the weighted average of A_{uv} based on the size distribution of real works
7: /*Compute V_u and its neighbors' potential ability $S^{(p)}$*/
8: Learn other nodes' A_{uv} and S_v via pairwise contacting
9: Compute potential ability by equation 9
10: /*Workload allocation scheme on node u*/
11: Exchange and update state information with v
12: **if** V_u is contacting V_v and $\Phi_u + \Phi_v > 0$ **then**
13: $\Phi \leftarrow \Phi_u + \Phi_v$, $\alpha \leftarrow (S_u + \beta S_u^{(p)})/(S_u + S_v + \beta S_u^{(p)} + \beta S_v^{(p)})$
14: $\Phi_u \leftarrow \alpha\Phi$, $\Phi_v \leftarrow (1-\alpha)\Phi$

compute the value of $S_u^{(p)}$ either by directly using equation 9, or by first calculating matrix A's stationary distribution π and then computing $\mathbf{S} \times \mathbf{1} \times \pi$, where $\mathbf{S}_{1\times M}$ is the speed vector.

3) Workload Allocation: Once obtaining the potential abilities of direct contactors, a pair of encountering nodes can redistribute their workloads according to their potential abilities. Algorithm 3 shows the procedure for potential ability-based work segments allocation. The basic idea of Algorithm 3 is that, during the propagation of work segments, especially at the initial phase, if participating nodes can roughly estimate the overall ability of each work assigning flow, the work segments could be disseminated more uniformly. Note that the potential ability is a rough estimation about a participant's future computing capacity. In practice, one should consider both the potential ability and local speed by assigning certain weights to them (Algorithm 3 line 13).

Fig. 5 shows the diffusion process after using Algorithm 3. This approach has a better performance than does Algorithm 2, and it assigns more appropriate amount of work segments to the participants. From Fig. 5, we can see that a large part of participants (nodes 15 to 34) are assigned with more workloads within the first 400 units of observation time. The completion progression of a given work is presented in Fig. 8; From the very beginning, Algorithm 3 beats Algorithm 2. However, based on the algorithms, we know that the 1-hop status-based approach's local workload adjustment is better than that of the potential ability-based scheme: by further comparing Fig. 4 with Fig. 5, we can see that 1-hop status-based approach use more idle resources on nodes 5 to 15. However, in Figs. 12 and 13, we did not see such a difference. Probably due to the chain structure, a 1-hop status-based approach cannot equilibrate the workloads well. Based on the respective advantages of the 1-hop status-based scheme and the potential ability-based scheme, we wonder whether we could combine them together and get a better result.

E. Adaptive Scheme

Workload allocation process can be fuzzily partitioned into three phases. Initially, most nodes are idle, but their realistic accepting decision is unknown. In this phase, the large-scale propagation of workloads, in a balanced way, should be the first priority of an allocation scheme. As more and more nodes participate in the cooperative work, the amount of exchanged workloads gradually becomes smaller, and the undertaken workload of each node trends to stable. Therefore, at the initial phase, we allocate the workloads mainly based on each node's expected potential ability. Here, we adopt the potential ability-based workload allocation method (Algorithm 3.) Because most nodes have not participated in the new work, and the workload partition decisions are made based on the average condition, the workload allocation scheme used in this phase is more progressive than those of the later phases.

As time grows, the accepting decisions of a majority of nodes become clear. The workload allocation process gradually enters the second phase, which focuses on balanced distributing workloads among the real participating nodes. A typical feature of phase two is that all participating nodes are physically working on the same work's segments, simultaneously. Nodes in our model essentially perform two operations: processing work segments one-by-one, and re-balancing the remaining workloads among the contactors within several hops (these nodes virtually form a region). Although a certain amount of workload has been transmitted from the source node to each region, due to the uncertainty of nodes' accepting decision, the realistic workload distribution among different regions may not be balanced. Therefore, we combine the potential ability-based method with the local 1-hop status-based approach (Algorithm 2): the former approach helps to re-balance workloads between different regions, and the latter one re-allocates the workloads within each local region.

During the final stage, some nodes complete their work and become idle again, while some other nodes are still working. Opportunistically and locally re-balancing has the top priority. Consider the fact that the local 1-hop status-based approach holds more accurate neighborhood completion time information than the aggressive one. Hence, at the final stage, we focus on using the 1-hop status-based scheme.

Our adaptive solution basically combines the potential ability-based approach together with the local 1-hop information-based one. However, how to identify the three phases discussed above, and how to smoothly switch between the two approaches, is critical. Clearly, at the initial stage, the expected work completion times are highly unbalanced, where the expected finishing times for the initial participants are much higher than others. Meanwhile, at the final stage, the completion times are more uniformly distributed. Since there is not a clear boundary between each stage, we use the distribution of the expected completion times as an indicator, which is able to quantitatively describe how much closer the current system status is to the initial stage (a highly biased distribution) and the final stage (a uniform distribution). However, since the real-time global statuses are not available, the completion times' distribution is purely based on each node's local recording of the finishing time information about other nodes. As we have mentioned in Section IV. A, a node may learn other nodes' expected completion times by multiple rounds of information exchanging between different nodes.

We adopt the entropy of the completion times' distribution [9] as the metric. Let ΔW^p be the amount of transferred workloads by using the potential ability-based scheme, ΔW^l be the transferred workload by adopting a local 1-hop information-based scheme, and p_i be the percentage of nodes with completion time t_i. The transferred work amount in our

Algorithm 4 Time Adaptive-based Scheme

1: /*Assume the algorithm is running on node V_u*/
2: **for** Every contact between V_u and its neighbors V_v **do**
3: Exchange and update state information
4: Compute the completion time's distribution $\{p_i\}$
5: $\alpha \leftarrow 1 + \log^{-1}(M) \times \sum_{\forall i} p_i \log(p_i)$
6: Compute ΔW^p and ΔW^l by calling Algorithms 2 and 3
7: $\Delta W \leftarrow \alpha \times \Delta W^p + (1-\alpha) \times \Delta W^l$

adaptive scheme is determined as follows:

$$\Delta W = \alpha \times \Delta W^p + (1-\alpha) \times \Delta W^l \tag{10}$$

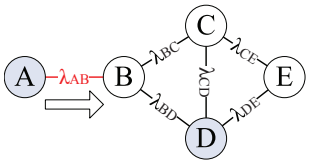
$$\alpha = 1 + \frac{\sum_{\forall i} p_i \times \log(p_i)}{\log(M)} \tag{11}$$

where α is a mixing parameter, and M is the total number of a work's participants. When $p_i = 0$, we assign zero to the computing result of $p_i \times \log(p_i)$. During the computing of α, we compare the entropy of the current completion times' distribution with that of the uniform distribution, which is equal to $\sum_M 1/M \times \log(1/M) = -\log(M)$.

Algorithm 4 shows the procedure of our adaptive scheme. Note that more than one work may coexist in our system. Since the goal of this paper is to maximize the system's overall throughput, here, we do not discriminate the segments from different works. Figs. 6 and 14 show the workload diffusion pattern by using the adaptive scheme. By comparing Fig. 6 with Figs. 4 and 5, we can clearly see that the adaptive scheme takes the advantage of both the potential ability-based scheme and the local 1-hop status-based scheme: in the initial stage, workloads are widely and appropriately disseminated among participants, and at the later stage, workloads are locally equilibrated among neighbors. In Fig. 8, the proposed adaptive solution is the best approximation algorithm. However, as for the simulation result on the special chain-style contacting graph, although the adaptive scheme still beats all other solutions, its excellence is smaller than that of the simulation on the regular contacting graph.

F. Example

Take Fig. 15 as an example. Assume that, at time t, nodes V_A and V_B come across each other for the first time (after the work arrived the system), and Fig. 15 (b) gives the parameters about participants. The carrying workloads of V_A and V_D are 1000, respectively. All other nodes have zero workload. After the contact between V_A and V_B, intuitively, V_A should transfer a large portion of workload to V_B, since V_B has more potential computing resources. However, if we assign workload only based on local speed, V_B can only get $0.01 \times 1000/(1+0.01) \approx 10$ units of workload. According to the current 1-hop information of V_B, the 1-hop status-based scheme first considers the available computing resources within V_B's neighborhood except V_A, and then it will assign 834 workloads to V_B such that the expected finishing times of V_A and V_B are 166 and 167. Note that the estimation of the finishing time also considers the impacts from the neighbors of V_B. If we adopt the potential ability-based scheme, the overall potential abilities of V_A and V_B are 1.87 and 8.49. However, we should not directly use these scores, since the scores count the potential speeds coming from both sides. By temporarily removing the edge between V_A and V_B, the mutual potential abilities of them become 0 and 6.62. As a result, V_B will get $(0.01+6.62) \times 1000/(1+0+0.01+6.62) \approx 869$ units of workload. Note that the computation of the potential speed is related with the size of work. Here, we present another simpler approach to compute the potential speed. After temporarily removing the edge between V_A and V_B, the potential speed of V_B can be estimated as $[\lambda_{BC} + (1-\lambda_{BC})\lambda_{BD}\lambda_{DC}]S_C + [\lambda_{BD} + (1-\lambda_{BD})\lambda_{BC}\lambda_{CD}]S_D + (\lambda_{BC}\lambda_{CE} + \lambda_{BD}\lambda_{DE} - \lambda_{BC}\lambda_{CE}\lambda_{BD}\lambda_{DE})S_E = 2.764$. By using this score, V_B will get $(0.01+2.764) \times 1000/(1+0+0.01+2.764) \approx 735$ units of workload.



(a) user contact graph

$S_A = 1$	$\lambda_{AB} = 1/5$	$\lambda_{DE} = 1/5$
$S_B = 0.01$	$\lambda_{BC} = 1/5$	$W_A = 1000$
$S_C = 1$	$\lambda_{BD} = 1/10$	$W_D = 1000$
$S_D = 10$	$\lambda_{CD} = 1/5$	$W_B, W_C, W_E = 0$
$S_E = 30$	$\lambda_{CE} = 1/10$	

(b) parameters

Fig. 15. A simple example about workload allocation in social-crowdsourcing. Shadowed users indicate the ones carrying work segments. Idle users are represented by the color white.

G. Further Discussion

In reality, it is possible that a work has more than one owner. For example, a study group possesses a certain amount of data, which needs to be processed. Each member of the group could be the work's owner. In order to guarantee that the work segments that originated from different members have the same work identity, the group may choose the smallest identity of its group members, and concatenate it with the releasing time as the work's identity. For ease of description, we call such cases *multiple sources conditions*. We can assume that there is a virtual node, which only has contact with each of the group members. Since the workload allocation process at all nodes, in multiple sources conditions, is the same as that in the single source condition, our proposed adaptive workload allocation scheme can still function well. Essentially, the multiple sources conditions just shorten the length of the initial propagation phase. Since the work segments' diffusion patterns are similar to the patterns of a single source condition, we do not provide the illustration figures.

Any node could be the owner of a work. Therefore, it is possible that different works coexist in our system. Clearly, if there is always a large time gap between the arriving times of different works, our schemes definitely work. However, it is also possible that some works arrive while other works have not been completed yet. Clearly, the coming of new works may cause an unignorable difference between the recorded workload statuses of other nodes and their real conditions. However, since the goal of our paper is to maximize the system's overall throughput, instead of minimizing each work's completion time, all of our discussed schemes do not need to discriminate the segments of different work.

Note that, when multiple works coexist in a system, minimizing the works' completion time becomes a very difficult problem. The segments from different works inevitably will compete for the computing resources. Moreover, since social-crowdsourcing is a fully-distributed system, and the makespan of a work is determined by the finishing time of the last

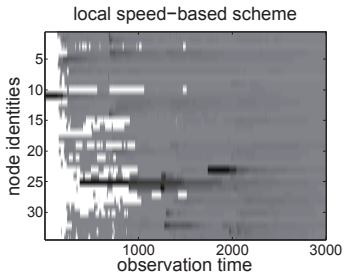


Fig. 16. Multiple works: case 1

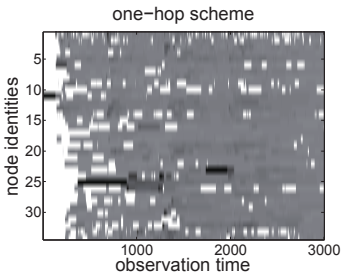


Fig. 17. Multiple works: case 2

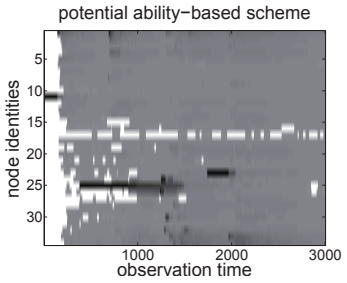


Fig. 18. Multiple works: case 3

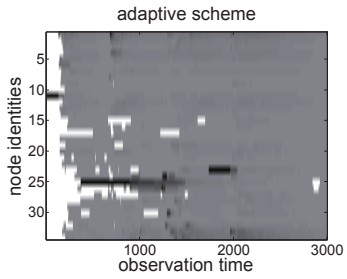


Fig. 19. Multiple works: case 4

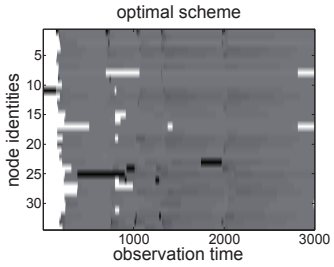


Fig. 20. Multiple works: case 5

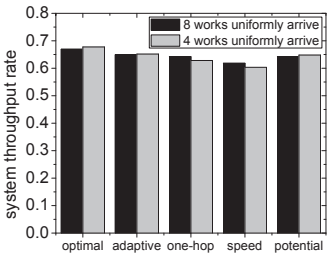


Fig. 21. Throughput rate comparison

segment, it is very hard for a node to appropriately allocate the computing resources to different works based on the node's local view. One possible research direction is to explore the backpressure routing algorithms [10]–[12]. However, unlike the traditional routing problems, social-crowdsourcing does not have the concept of message destination. We leave the problem for minimizing the completion time as our future work.

Figs. 16 to 20 illustrate the work segments' diffusion patterns under different work allocation approaches. During the simulation, we randomly create 4 works with a relatively large size. However, unlike the results in single source condition, the work's processing progression curves are closer to each other. In order to clearly observe the differences between schemes, instead of drawing the processing progression graph, we consider the system throughput rate by each scheme, as shown by Fig. 21. From the simulation results, we find that, comparing to other schemes, the coexistence of multiple works has a negative effect on the 1-hop information-based solution. One possible explanation is that, among all solutions, the 1-hop information-based solution highly depends on the recorded workloads of neighbor nodes; due to the arrival of new works, the workload status information changes much more frequently. When the changing rate exceeds the records' updating rate about a majority of neighbors, nodes will make wrong workload allocation decisions, which could also result in a chain reaction. Although the adaptive scheme also uses the historical information, it still has the best performance. The reason is that, whenever a node receives a statue of information indicating the existence of a new work, the node will mainly use the workload allocation decision based on potential ability. Therefore, the chain reaction of the wrong decisions is avoided.

V. Performance Analysis and Evaluation

In this section, we conduct extensive simulations to evaluate the performances of our proposed algorithms. For ease of comparison, we call the naive scheme, which simply splits workloads according to the contactors' local computing speeds, *speed*, the allocation scheme based on each contactor's local 1-hop historic information is called *one-hop*, the algorithm that pre-calculates a participant's future potential computing ability is called *potential*, and the algorithm that adaptively switches different allocation methods is called *adaptive*. In order to compare these solutions with the optimal results, after the whole observation is completed (posterior knowledge), we find the optimal allocation and use *optimal* to represent it. Based on the posterior knowledge, one can compute the shortest path from source to participants, and the optimal allocation can be found by letting all participants complete their workload at the same time. Although this algorithm is useless in practice, we can use its result as a comparison criterion.

A. Evaluation Setup and Comparison Metric

The evaluations use both synthetic data and real data. In the synthetic simulation, we use a real social network data set called karate club [6]. However, since in real life the social-crowdsourcing system always contains too much diversity, such as users' computing features, in order to quickly and accurately find out the changing trends of our interested attributes, in each simulation, we only change one variable. The work source nodes are randomly selected, and each simulation is executed 10 times. There are 34 users, and average inter-contacting time between each pair of nodes is uniformly distributed from 40 units of time to 100 units of time. The computing speed of nodes also follows a uniform distribution from 0.0001 to 10 units of work per unit of time. The total size of a work is set as $200,000$ units. The accepting rates are uniformly distributed in the interval from 0.35 to 0.99. We take $3,000$ units of time as the length of an observation. In the real data-based experiments, we use Infocom06 trace. We eliminate the nodes with very low contacting frequencies, and we only use the data that was collected during daytime.

For efficient comparison, we adopt two metrics: percentage of completion and system (average) throughput rate. As we have mentioned in the introduction, the more nodes that join the system at an earlier phase, the sooner the work can be completed. The first metric can clearly show how well work segments are disseminated and processed within the systems at any time. By using this metric, one can observe the impacts of different methods on work segment diffusion. The second metric calculates the utilization of the system's available resources. The system throughput rate is computed as the ratio between the number of computing resources that have been used during an observation (the time interval before a majority of work segments has been completed) and the total

number of available resources that the system can provide. Due to the existence of work segments' propagation delays, system throughput rate can never reach 100%, even for the optimal result. The closer that a scheme comes to approaching the optimal result, the higher the system throughput rate is. The simulation basically consists of three parts. The first part focuses on the allocation of a single work on a regular social contacting graph, the second part shows the results regarding multiple works coexisting in our system, and the last part gives the results based on real trace.

B. Evaluation results

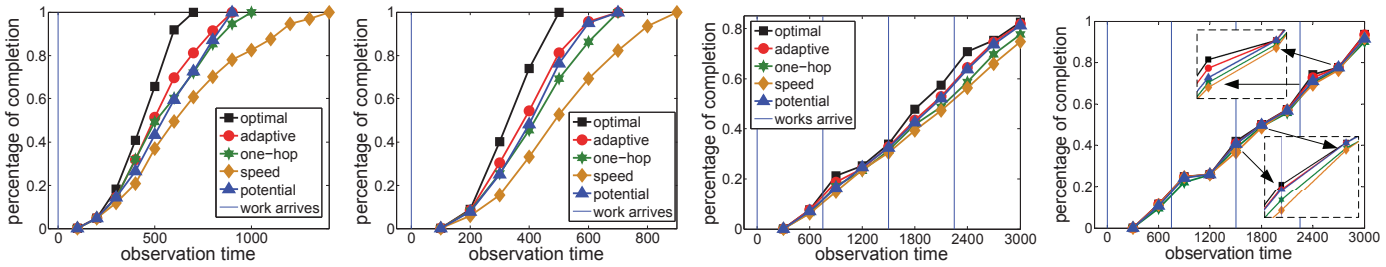
Fig. 22 shows the impacts of average accepting ratio on task's processing progress. During the simulation, we keep the average inter-contacting time of nodes uniformly distributed from 40 to 100 time units. The computing speeds are also randomly and uniformly generated with the interval $(0, 10]$ units of work per unit time. The minimal average acceptance rate is 35%, and the maximum is 99%. In Fig. 22, the x-axis represents the observation time, and the y-axis indicates the percentage of workloads that have been completed by the observed time. We can see that, with the growth of the accepting rate, the speed-based allocation scheme performs worse. One possible explanation is that when there are more participants in the system, the scheme cannot properly determine the workloads during each time of contacts. However, as the average accepting rate grows, it processes more quickly: the shape of its curve becomes closer to the upper left corner of the graph. For the other four methods, their work's processing progresses also become better at the high accepting rate scenario. Since the percentage of completion lines may become closer to each other, for the rest of evaluation, we focus on the average value of the system throughput rate.

Fig. 23 shows the impacts of average accepting ratio on the system throughput rate. With the increasing of the average accepting rate, all schemes' system throughput rates are growing upward. When the system has only a single task (Fig. 23 (a)), the growing speed of the optimal scheme is much faster than the other four solutions, and therefore, there is a gap between the optimal result and others. However, when there are multiple tasks (Figs. 23 (b) and (c)), the optimal scheme, the adaptive scheme, and the potential-ability-based scheme have similar throughput rates at the higher accepting rates. One possible explanation is that, when there is a single work within a social-crowdsourcing system, a certain amount of computing resources is wasted due to the inappropriate workload assignments; however, when the system contains multiple works from different sources, those unused resources are taken by other works, and therefore, the system throughput lines become closer to the optimal one in multiple works condition. Fig. 23 (d) gives the simulation results on real trace. Note that our approximation algorithms only consider the pairwise contacts, but in this real trace, there are situations where a group of users encounter each other at the same time. In these situations, we simply assign a sequential order to each pair of contacts, and then apply our algorithms. For Fig. 23 (d), we can see that the system throughput rate goes up with the increasing of accepting rate and becomes relatively stable at high accepting rate part.

In Fig. 24, we check the impacts of users' average computing speeds on the system throughput rate. We let the average speed gradually change from a relatively low speed to a high speed. During this process, we check the system throughput rate under three distinct conditions: a system with a single work, a system with multiple works (periodically arrived), and a system with multiple randomly arrived works. We also test our scheme on real trace with a single work source. All simulation results show that, with the average speed growing upward, the throughput of all workload allocation methods decrease. When nodes have higher computing speeds, obviously, the completion time of the works can be reduced. As a result, the effect of the amount of wasted resources during the work segments' propagation phase exacerbates the system's overall throughput. We further check the influence of the average computing speed on the changing pattern of completion time. For a given fixed-size work, with the growing speed, the completion time becomes smaller, but the reducing speed is not linear. One possible explanation is that, with the growth of the speed, a task can be finished before other workers participate in the task. One interesting phenomenon in Fig. 24 is that, comparing to the situation of single work source, the performance of one-hop scheme becomes worse when multiple works coexisting in the system. Since the contacts in our system are intermittent, we think this phenomenon is related with the timeliness of historic neighborhood workload records. One way to solve the problem is to assign a time window for each node's historic records. The length of the window must be related with the node's contacting rates with others. For example, we can first set up a threshold l about contact times, and then compute the expected time Δt for a node to have l times of contacts with others. The expected time Δt will be used as the window's size. Once a record about historic workload expires, it will be invalid.

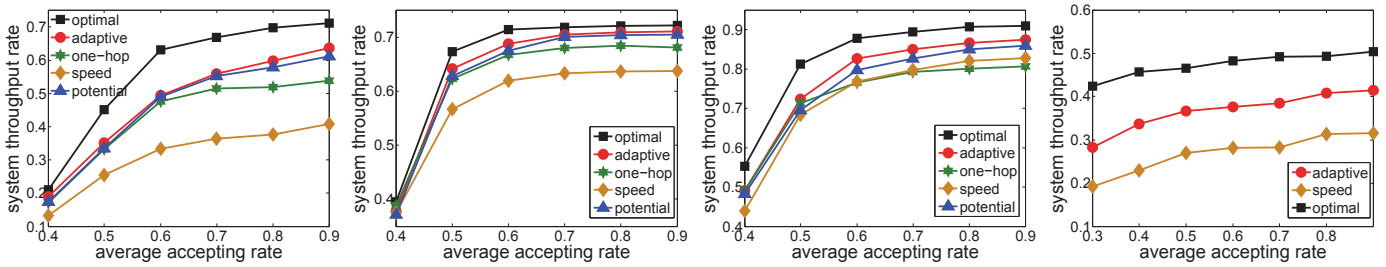
Next, we consider the influence of the size of works. In social-crowdsourcing, a system's computing resources are wasted during the segments' propagation phase. For a huge work, its work segments' propagation times are relatively small, compared to its completion time. Therefore, in general, the system throughput rate of a large work must be greater than that of a small-sized work. The simulation results are consistent with our qualitative analysis. In Fig. 25, the throughput rates increase along with growing work sizes. When there is only a single work in the system, there is a small gap on the throughput values of the optimal scheme and the others. However, when the system contains multiple works, such a gap disappears. The reason for this phenomenon is that, when nodes fully-occupy work segments (from plenty of tasks), the nodes always possess unfinished segments. Therefore, when propagating a task's segments, workers are using their resources on another task; no system resources are wasted.

Fig. 26 shows the impacts of users' average inter-contacting time. The average inter-contacting time directly affects the frequency of the segments' redistribution among nodes. The more frequently segments are exchanged, the more likely it is that the workloads of workers are balanced. As shown by Fig. 26, all schemes' system throughput rates are decreasing with the growth of average inter-contacting time. Moreover, all of them have a similar decreasing rate. It seems that the average inter-contacting time has a similar influence on all workload allocation approaches.



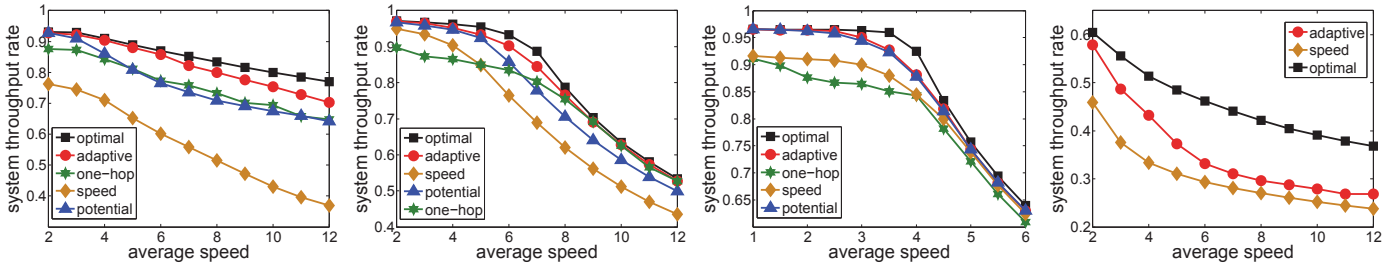
(a) single source, accepting rate: 50% (b) single source, accepting rate: 90% (c) multi-source, accepting rate: 50% (d) multi-source, accepting rate: 90%

Fig. 22. The impacts of average accept ratio on processing progress



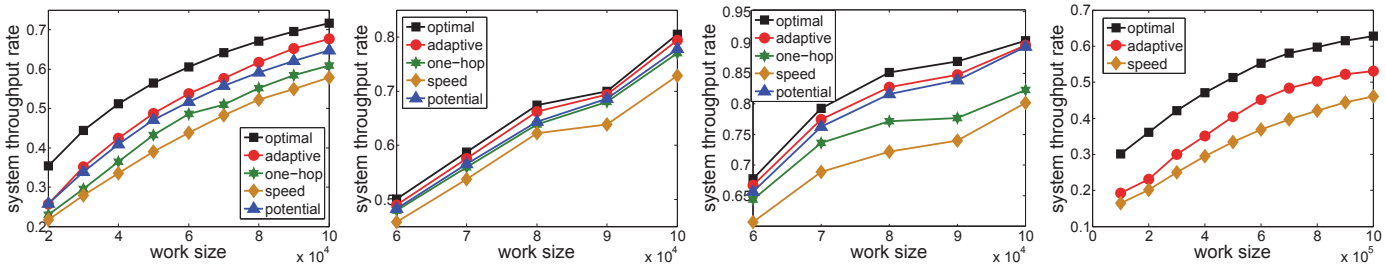
(a) single work (b) multi-works evenly arrive (c) multi-works randomly arrive (d) real trace

Fig. 23. Impacts of average accepting ratio on the system throughput rate



(a) single work (b) multi-works evenly arrive (c) multi-works randomly arrive (d) real trace

Fig. 24. Impacts of average speed on the system throughput rate

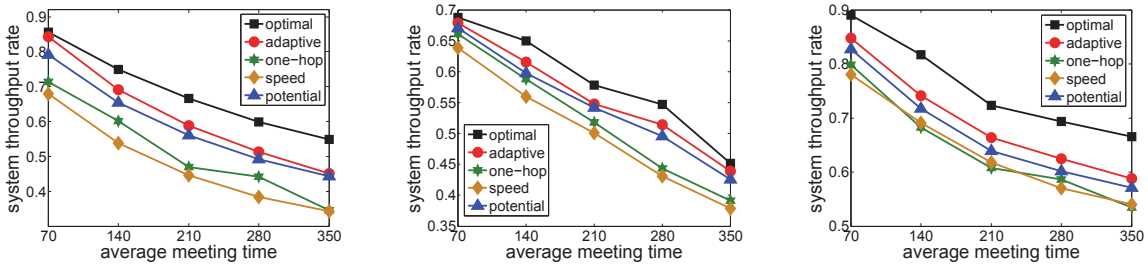


(a) single work (b) multi-works evenly arrive (c) multi-works randomly arrive (d) real trace

Fig. 25. Impacts of the size of works on the system throughput rate

Fig. 27 shows the impact of the number of work sources. In this simulation, we assume that there is a huge work, the segments of which are owned by several users. Unlike the condition of multiple works, here, the multiple source condition requires that all owners of a single work put the work into social-crowdsourcing at the same time. During the simulation, we gradually increase the number of the work segments' owners from 1 to 10. The total size of the whole work is a fixed value, and all work sources are randomly selected. From Fig. 27, we can see that our proposed adaptive scheme has the highest system throughput rate, compared to the other three approaches. All methods experience an increment with the growth of the number of sources. Since the total number of users is a fixed value, increasing the number of work owners can reduce its completion time, but the speed must converge to the condition that every node has already possessed his corresponding workload at the beginning.

When a social-crowdsourcing system contains multiple works, the arrival of a new work may disturb the current workload distributions. In this simulation, we want to check the mutual influence among works. Fig. 28 shows the impact of the number of coexisting works on our system, especially when the works randomly come at different times. If one work has not been finished while the other works appear, there must



(a) single work (b) multi-works evenly arrive (c) multi-works randomly arrive

Fig. 26. Impacts of the average length of inter-contacting time on the system throughput rate

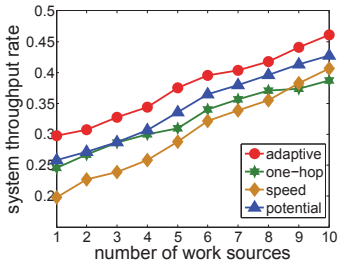


Fig. 27. Impact of the num. of sources

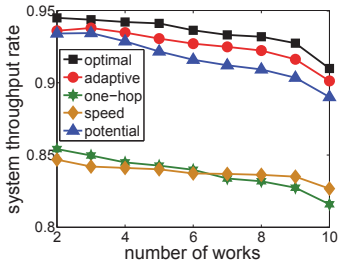


Fig. 28. Impact of the number of works

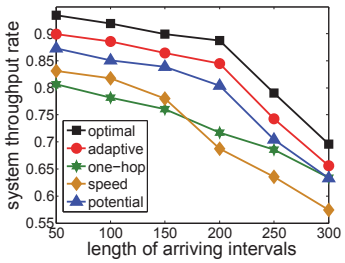


Fig. 29. Impact of arriving density

be certain interference between the old and the coming new works. In simulation, there are 2 to 10 works randomly coming during the observation time. The total length of the observation is set as $3,000$ units of time. The works' arrival times follow a uniform distribution. Also, the works' owners are randomly selected, and the total size of the works is a fixed value. We observe the system throughput rate decrease with the growth of the number of works. Moreover, all of them have a similar increasing pattern. Fig. 28 shows that the coexisting of multiple works do influence the workload assignments.

Since the arrival of a new work may interfere with the existing ones, in this part of simulation, we check the impacts of works' arriving densities. We let works be periodically generated at some random nodes, and we gradually increase the period from 50 to 300 units of time, which means the arriving densities change from high to low. Fig. 29 gives simulation results: with the growth of the length of periods, the system throughput rate decreases. Although our approximation algorithms try to assign certain workloads to nodes such that they can complete the task at the same time, the real finishing times are different; In fact, lots of resources are wasted during the a work's ending stage. When works arrive in a shorter period, the nodes, who just finished the previous work, are able to work on the new one.

VI. Related Work

In crowdsourcing [3], [13], [14], a tedious work is often subdivided into smaller pieces, which are then assigned to an undefined group of workers [15]. However, the current crowdsourcing systems are centralized [16]. Workers proactively join a crowdsourcing platform and seek tasks. When using a crowdsourcing platform [17], [18], inevitably, someone has to pay a fee for using the centralized server. For example, currently, Mturk collects a 10% commission on top of the amount that you paid for your work [19]. Moreover, these centralized platforms cannot assign works to the workers if there is not an internet connection. Paper [20] proposed an archetype for building a distributed crowdsourcing system in DTN. In this paper, we extend their model on mobile social networks, and further analyze the schemes for opportunistically allocating work segments among participants. When determining nodes' workloads, our schemes take contacting delay, computing speed, acceptance probability, the number of initial work sources, and the existence of resource competition works, into consideration.

Scheduling is a process of determining how to commit machines (executants) among a variety of works. Based on the number of executants, scheduling can be categorized into single-machine scheduling [21] or multiple-machine scheduling [22]. Generally speaking, the scheduling method used in this paper belongs to the multiple-machine scheduling. As the name suggests, multiple-machine scheduling tries to make up schedules for different executants. However, in our model, data segments are transferred based on nodes' opportunistic contacts. We cannot explicitly assign an exact work segment or workload, to a specific machine in advance, like the traditional multiple-machine scheduling did.

Scheduling in distributed systems involves load-balance and congestion control [23]. For the heterogeneous system, where nodes have different computing abilities, partitionable workloads are considered. Usually, a task is divided into different-sized segments, according to the capability of their executants [24], [25]. But in our problem, the size of an assigned workload is not only related with the local computing ability of a node, but also the abilities of the node's future contactors. Backpressure routing [10] is an algorithm for dynamically adjusting traffic over a multi-hop network. However, backpressure routing is hard to implement in a fully-distributed style, and it aims at enhancing the throughput of a network, while, our problem focuses on designing a local algorithm for reducing the completion time of works. Adopting backpressure routing may not be a good solution for our problem.

VII. Conclusion

In this paper, we proposed a new crowdsourcing system, called social-crowdsourcing. Unlike the traditional outsourcing platforms, our system is a distributed and self-organized system, which explores the social relations among users. Based on the proposed system, we consider the problem of workload partitioning and allocation among users. Unlike the traditional scheduling problems, the solution to our problem needs trade-offs between users' computing abilities, participance probabilities, and their social neighbors' abilities. By observing the work segments' diffusion pattern, we find that, at different processing stages, we should adopt different workload allocation schemes, from the progressive one to the conservative one. In this paper, we first propose a conservative approach and a progressive approach, and then, we design an adaptive mechanism to combine the two approaches. The proposed progressive scheme is based on the overall computing ability of each user's future potential contactors, and the conservative scheme takes the historical workload statuses of nodes' 1-hop neighbors into account. For automatically and gradually switching between different approaches, we consider the distribution of each node's expected completion time, based on collected historical information: a highly biased distribution indicates that the system needs a progressive way to transfer the surplus workloads to others, while an approximately uniform distribution simply requires local adjustment. Our solution is applicable, not only when a system has a single work, but also when the system contains multiple coexisting works. Extensive simulations prove the significant performance of our schemes.

References

[1] [Online]. Available: boinc.berkeley.edu/

[2] [Online]. Available: folding.stanford.edu/

[3] [Online]. Available: www.mturk.com/mturk/

[4] L. B. Chilton, G. Little, D. Edge, D. S. Weld, and J. A. Landay, "Cascade: Crowdsourcing taxonomy creation," in *ACM SIGCHI*.

[5] S. M. Ross, *Introduction to probability models*. Access Online via Elsevier, 2006.

[6] [Online]. Available: www-personal.umich.edu/∼mejn/netdata/

[7] J. M. Kleinberg, R. Kumar, P. Raghavan, S. Rajagopalan, and A. S. Tomkins, "The web as a graph: Measurements, models, and methods," in *Computing and combinatorics*. Springer, 1999.

[8] H. Deng, M. R. Lyu, and I. King, "A generalized Co-HITS algorithm and its application to bipartite graphs," in *ACM SIGKDD*, 2009.

[9] A. McCallum, D. Freitag, and F. C. Pereira, "Maximum Entropy Markov Models for Information Extraction and Segmentation," in *ICML*, 2000.

[10] L. Tassiulas, "Adaptive back-pressure congestion control based on local information," *IEEE Transactions on Automatic Control*, 1995.

[11] S. Moeller, A. Sridharan, B. Krishnamachari, and O. Gnawali, "Routing without routes: The backpressure collection protocol," in *ACM/IEEE IPSN*, 2010.

[12] M. J. Neely and R. Urgaonkar, "Optimal backpressure routing for wireless networks with multi-receiver diversity," *Ad Hoc Networks*, 2009.

[13] D. C. Brabham, *Crowdsourcing*. MIT Press, 2013.

[14] J. Howe, "The rise of crowdsourcing," *Wired magazine*, 2006.

[15] J. J. Horton and L. B. Chilton, "The labor economics of paid crowdsourcing," in *ACM EC*, 2010.

[16] D. C. Brabham, "Crowdsourcing as a model for problem solving an introduction and cases," *Convergence: the international journal of research into new media technologies*, 2008.

[17] A. Kittur, E. H. Chi, and B. Suh, "Crowdsourcing user studies with Mechanical Turk," in *ACM SIGCHI*, 2008.

[18] N. Eagle, "txteagle: Mobile crowdsourcing," in *Internationalization, Design and Global Development*. Springer, 2009.

[19] [Online]. Available: aws.amazon.com/pricing/mturk/

[20] S. Zhang, J. Wu, and S. Lu, "Minimum makespan workload dissemination in dtns: Making full utilization of computational surplus around," in *ACM MobiHoc*, 2013.

[21] C. Koulamas, S. Gupta, and G. J. Kyparisis, "A unified analysis for the single-machine scheduling problem with controllable and non-controllable variable job processing times," *European Journal of Operational Research*, 2010.

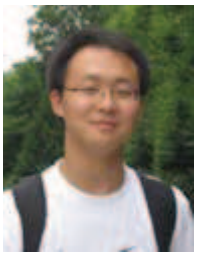
[22] S. A. Slotnick, "Order acceptance and scheduling: a taxonomy and review," *European Journal of Operational Research*, 2011.

[23] L. Wang, H. J. Siegel, V. P. Roychowdhury, and A. A. Maciejewski, "Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach," *Journal of Parallel and Distributed Computing*, 1997.

[24] M. Thiele, U. Fischer, and W. Lehner, "Partition-based workload scheduling in living data warehouse environments," *Information Systems*, 2009.

[25] A. L. Rosenberg, "Sharing partitionable workloads in heterogeneous NOWs: greedier is not better," in *IEEE Cluster*, 2001.

About Authors



Wei Chang received his B.Sc. in Computer Science in 2009 from Beijing University of Posts and Telecommunications, China. He is a Ph.D. student within the Department of Computer and Information Sciences, Temple University, USA. His research interests include security and resource allocation issues in social networks.

Jie Wu is the chair and a Laura H. Carnell Professor in the Department of Computer and Information Sciences at Temple University. Prior to joining Temple University, he was a program director at the National Science Foundation and Distinguished Professor at Florida Atlantic University. His current research interests include mobile computing and wireless networks, routing protocols, cloud and green computing, network trust and security, and social network applications. Dr. Wu regularly publishes in scholarly journals, conference proceedings, and books. He serves on several editorial boards, including IEEE Transactions on Computers, IEEE Transactions on Service Computing, and Journal of Parallel and Distributed Computing. Dr. Wu was general co-chair/chair for IEEE MASS 2006 and IEEE IPDPS 2008 and program co-chair for IEEE INFOCOM 2011. Currently, he is serving as general chair for IEEE ICDCS 2013 and ACM MobiHoc 2014, and as program chair for CCF CNCC 2013. He was an IEEE Computer Society Distinguished Visitor, ACM Distinguished Speaker, and chair for the IEEE Technical Committee on Distributed Processing (TCDP). Dr. Wu is a CCF Distinguished Speaker and a Fellow of the IEEE. He is the recipient of the 2011 China Computer Federation (CCF) Overseas Outstanding Achievement Award.