Effective Task Scheduling in Proximate Mobile Device based Communication Systems

Longfei Wu[†], Xiaojiang Du[†], Hongli Zhang[*], Wei Yu[‡], and Chonggang Wang[§]
[†]Dept. of Computer and Information Science, Temple University, Philadelphia, PA, USA, {longfei.wu, dux}@temple.edu
[*]School of Computer Science and Engineering, Harbin Institute of Technology, Harbin, China, zhanghongli@hit.edu.cn
[‡]Dept. of Computer and Information Science, Towson University, Towson, MD, USA, wyu@towson.edu
[§]InterDigital Communications, King of Prussia, PA, USA. chonggang.wang@interdigital.com

***Abstract*—Despite the increasing capabilities, mobile devices still cannot satisfy the computation requirement of many applications. Intuitively, this can be solved by outsourcing tasks to external resources such as a remote server, cloud, or closely deployed cloudlet. However, all of them require extra infrastructures. In this paper, we consider a proximate-mobile-device based communication system in which all tasks and resources are under the control of a central scheduler. We propose a friendship-based task scheduling algorithm to address the contentions when resources are not sufficient. We also present two attack models including the denial-of-service (DoS) attack and the collusion attack. We evaluate the performance of the proposed algorithm along with another contribution-based task scheduling algorithm through extensive experiments.**

***Keywords*—Resource sharing; mobile network; task scheduling**

I. Introduction

As today's mobile devices becoming increasingly powerful, mobile applications are more demanding on computation resources. A lot of software have published the mobile version, and mobile users expect their devices to run the applications as smooth as on PC. However, mobile devices are resource-constrained compared to PC, which results in poor user experience when executing complicated tasks. To deal with this issue, mobile devices may utilize external resources to run computational-intensive tasks, which is commonly known as Mobile Could Computing (MCC).

Mobile devices may be augmented through various cloud-based computing resources. In general, there are four types of cloud-based resources for mobile augmentation [1]: distant immobile clouds, proximate immobile computing entities, proximate mobile computing entities, and hybrid (a combination of the other three models). The first case, second case and hybrid of the two have been well studied in previous works [2][3][4][5][6]. For the proximate-mobile-device based MCC, previous works mainly focus on Ad Hoc mobile cloud framework [7][8].

Instead, our work studies the proximate-mobile-device based cloud system in which mobile devices are controlled in centralized manner. The following is a practical scenario: in a wireless local area network (WLAN), a cluster of mobile devices connected via Wi-Fi are devoted to share their computation resources with others running computation-intensive applications. A central scheduler built in the Wi-Fi Access Point (AP) could provide fast and effective scheduling for proximate mobile devices. Since a centralized scheduler has a full view of the outsourced tasks and available resources, the scheduling of tasks can easily achieve the maximum resource utilization and/or computational load balancing. However, the design of task scheduling algorithm should consider many factors including resource utilization, energy consumption, delay, and resource contention. During the peak time in which resources are not enough to support all of the outsourced tasks, some of them have to be postponed. Hence, a priority should be assigned to each outsourced task so that those with lower priorities get delayed when resource contention happens. The two schemes in our study determine the priority based on the contribution of mobile devices (users) in terms of helping others with their outsourced tasks, and the social relationship of the users, respectively. On the one hand, users that contribute more should get more rewards (e.g., higher priority). On the other hand, each user may have his/her own preference regarding whom to help with, and whom to get help from (e.g., close friends). The scheduling rule based on contribution and friendship can make a big difference on the system performance.

Security is another critical issue in the design of mobile cloud system. In our work, we discover two attacks that could degrade the performance of a MCC system. In the first threat, malicious participants could launch the denial-of-service (DoS) attack, in which they accept outsourced tasks but do not execute or return the results. This happens most likely among new/unfamiliar participants. The second attack is performed by collusions of two or more users, in which both the outsourced task owners and the resource providers are malicious users. They can fabricate a "successful" task outourcing while in fact no computation is done. We need a secure task scheduling algorithm that can mitigate these attacks.

In this paper, we propose a friendship-based task scheduling algorithm for a proximate-mobile-device based communication system. The friendship-based task scheduling algorithm utilizes the friendship among users to mitigate the DoS attack and the collusion attack. Specifically, the stable task-resource matching algorithm is used to achieve the optimization that tasks are allocated to the most reliable resource providers. We also implement a contribution-based task scheduling algorithm which ranks the priority of users based on their contributions to other users. We evaluate the two task scheduling algorithms with extensive simulations.

The rest of the paper is organized as follows. Section II discusses related works. Section III gives an overview of

978-1-4673-6432-4/15/$31.00 ©2015 IEEE

the background and the system model. Section IV presents the two task scheduling algorithms. Section V conducts the performance evaluation, and Section VI concludes the paper.

II. Related Work

Regarding proximate-mobile-device based augmentation of computation capacity, an ad hoc mobile cloud architecture has been proposed and implemented in [7] and [8]. The nearby mobile devices are pooled together, and share computing resources with others in need. Huerta-Canepa et al. [7] proposed a peer-based MapReduce framework formed by virtual mobile cloud computing providers in vicinity. Their evaluation results proved the feasibility of a virtual cloud computing platform using mobile phones. Marinelli [8] presented Hyrax, which employs a similar idea of utilizing Hadoop framework on Android smartphones to share data and computation resource. Hyrax allows client applications to execute computing tasks on both a network of smartphones and a heterogeneous network of phones and servers. They also showed that Hyrax allows applications to use distributed resources abstractly with no respect to the physical nature of cloud. Different from the above related work, in this paper we focus on the task scheduling issue for a mobile cloud system.

References [9] and [10] studied the computational offloading among a set of mobile devices. The Serendipity system is proposed and implemented in [9], which allows mobile tasks to use remote computational resources available from other mobile devices in its environment. The remote intermittently-connected mobile devices are regarded as resource providers, and the task allocation algorithm aims to improve computation speed as well as save energy for the initiating mobile device. [10] considers the same context in which mobile devices can offload computational tasks to other mobile devices in various connectivity conditions. The main goal of the task allocation is to maximize the lifetime of the whole system. However, the goal of our task scheduling algorithm is different: we mainly focuses on solving the resource contention issue when resources are insufficient and the security vulnerabilities. Besides, we consider and utilize two practical factors - contribution and friendship for task scheduling.

III. Background and System Model

We consider a proximate-mobile-device communication system in which computation tasks can be outsourced to available resources. Specifically, each mobile device in the pool may request for external computation resources when executing heavy tasks. Meanwhile, a device may provide its idle computation resources to others in need. The system consists of two components: a central scheduler and a number of mobile participants (e.g., smartphones and tablets). The responsibility of the scheduler is to collect task and resource information, then assign tasks to available resources. The sharing of computation resources in this mobile environment follows the basic rule that all participants offer their idle resources dedicatedly. The central scheduler also monitors the behavior of participants. If the amount of outsourced tasks exceeds the resources provided beyond a given threshold, that user will be prohibited from requesting external resources.

In our system model, we divide time into rounds (slots of equal length), and tasks are allocated and executed by round. In each round, users report their tasks and available resources to the central scheduler, and the scheduler allocates tasks to resources according to the task scheduling algorithm. To deal with the heterogeneity in both jobs and resources, we introduce in the concept of a basic unit. The length of one time slot T (one round) is defined as the time required by one unit of resource to execute one unit of task. The definition of unit resource and unit task are described below in Section III-A and III-B, respectively.

A. The Resource Model

The computation capacity of a mobile device depends on its CPU power, memory size and disk space. Mobile devices usually have different number of CPUs, amount of memory and disk size. Even with the same number of CPUs, the computation power may be different for different types of CPUs. We propose a measurement-based approach to benchmark the computation resource of a mobile device:

1) Build a benchmark task set with both computation-intensive and data-intensive tasks.
2) Pick some popular mobile device models denoted as $1, 2, ..., N$, and run the benchmark tasks on each of the devices while ensuring no other tasks are running. The execution time T_b measured for each device is inversely proportional to its computation capacity C_b, i.e., $C_b \propto \frac{1}{T_b}$. For simplicity, we set $C_b = 1/T_b$.
3) Calculate the basic unit of computation resource u_{res}, which is the greatest common divisor of the computation capacities $u_{res} = gcd(C_{b1}, C_{b2}, ..., C_{bN})$. The amount of resources of a device i can be obtained as $res_i = C_{bi}/u_{res}, (1 \leq i \leq N)$.

For each participant joining the system, we need to know its total amount of resources. If that device is among the tested samples, we can directly use the existing result; otherwise, we have to execute the benchmark tasks to quantify the number of unit resources it contains.

B. The Job Model

Similarly, we assume that a job can be divided into a number of task units u_{task}. For example, [7] gives a practical scenario where visitors in a museum want to understand an art description which is written in a foreign language. A feasible solution is to take a picture of the description, then multiple mobile devices can work collaboratively to extract and translate the foreign language into English. Specifically, a mobile device first uses an optical character recognition (OCR) software to split the text into characters (words), and distribute them to other devices nearby. Then the recognition and translation of each character (word) can be executed distributedly. Finally the results are collected and shared among all participants.

For a complex job that could not be splitted as independent tasks (e.g., certain tasks must be performed ahead of others), we use directed acyclic graph (DAG) to model the dependency. In a DAG $G = (V, E)$, each vertice $v_i \in V$ represents a task of the job while each edge $e_{ij} \in E$ represents the constraint that task v_j depends on task v_i. During each round, only tasks without predeccessors can be executed. In the rest of this paper, for simplicity, we use the term "task" and "resource" to represent "unit task" and "unit resource", respectively.

C. The Task Processing Model

In a given time slot, the total number of pending tasks include the newly generated tasks and previously unfinished tasks stored in the task queue. The task processing model consists of two modules: a local processing module and a central scheduling module. Tasks first go through the local processing module to be assigned to local resources. Then the unassigned tasks and unused resources are registered at the central scheduler. In the central scheduling module, the remaining tasks are assigned to unused (idle) resources of other devices based on certain rules/algorithms. When the amount of tasks exceeds the available resources, there will be resource contentions and some tasks will not be assigned. In such cases, the central scheduler should provide resources to tasks with higher priority. Unassigned tasks are put in the task queue and will be considered in the next round. By the end of each round, the results of outsourced tasks are collected by the scheduler and returned to task owners.

Note that since mobile devices could move out of the communication range, a task owner has the risk that it may never receive the result back. This issue could be addressed by the following approach: the scheduler can be equipped with the capability of tracking the signal strength of each registered device. If the signal strength is below a threshold, the registered device is considered unavailable and the central scheduler marks all outsourced tasks assigned to that device as *unfinished*, and the *unfinished* tasks are put back into the task queue. To deal with the case that a mobile device may run out of battery, we use the following approach: If the battery level of a mobile device is below a certain threshold, no more outsourced task will be allocated to that device.

IV. The Task Scheduling Algorithms

The details of friendship-based task scheduling (FTS) algorithm and contribution-based task scheduling (CTS) algorithm are given below.

A. The Friendship-based Task Scheduling Algorithm

The reliability and security issues arise for outsourced tasks that are executed by shared computation resources from other users. A single malicious participant could launch the denial-of-service (DoS) attack. Multiple malicious participants could conspire to fake task outsourcings. To mitigate these attacks, we propose a friendship-based task scheduling algorithm. The intuition is that the outsourced tasks are preferably executed by friends of the task owner, since friends tend to be more reliable and trustful. The friendship level could be evaluated based on familiarity (e.g., closer friends have higher levels). When joining the system, each new user will provide a list of friends in the order of friendship level to the central scheduler. For participants who are not a friend with that user, the scheduler put them at the end of the friend list in a random order such that the initial preference list includes all other participants in the pool. Then, outsourced tasks are able to apply for available resources in order according to the task owner's preference list. On the other side, resources are preferably offered to the provider's friends who have executed more tasks for the provider as a reward (*friendship reward rule*). When tasks from multiple users request for the same resource, the resource provider select the task to execute based on its friend list (preference list).

Algorithm 1 The FTS Algorithm in One Round

Input: Parameters N_t, N_r, t_{ij}, r_{ij};
Output: Parameters t_{ij}, r_{ij};

for each user i **do**
  Sort outsourced tasks of user i, t_{ij}, to obtain pl_i^{to};
  Sort provided resources of user i, r_{ij}, to obtain pl_i^{rp};
end for
while there is unassigned task that has not requested for every resource **do**
  Choose such a task t;
  Update resource preference list of t, rpl_t based on pl_i^{to};
  Pick the highest ranked resource r from rpl_t that t has not applied for;
  if r has not been allocated **then**
    Assign t to r;
  else
    Update task preference list of r, tpl_r based on pl_i^{rp};
    if r prefers current task to t (based on tpl_r) **then**
      t remains unassigned;
    else
      Reset r's current task to unassigned;
      Assign t to r;
    end if
  end if
end while
if $N_t > N_r$ **then**
  Put $N_t - N_r$ unassigned tasks into the task queue;
end if
Update t_{ij} and r_{ij} based on $\min(N_t, N_r)$ assigned tasks;

TABLE I. Parameters in the FTS Algorithms

Variable	Description
t_{ij}	Number of outsourced tasks of user i executed by user j.
r_{ij}	Number of available resources provided by user i to user j.
pl_i^{to}	Preference list of user i as task owner. (*offered help* list)
pl_i^{rp}	Preference list of user i as resource provider. (*received help* list)
tpl_r	Task preference list of resource r.
rpl_t	Resource preference list of task t.
N_t	The number of outsourced tasks in current round.
N_r	The number of available resources in current round.

As we can see, the friendship-based task scheduling issue is actually a two-way selection problem. Tasks and resources choose each other based on their own preference lists. We may solve this problem using the idea of stable matching (SM) [11]: tasks are regarded as men and resources are regarded as women. In SM problem, the result of men-propose stable matching is optimal for men while women are paired to the worst valid partner, or vice versa. However, in our scenario, only the task may have reliability concerns if it is assigned to an unknown resource provider; while for a resource, it does not matter which task to be coupled with from reliability perspective. Therefore, stable task-resource matching with task-propose can achieve an optimal solution such that tasks are assigned to the most reliable resource providers and providers are also satisfied with the assignment.

The relationship among participants are maintained and updated dynamically using a credit-based mechanism. This is necessary since the initial preference list could be outdated in circumstances like: a user find her friend does not regard her

as a friend any more (does not execute tasks for her), or users who are initially strangers help each other and become close friends. In the beginning, the preference list of a new user is the friend list provided by the user when joining the system. Later, preference lists are updated according to the number of tasks executed for other users (offered help), and the number of tasks outsourced to others (received help). Each task is counted as one credit. The scheduler maintains an ordered *offered help* list and an ordered *received help* list for each participant, which can reflect the up-to-date relationship. Tasks from a given user request resources from the top ranked user in the *offered help* list, since by the aforementioned *friendship reward rule*, tasks are most likely to be executed by those have received most help from the requestor. Meanwhile, resource providers also follow the *friendship reward rule*, and give resources to top ranked user in the *received help* list, namely those who have offered most help so far.

The task-resource matching issue is a variance of the stable matching problem which involves ties in preference list. Each user can be treated as a number of tasks or resources, and the corresponding preference list of the user (*offered help* list or *received help* list) is passed on to the associated tasks or resources. The tasks and resources of the same user tie in a list because they have the same preference ranking. For a stable marriage problem with ties (SMT), there are three forms of stability [12]. We adopt the weak stability for our problem. A weakly stable matching exists for every instance of SMT, it can be found in linear time by breaking all ties in an arbitrary way (i.e., by strictly ranking the members of each tie randomly) and then applying the Gale-Shapley algorithm [11].

The strategic manipulation of the preference list in the stable marriage are discussed in [13] and [14], by which men/women may find better mates. Teo et al. [13] studied the situation that there exists a specified woman who is the only deceitful agent. They proposed a polynomial time algorithm for constructing this woman's optimal cheating strategy. In terms of permuting men's preference lists to manipulate the outcome of stable matching, there is an example presented in [14]. However, the cheating schemes are based on the knowledge of all other users' preferences, which is not achievable in our system (only the central scheduler has the global knowledge).

Note that the amount of tasks and the amount of resources are not necessarily to be the same. It has been studied in [15] that for the instances of SM with unequal size of men and women sets, there always exists at least one stable matching in which each member of the smaller set is matched, and the larger set can be splitted into two subsets such that all members of one subset are matched while the members of the other subset are left single. Actually, the instance of SM with unequal size sets has exactly the same set of stable matchings as the instance with the unmatched members removed.

The FTS algorithm is described in Algorithm 1. All the parameters are listed in Table I.

B. The Contribution-based Task Scheduling Algorithm

To evaluate the performance of our FTS algorithm, we implement a contribution-based task scheduling (CTS) algorithm as comparison, which has the similar idea as the schemes used in [16][17][18][19]. In Disruption Tolerant Network (DTN), users can build up reputation credits by forwarding packets for others, and are rewarded with higher priority (receiving better forwarding service) when sending their own packets [16][17]. Similar mechanisms are also utilized in P2P systems, in which downloading services are provided depending on reputation credits accumulated during uploading [18][19].

We adopt the idea of reputation (contribution) in the CTS algorithm. Since all of the computation resources are provided by mobile devices, the performance of the mobile cloud system depends on the active involvements (i.e., contributions) of the registered devices. The contribution of a mobile device is not amount to the number of resources being used for the execution of outsourced tasks. Instead, the total idle resources provided (used or not) should be counted as contribution. If the amount of available resources is more than the number of tasks, tasks are randomly assigned to resources. Regarding participants who provide more resources may also consume more resources for their outsourced tasks, we use the concept of "absolute contribution" as the metric of priority (credit), which is defined as the balance of total resources provided and the total external resource usage. Each unit of resource and task is considered as one credit. If user A executes n tasks for user B, the credit of user A increases by n while user B's credit decreases by n.

The CTS algorithm is presented in Algorithm 2, and the parameters are listed in Table II.

Algorithm 2 The CTS Algorithm in One Round

Input: Parameters t_i, t_i^s, r_i, r_i^s;
Output: Parameters t_i^s, r_i^s;

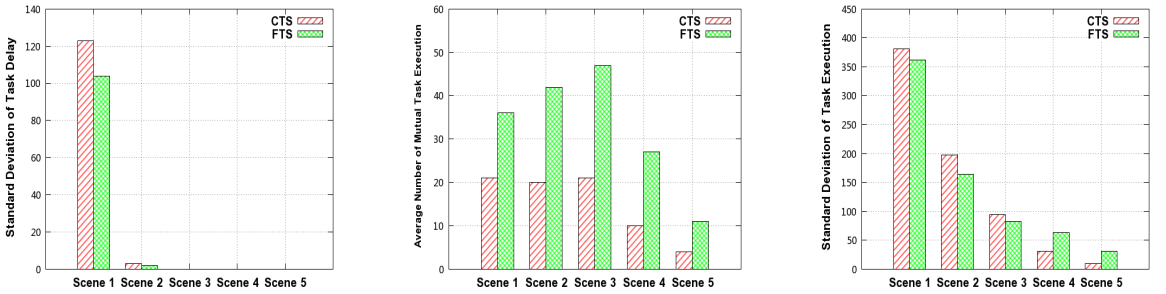
1: **for** each user i **do**
2: Update sum of provided resources, $r_i^s = r_i^s + r_i$;
3: Calculate (absolute) contribution credit, $c_i = r_i^s - t_i^s$;
4: **end for**
5: Sort contribution credits to obtain user contribution list l_c;
6: Generate task list l_t based on task owner's priority in l_c ($|l_t| = \sum_i t_i$);
7: Generate and randomize resource list l_r ($|l_r| = \sum_i r_i$);
8: **for** $k = 1$ *to* $\min(|l_t|, |l_r|)$ **do**
9: Assign kth task to kth resource;
10: **end for**
11: **if** $|l_t| > |l_r|$ **then**
12: Put $|l_t| - |l_r|$ unassigned tasks into the task queue;
13: **end if**
14: Update t_i^s according to $\min(|l_t|, |l_r|)$ assigned tasks;

TABLE II. PARAMETERS IN THE CTS ALGORITHMS

Variable	Description
t_i	Number of outsourced tasks of user i in current round.
t_i^s	Sum of outsourced tasks of user i.
r_i	Number of available resources provided by user i in current round.
r_i^s	Sum of available resources provided by user i.
c_i	The contribution credit of user i.
$l_c/l_t/l_r$	The user contribution list / task list / resource list.
$\|l_t\|/\|l_r\|$	The length of task list / resource list.

V. PERFORMANCE EVALUATION

In this section, we evaluate the performance of the friendship-based task scheduling (FTS) algorithm, which is compared with the contribution-based task scheduling (CTS) algorithm. Five metrics are used in the evaluation.



(a) Standard Deviation of Task Postpone (b) Average Mutual Task Execution (c) Standard Deviation of Task Finished

Fig. 1. Performance Comparison of CTS and FTS

• **Standard Deviation of Task Postpone.** Task postpone is caused by resource contentions during peak time. With the resource utilization maximized, the total amount of postponed tasks are the same whatever task scheduling algorithm is used. However, as occasional short postpone is tolerable for both application and user experience, the scheduling algorithm should prevent long postpone from happening to certain minority users like new participants. The standard deviation of postpone tasks shows the distribution of postpones among mobile users.

• **Average Number of Mutual Task Execution.** Friends are encouraged to help each other for outsourced tasks in FTS. The number of mutually executed tasks can indicate the amount of mutual help. If user A has executed m tasks for user B, and B has executed n tasks for A. The number of mutual task execution is $min\{m, n\}$.

• **Standard Deviation of Finished Tasks.** Battery is one of the severe constraints for mobile devices, and the execution of outsourced tasks would be extra burden for resource providers in terms of power consumption. The distribution of total finished tasks (own tasks and others' tasks) can reflect if some users suffer much greater power consumption than others.

• **Average Task Postpone of Existing/New Users.** The system should be scalable, and newly joined users should be treated fairly considering their outsourced tasks are not delayed much longer than existing users.

• **Average Number of Task Executors.** For a given user, the number of other participants who have executed outsourced tasks for that user can reflect the vulnerability of suffering the DoS attack. More executors means higher probability of encountering a DoS attack, given the same possibility for a participant to be malicious.

A. Simulation Setup

In the simulation, we build a mobile computation sharing environment with 30 users. We generate five scenarios (Scene 1 to 5) with different amount of resource supplies (listed in Table III). The number of tasks generated by a user in one time slot is uniformly distributed between 10 and 15. Each scenario is tested for 100 rounds with each round consisting of 500 time slots (TS). The average value of 100 rounds is calculated as the result.

B. Impact of Resource Supply

In this set of experiments, we analyze the effect of resource supply by comparing the performance in different scenarios.

TABLE III. RESOURCE SUPPLY IN DIFFERENT SCENARIOS

Scenarios	Resource Uniform Distribution Range
Scene 1	10 - 15
Scene 2	11 - 15
Scene 3	12 - 15
Scene 4	13 - 15
Scene 5	14 - 15

Figure 1(a) shows the standard deviation of task postpone decreases as the amount of resources increases. In Scene 3, the standard deviation of task delay drops to 0.066 and 0.053 for CTS and FTS, respectively. While for Scene 4 and 5, there is no delay at all. As we can see, the FTS has a lower postpone deviation than CTS when resource contentions occur.

Figure 1(b) shows that when there are resource contentions (Scene 1, 2 and 3), the average number of mutual task execution increases along with the total amount of resources (namely the total finished outsourced tasks) for FTS, while it's nearly unchanged for CTS. But when resources are sufficient and no resource contention happens (Scene 4 and 5), the number of mutually executed tasks drops in both FTS and CTS since less tasks need to be outsourced.

Figure 1(c) shows the standard deviation of finished tasks in CTS and FTS, which both decline as the amount of resources rises up. In Scene 1-3 when computation resources are limited, the distribution of outsourced task execution is more evenly in FTS. Starting from Scene 4 (plenty computation resources), the finished task deviation of CTS becomes lower than FTS. But this does not mean that FTS would deplete battery drastically, since the number of outsourced tasks also reduces as local resources become more sufficient.

C. Scalability and Fairness

The proposed system should be well scalable so that the resource sharing is fair to newly joined users. Initially, new users have no history (credit). This means if a new user is involved in a resource contention, it may not get resources for its outsourced tasks. Figure 2 presents the average task postpone in Scene 6, where 26 users participate from the beginning, and 4 others join at 1000 TS. The test starts recording at 1000 TS and last for 500 TS. The tasks and resources are generated with the same distribution as Scene 1 (limited resources). As we can see, the average task delay of new users in CTS is quadrupled to existing users; while in FTS, the task delay for the two groups of users are about the same. This is because new users with the lowest (no) contribution

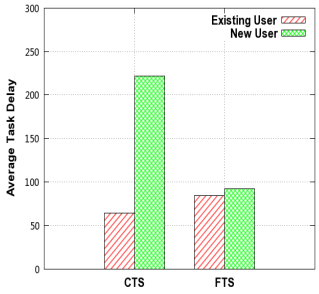


Fig. 2. Average Task Postpone of Existing/New Users in CTS and FTS

Fig. 3. Average Task Executor Number in CTS and FTS

may have friends among new users, or become friends with existing users for resources sharing.

D. Security

1) Mitigating the DoS Attack: A malicious user who claims to be a resource provider could launch the DoS attack, in which accepted outsourced tasks are not executed and fake answers are returned. When tasks are outsourced to an unfamiliar resource provider, they are in the risk of suffering the DoS attack that can cause extra postpone or more serious problems (e.g., the application may crash if fake results are used). Intuitively, the fewer unfamiliar task executors, the more secure the outsourced tasks are. Figure 3 shows the average number of task executors from Scene 1 to Scene 5. We can find that there are less task executors in FTS than in CTS, which means that FTS is more robust to the DoS attack than CTS.

2) Defending the Collusion Attack: If more than one malicious users collude during task outsourcing, the malicious users may deceive the credit accumulation mechanism. For example, malicious users can outsource "fake" tasks whose answers are already known to the accomplice, such tasks are not executed in fact. Since the returned answer is correct, even a central computation verifier cannot detect this kind of attack. Although the absolute contribution used in CTS can ensure that the total amount of contribution of malicious users remains the same, malicious users may exchange contribution credits and promote the priority of one particular attacker in a short period of time. This prioritized attacker has a better chance of getting resources than normal users during contention, and hence is capable of paralyzing the system by generating huge amount of outsourced tasks (normal users' outsourced tasks will be postponed). While in FTS, such collusion attack can only enhance the "friendship" among the attackers themselves, the task outsourcing of other legitimate users are not affected.

VI. Conclusion

In this paper, we consider the centralized task scheduling in a proximate-mobile-device based communication system. We propose a friendship-based task scheduling algorithm to solve the resource contention problem when resources are insufficient for outsourced tasks. The friendship-based task scheduling algorithm performs stable task-resource matching so that outsourced tasks can be assigned to the most reliable resource providers. Then we implement a contribution-based task scheduling algorithm which ranks the priority of users by absolute contribution. We evaluate and compare the two task scheduling algorithms with extensive simulations. We also present two types of attacks - the DoS attack and the collusion attack. Our analysis shows that the friendship-based task scheduling algorithm is more robust against these attacks.

Acknowledgement

This research was supported in part by the US National Science Foundation (NSF) under grant 1065444, and by the US Army Research Office under grant WF911NF-14-1-0518.

References

[1] S.Abolfazli, Z.Sanaei, E. Ahmed, A.Gani, and R.Buyya, "Cloud-based augmentation for mobile devices: Motivation, taxonomies, and open issues," *IEEE Communications Surveys and Tutorials*, June, 2013.

[2] X. Zhang, A. Kunjithapatham, S. Jeong, and S. Gibbs, "Towards an elastic application model for augmenting the computing capabilities of mobile devices with cloud computing," *Mob. Netw. Appl.*, vol. 16, no. 3, pp. 270–284, Jun. 2011.

[3] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: elastic execution between mobile device and cloud," in *Proceedings of the sixth conference on Computer systems (EuroSys)*, 2011.

[4] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *Pervasive Computing, IEEE*, vol. 8, no. 4, pp. 14–23, 2009.

[5] T. Verbelen, P. Simoens, F. De Turck, and B. Dhoedt, "Cloudlets: bringing the cloud to the mobile user," in *Proceedings of the third ACM workshop on Mobile cloud computing and services (MCS)*, 2012.

[6] T. Soyata, R. Muraleedharan, C. Funai, M. Kwon, and W. Heinzelman, "Cloud-vision: Real-time face recognition using a mobile-cloudlet-cloud acceleration architecture," in *IEEE Symposium on Computers and Communications (ISCC)*, 2012.

[7] G. Huerta-Canepa and D. Lee, "A virtual cloud computing provider for mobile devices," in *Proceedings of the 1st ACM Workshop on Mobile Cloud Computing (MCS)*, 2010.

[8] E. Marinelli, "Hyrax: Cloud computing on mobile devices using mapreduce," Master Thesis, Carnegie Mellon University, 2009.

[9] C. Shi, V. Lakafosis, M. H. Ammar, and E. W. Zegura, "Serendipity: enabling remote computing among intermittently connected mobile devices," in *Proceedings of the thirteenth ACM international symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, 2012.

[10] A. Mtibaa, A. Fahim, K. A. Harras, and M. H. Ammar, "Towards resource sharing in mobile device clouds: power balancing across mobile devices," in *Proceedings of the second ACM SIGCOMM workshop on Mobile cloud computing (MCC)*, 2013.

[11] D. Gale and L. Shapley, "College admissions and the stability of marriage," *American Mathematical Monthly*, vol. 69, no. 1, pp. 9–15, 1962.

[12] R. W. Irving, "Stable marriage and indifference," *Discrete Applied Mathematics*, vol. 48, no. 3, pp. 261 – 272, 1994.

[13] C.-P. Teo, J. Sethuraman, and W.-P. Tan, "Gale-shapley stable marriage problem revisited: Strategic issues and applications," *Management Science*, vol. 47, no. 9, pp. 1252–1267, Sep. 2001.

[14] D. Guseld and R. Irving, "The stable marriage problem," *Foundations of computing series MIT Press*, 1989.

[15] D. McVitie and L. Wilson, "Stable marriage assignment for unequal sets," *BIT Numerical Mathematics*, vol. 10, no. 3, pp. 295–309, 1970.

[16] Y. Zhu, B. Xu, X. Shi, and Y. Wang, "A survey of social-based routing in delay tolerant networks: Positive and negative social effects," *Communications Surveys Tutorials, IEEE*, First Quarter 2013.

[17] F. Xia, L. Liu, J. Li, J. Ma, and A. Vasilakos, "Socially aware networking: A survey," *Systems Journal, IEEE*, October 2013.

[18] P. Shi and K. Bhawalkar, "Megatorrent: An incentive-based solution to freeriding in p2p file-sharing networks," Operations Research Center, Massachusetts Institute of Technology, Tech. Rep., April 2008.

[19] M. R. Rahman, "A survey of incentive mechanisms in peer-to-peer systems," Cheriton School of Computer Science, University of Waterloo, Tech. Rep. CS-2009-22, June 2009.