Opportunistic Resource Sharing in Mobile Cloud Computing

SUMMARY The mobile cloud computing (MCC) paradigm is aimed at integrating mobile devices with cloud computing. In the client-server architecture of MCC, mobile devices offload tasks to the cloud to utilize the computation and storage resources of data centers. However, due to the rapid increase in the traffic demand and complexity of mobile applications, service providers have to continuously upgrade their infrastructures at great expense. At the same time, modern mobile devices have greater resources (communication, computation, and sensing), and these resources are not always fully utilized by device users. Therefore, mobile devices, from time to time, encounter other devices that could provide resources to them. Because the amount of such resources has increased with the number of mobile devices, researchers have begun to consider making use of these resources, located at the “edge” of mobile networks, to increase the scalability of future information networks. This has led to a cooperation-based architecture of MCC. This paper reports the concept and design of an resource sharing mechanism that utilize resources in mobile devices through opportunistic contacts between them. Theoretical models and formal definitions of problems are presented. The efficiency of the proposed mechanism is validated through formal proofs and extensive simulations.

key words: resource sharing, opportunistic networks, delay tolerant networks, mobile cloud computing

1. Introduction

Mobile cloud computing (MCC) is aimed at integrating mobile devices with cloud computing. MCC can be roughly divided into two different architectures, a client-server based one and a cooperation-based one [1]. In the client-server based architecture, mobile devices utilize resources in the cloud to enhance their functionality and improve their processing capabilities. However, the mobile data traffic is forecast to increase 13 times by 2017, with the volume climbing to 13.2 exabytes per month and the number of users approaching approximately 5.2 billion [2]. This increase in traffic demand is overloading cellular networks (especially in metro areas), forcing them to operate close to (and often beyond) their capacity, and causing a significant degradation in provided services. Considering the surge in machine-to-machine (M2M) and internet of things (IoT), the sheer number of mobile devices/applications will saturate resources in the cloud. Possible solutions to this problem include upgrading of cellular networks (LTE or LTE-advanced) and data centers. However, these solutions may not be cost-effective from the operator’s perspective due to the high cost (for power, location rents, deployment, and maintenance).

At the same time, along with the developments in hardware and software technologies, modern mobile devices (e.g., smart phones, wearable devices, and smart vehicles) have many more resources than previously, e.g., communication (3G/LTE, Bluetooth, and WLAN), computation (powerful CPUs), and sensing (location and thermo) resources. Mobile devices, from time to time, make contact with other devices that could provide resources to them. However, such contacts are intermittent and of indeterminate duration. Networks that utilize short-range communications and contact opportunities between mobile devices to transmit data are called “opportunistic networks” or “delay tolerant networks (DTNs)”. The focus has recently been on routing and content sharing in opportunistic networks. However, along with the increasing amounts of mobile resources, Conti et al. [3] estimated that exploiting contact opportunities only in smart phones of the world would give potential to perform one quadrillion processing tasks, and exchange one petabyte of data per second. As a result, researchers have begun considering resource sharing among mobile devices themselves to better utilize idle resources located at the “edge” of mobile networks and increase the scalability of future information networks. This strategy is called “opportunistic resource sharing” in cooperative MCC [4] or “opportunistic computing” [3].

A typical example application of opportunistic resource sharing could be the speech-to-text application as discussed in [5], [6]. Assume that a student named Alice wants to translate audio scripts into text for language study, while her smart phone lacks required resources (e.g., a speech-to-text application or powerful CPUs). Without opportunistic resource sharing, she has to rely on centralized solutions in which the remote data center translates the audio scripts for her, as shown in Fig. 1(a). Opportunistic
resource sharing solves this problem from a different approach, as shown in Fig. 1(b). If Alice knows that she will encounter her friends Bob and John in a short time (e.g., they have the same next class) and both of their devices own but seldom use the required resources, she could allocate the audio scripts to them to help her finish the translation task. Therefore, the overhead in terms of network traffic and resource consumption of the data center will be drastically reduced since only local communications (e.g., WLAN ad hoc, and Bluetooth) and local resources in end devices are utilized. This strategy also enables other interesting applications such as intelligent transportation systems, crisis management, and pervasive healthcare as described in [3].

However, the diverse capabilities and connectivities of mobile devices make the problem of opportunistic resource sharing very complicated [3]. Moreover, previous work in this area either analyzed simplified models that are not applicable to real situations or used heuristic algorithms without formal analysis. As a result, this paper proposes a novel mechanism to coordinate opportunistic resource sharing in cooperative MCC. The three main contributions of this paper are: (1) It presents the concept and design of an opportunistic resource sharing mechanism that utilizes resources in mobile devices through opportunistic contacts between them. (2) Theoretical models of opportunistic resource sharing that are based on multiplex parameters (inter-contact time, contact duration, bandwidth, and workload of resource providers) have been presented. (3) Algorithms are designed to implement the theoretical models. The efficiency of the proposed algorithms was validated through formal proofs and extensive simulation.

Following the literature review in Sect. 2, Sect. 3 explains the proposed system model of opportunistic resource sharing. Analyses of theoretical models and formal problem definitions are presented in Sect. 4. The proposed algorithms are described in Sect. 5. Section 6 presents the simulation results. Conclusions are drawn and future work is discussed in the last section.

2. Related Work

Much research has recently focused on resource sharing in opportunistic networks. Pasarella et al. [7] analyzed resource sharing by modeling opportunistic contacts between resource seekers and providers. Using their developed model, they investigated the optimal number of task replicas that should be offloaded onto encountered resource providers to minimize the task completion time. However, they considered an ideal homogeneous network, i.e., one in which every resource provider has the same amount of resources and the inter-contact times between nodes are identically distributed. Therefore, their model is not suitable for real situations in which the capabilities and movement patterns of mobile devices diversifies. Sadiq et al. [8] and Tamhane et al. [9] developed architectures of service composition in opportunistic networks. However, they only took into consideration the inter-contact time, thus neglecting other factors that greatly affect the quality of resource sharing like contact duration and contact bandwidth. Both work focused on selecting the best-qualified resource provider while neglecting the possibility of utilizing resources in multiple resource providers to improve the quality of resource sharing. Shi et al. [5], [6] described a task scheduling scheme called Serendipity that enables mobile resource seekers to offload tasks to other contacted mobile devices opportunistically. Different heuristic algorithms were used depending on whether a common control channel (CCC) was existed to forecast future contacts. Their preliminary results indicated the effectiveness of opportunistic resource sharing on two example applications (face recognition and speech-to-text). In contrast with the proposal in our paper, their scheme did not use historical information to predict future contact opportunities when an CCC was not available (was left for future work). Moreover, they did not present formal analysis of their heuristic algorithms. Conti et al. [3] described research background and motivation for studying in this area.

3. System Architecture

3.1 Opportunistic Contact Table (OCT)

In this paper, all mobile nodes are assumed to have some kind of short-range communication capability (e.g., WLAN ad hoc or Bluetooth) and emit a beacon signal periodically to advertise their presence. In practice, when another node senses the beacon, these two nodes establish a contact relationship by exchanging IDs and other relevant information. Bidirectional contacts are assumed in this paper, viz., if a contact exists between nodes \( N_d \) and \( N_b \), it also exists between \( N_b \) and \( N_d \). Also, each node is assumed to maintain a timer, which is used for recording contact information.

The concept of opportunistic contact table (OCT) is introduced in this section to capture characteristics of contact opportunities between different pairs of mobile nodes. Several terms are used to better clarify the study of OCT.

Inter-contact time (IT): the time interval between two contacts of the same pair of mobile nodes. The timer is used to record the time elapsed since the two nodes last encountered each other.

Contact duration (CD): the time during which a pair of mobile nodes are close enough to communicate with each other. The timer is used to record the time elapsed from the beginning to the end of their current contact.

Contact bandwidth (CB): the average bandwidth of short-range wireless communications between a pair of mobile nodes during their contact, e.g., the bandwidth with WLAN ad hoc differs from that with Bluetooth. In practice, any method like those proposed by Renesse et al. [10] and Sarr et al. [11] may be used to measure it.

Available resources (AR): the amount of idle resources in a mobile node that can be shared with other nodes, e.g., unused CPU cycles and cellular bandwidth. It depends on the node’s hardware/software configuration as
well as on its current workload. This information may be exchanged along with their IDs, when two nodes establish their contact relationship.

Every mobile node in this system model maintains an opportunistic contact table (OCT) for recording these four variables for other nodes that it has encountered. Consider a couple of mobile nodes $N_a$ and $N_b$. When $N_a$ and $N_b$ make contact, they exchange information of their available resources and probe their contact bandwidth. Both nodes also calculate inter-contact time and contact duration of the current contact by using their timers. They then update the related entries in their OCT using a moving window average strategy:

$$Q_{t+1} = \alpha \times Q_t + (1 - \alpha) \times Q_c,$$

where the variable $Q$ represents the four variables (CD, CB, IT, and AR) and $\alpha$ is a constant parameter between 0 and 1. $Q_t$ is the historical value in the OCT, and $Q_c$ is the value for the current contact. As a result, the updated value after this contact, $Q_{t+1}$, is a moving window average of the historical and current values. An example of OCT maintained by $N_a$ is shown in Table 1 in which it has had contact with nodes $N_b$, $N_c$, and $N_d$. Since $N_a$ can utilize its own resources without any communication cost, the values of its CB and CD are set to $+\infty$.

It can easily be proved that, as time passes, the moving window average strategy ensures that each entry in the OCT converges to the average value of the corresponding variables and this proof is independent of different $\alpha$ or initial values of the table entries [12]. It is actually a random process that fluctuates around the average values. There is a trade-off in the value of $\alpha$. A larger value of $\alpha$ reduces errors, but increase the time it take to reach the steady state and vice versa. In general, $\alpha$ should be set in accordance with the application requirements. Moreover, it may be difficult for a mobile node to manage all contacted nodes in its OCT because of their huge amount in reality. In that case, outdated entries may be removed to maintain the size of the OCT at a reasonable level, e.g., remove the least recently used (LRU) entry. Finally, since a mobile node makes contact with different groups of mobile nodes at different time and in different environments, different OCTs may be kept for different scenarios, e.g., an OCT for work time and an OCT for home time.

### 3.2 Overview of Opportunistic Resource Sharing

The mechanism of opportunistic resource sharing presented in this paper focuses on a resource seeker wishing to accelerate task completion by using available resources in other opportunistically contacted devices. The explanation of the mechanism uses four terms to improve clarity.

**Task (TS):** a job generated by a resource seeker that consumes resources to finish it. A task is described by $TS = (TS_i, TS_e, TS_o)$, where the size of the input parameters $TS_i$, the size of the execution body $TS_e$, and the size of the output results $TS_o$, are three key variables that determine the task completion time. Specifically, we define a task with a zero value for these three variables as a dummy task, $TS_{dummy}$.

**Resource seeker (RS):** a mobile node that seeks available resources in other nodes to accelerate the processing of its tasks.

**Resource provider (RP):** a mobile node that provides resources to an RS.

**Task latency (TL):** the length of time from when a task is generated to when it is finished. It represents the effectiveness of task processing from users’ perspective.

Different tasks consume different kinds of resources, e.g., image processing consumes CPU resources (FLOPS) and Internet accessing consumes cellular bandwidth resources (Mbps). Since the proposed mechanism is not constrained by specific types of tasks or resources, these details can be encapsulated by abstracting the size of tasks and units of resources in the following descriptions.

Task latency is comprised of three parts: uploading input parameters, processing task and downloading output results. When an RS generates a task, it first partitions the task into subtasks and uploads the parameters for the subtasks to RPs through opportunistic communications. Once an RP receives the required input parameters for its assigned subtask, it starts processing the subtask using its available resources. After the RP finishes its subtask, the RS downloads the results from it when they encounter again. Finally, the RS combines all the results received from the RPs to finish its task. Since both uploading the parameters and downloading the results may take multiple contacts to complete (depending on the CD and CB between the mobile nodes), an upload (or download) that was not completed is resumed from where it was stopped at the next contact. An example flow of opportunistic resource sharing is shown in Fig. 2.

### Table 1  Opportunistic contact table for node $N_a$.

<table>
<thead>
<tr>
<th>Node</th>
<th>IT</th>
<th>CD</th>
<th>CB</th>
<th>AR</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N_a$</td>
<td>0 s</td>
<td>$+\infty$</td>
<td>$+\infty$</td>
<td>500 units</td>
</tr>
<tr>
<td>$N_b$</td>
<td>1000 s</td>
<td>100 s</td>
<td>500 kbps</td>
<td>1000 units</td>
</tr>
<tr>
<td>$N_c$</td>
<td>5000 s</td>
<td>1000 s</td>
<td>11 Mbps</td>
<td>5000 units</td>
</tr>
<tr>
<td>$N_d$</td>
<td>300 s</td>
<td>100 s</td>
<td>6 Mbps</td>
<td>100 units</td>
</tr>
</tbody>
</table>

**Fig. 2** Example flow of opportunistic resource sharing.
4. Proposed Models for Opportunistic Resource Sharing

In this section, theoretical models of calculating task latency are first analyzed. The problem of minimizing task latency through opportunistic resource sharing is then formalized. Without loss of generality, a situation in which an RS \( N_i \) wants to offload a task \( ts \) to an RP \( N_j \) is assumed. Since there is no need to process a dummy task, the task latency of \( TS_{\text{dummy}} \) is always zero. Therefore, only non-dummy tasks are considered in this section.

4.1 Time Consumed by Uploading Input Parameters

As described above, due to the limited bandwidth and contact duration between mobile nodes in opportunistic networks, uploading input parameters may take one or more contacts to complete. Therefore, the time consumed by uploading input parameters is composed of two parts: (1) the time consumed by transmitting the input parameters, \( T_{\text{transmit}} \), and (2) the time consumed by waiting for contact opportunities, \( T_{\text{wait}} \).

\[
T_{\text{transmit}} = \frac{ts_i}{CB_{ji}},
\]

where \( ts_i \) is the size of the input parameters of task \( ts \) and \( CB_{ji} \) is the contact bandwidth between nodes \( N_i \) and \( N_j \), which is obtained from the OCT maintained in \( N_i \).

The estimation of \( T_{\text{wait}} \) is more complicated. First, the size of the input parameters that can be uploaded in one contact is given by

\[
T_{\text{up-one}} = CB_{ji} \times CD_{ji},
\]

where \( CD_{ji} \) is the contact duration between nodes \( N_i \) and \( N_j \). The value of \( T_{\text{wait}} \) depends on whether \( N_i \) is in contact with \( N_j \) when task \( ts \) is generated.

(1) \( N_i \) is not in contact with \( N_j \)

In this case, \( N_i \) has to wait for the next contact with \( N_j \) to begin the uploading process. Under the assumption that a task is generated uniformly in time, the expected length of the waiting period is one half of their inter-contact time.

\[
T_{\text{first-wait}} = \frac{1}{2} \times IT_j,
\]

where \( IT_j \) is the inter-contact time between nodes \( N_i \) and \( N_j \). After the first contact, if the process of uploading is not finished, the expected size of the remaining input parameters is given by

\[
IS_{i-\text{remaining}} = IS_i - T_{\text{up-one}}.
\]

(2) \( N_i \) is in contact with \( N_j \)

In this case, \( N_i \) is able to upload the input parameters to \( N_j \) as soon as the task is generated. However, under the assumption of uniform distribution of task generation, the expected remaining contact time after the generation of tasks is one-half of the expected contact duration between nodes \( N_i \) and \( N_j \). Consequently, the size of the input parameters that can be uploaded during the first contact is given by

\[
T_{\text{up-first}} = CB_{ji} \times 1 \times CD_{ji} = \frac{1}{2} \times T_{\text{up-one}}.
\]

After the current contact, if the process of uploading has not been finished, the expected size of remaining input parameters is given by

\[
IS_{i-\text{remaining}} = IS_i - T_{\text{up-first}}.
\]

Since \( N_i \) is expected to be contacting with \( N_j \) with an interval of \( IT_j \), and \( T_{\text{up-one}} \) input parameters will be uploaded during each contact, the remaining waiting time for uploading the input parameters is given by

\[
T_{\text{remaining}} = \left[ \frac{IS_{i-\text{remaining}}}{T_{\text{up-one}}} \right] \times IT_j, \tag{6}
\]

where \( \lceil \cdot \rceil \) is a ceiling function. Therefore, the total expected waiting time is given by

\[
T_{\text{wait}} = T_{\text{first-wait}} + T_{\text{remaining}} = \frac{1}{2} \times IT_j + \left[ \frac{IS_i}{T_{\text{up-one}}} - 1 \right] \times IT_j. \tag{7}
\]

Finally, the total time consumed by uploading the input parameters in this case is given by

\[
T_{\text{upload}} = T_{\text{transmit}} + T_{\text{wait}}. \tag{11}
\]

4.2 Time Consumed by Processing Tasks

Intuitively, the time consumed by processing a task decreases if the utilized resources increase. The model proposed by Nishio et al. [4] is used to calculate it here

\[
T_{\text{proc}} = \frac{ts_e}{AR_j}, \tag{12}
\]

where \( ts_e \) is the execution body size of \( ts \) and \( AR_j \) is the estimated available resources in node \( N_j \) according to the OCT in \( N_i \), e.g., if using a CPU that perform \( AR_j \) operations per second to process a task with \( ts_e \) operations, the processing time can be calculated using Eq. (12).
4.3 Time Consumed by Downloading Output Results

Since the process of downloading the output results is the reverse of uploading input parameters analyzed in Sect. 4.1, we can substitute \( t_s \) in Eqs. (2), (7), and (10) with the size of the output results \( t_{so} \) to get the time consumed by downloading results, \( T_{download} \) (Eq. (7) or (10) is used depending on whether \( N_i \) is in contact with \( N_j \) when the task \( ts \) is finished). As a result, the total task latency of \( ts \) is

\[
TL (ts) = T_{upload} (ts) + T_{proc} (ts) + T_{download} (ts) \quad (13)
\]

4.4 Problem Definitions

An RS generally wants to utilize resources in RPs to reduce the latencies of its tasks. Therefore, it has to partition its task into subtasks and assign these subtasks to appropriate RPs. Here, an opportunistic network in which an RS may make contact with \( N \) RPs (including the RS itself) is assumed, and \( x_k \) is defined as the percentage of the original task assigned to the \( k \)-th RP (i.e., the subtask). The subtask is dummy if \( x_k \) is zero. As a result, the problem of minimizing task latency for a generated task \( ts \) is formalized as

**P1:**

\[
\text{Min. } \max_{k=1}^{N} TL_k (x_k \times ts) ,
\]

\[
\text{S.t. } \sum_{k=1}^{N} x_k = 1 ; \quad x_k \geq 0 \quad (k = 1, \ldots, N) ,
\]

where \( TL_k (x_k \times ts) \) is the latency of the subtask assigned to the \( k \)-th RP. The objective function minimizes the maximum task latency for \( N \) RPs since the task is finished only when the RS has received the results from all RPs. The constraint conditions ensure a complete partition of tasks.

Solution vector \( X \) is defined as an \( N \) dimensional vector \( (x_1, x_2, x_3, \ldots, x_N) \) that satisfies the constraint conditions of problem **P1**. The subset of coordinates in \( X \) that are positive \( (x_k > 0) \) is called the support of \( X \). A solution vector is full support if all its coordinates are positive. A particular reordering of \( X \), \( (x_{m1}, x_{m2}, x_{m3}, \ldots, x_{mN}) \), such that the inter-contact time of \( x_{mk} \) is in a nondecreasing order is an order vector of \( X \) and is represented by \( X_{order} \). The objective value of problem **P1** represents the expected latency of \( ts \) when it is partitioned and allocated according to \( X \). Without loss of generality, \( t(X) \) is used to represent it.

5. Proposed Algorithms for Opportunistic Resource Sharing

5.1 LP-Based Approximation Algorithm

It is difficult to solve optimization problem **P1** directly due to the discontinuities of the ceiling functions in Eqs. (6), (7), and (10). Therefore, an approximation algorithm based on linear programming (LP) is proposed to solve it and is referred to as “ALP”.

First, an auxiliary function called \( TL' \) is defined to approximate \( TL \) defined in Eq. (13):

\[
TL' (ts) = \frac{ts_1 + ts_2}{CB_j} + \frac{ts_v}{AR_j} + \frac{ts_l + ts_o}{T_{up-one} \times IT_j} , \quad (14)
\]

where \( T_{up-one} \) is defined in Eq. (3). It is easy to verify that \( TL'(TS_{dummy}) \) is zero. As a result, optimization problem **P1** is transformed into:

**P2:**

\[
\text{Min. } \max_{k=1}^{N} TL'_k (x_k \times ts) ,
\]

\[
\text{S.t. } \sum_{k=1}^{N} x_k = 1 ; \quad x_k \geq 0 \quad (k = 1, \ldots, N) ,
\]

The definitions of solution vectors, supports of solution vectors and order vectors are the same as those for the problem **P1**. The objective value of problem **P2** is represented by \( t'(X) \). It is easy to see that both the definitions of \( TL' \) and the constraint conditions in **P2** are convex. As a result, problem **P2** is a piecewise-linear minimization problem and is equivalent to the LP problem

**P3:**

\[
\text{Min. } v ,
\]

\[
\text{S.t. } TL'_k (x_k \times ts) \leq v \quad (k = 1, \ldots, N) ;
\]

\[
\sum_{k=1}^{N} x_k = 1 ; \quad x_k \geq 0 \quad (k = 1, \ldots, N) ,
\]

where \( v \) is an auxiliary variable.

**Theorem 1.** Application of the same solution vector \( X \) to problems **P1** and **P2** limits the error between the objective values to within the maximum inter-contact time of the support of \( X \), \( IT_{max-support}(X) \), i.e.,

\[
|t'(X) - t(X)| \leq IT_{max-support}(X) .
\]

\[\square\]

From Theorem 1, it is easy to see that \( t'(X) + IT_{max-support}(X) \) is the worst-case bound of \( t(X) \), i.e., \( t_{worst-bound} \). However, we can only get a solution vector \( X \) with full support by solving **P3** directly due to the limitation of LP approximation. This defect may cause a large \( t_{worst-bound} \) in some cases, e.g., a small subtask is assigned to an RP that is rarely encountered (with large inter-contact time). As a result, if the support of the optimal solution vector \( X_{optimal} \) for problem **P1** can be found and **P3** can be solved on the support of \( X_{optimal} \) the error is further decreased.\(^\dagger\)

\[\dagger\]

**Theorem 2.** Suppose that \( (x_{m1}, x_{m2}, x_{m3}, \ldots, x_{mN}) \) is the optimal solution vector \( X_{optimal} \) to the problem **P1** and that \( (x_{m1}, \ldots, x_{mN}) \).

\[\dagger\] Here, solving **P3** on the support of a solution vector \( X \) means set \( x_0 \) to be zero in the constraint conditions of **P3** if it is not in the support of \( X \).
ALP algorithm

\[ X_r = \text{Solve } P.3 \text{ on the support } (x_{m_1}, x_{m_2}, x_{m_3}, \ldots, x_{m_k}) \, ; \]
\[ X_{\text{temp}} = 0 \, ; \quad i = 1 \, ; \]
for \((i = 1 ; i < N ; +i)\)
\[ X_{\text{temp}} = \text{Solve } P.3 \text{ on the support } (x_{m_1}, x_{m_2}, x_{m_3}, \ldots, x_{m_i}) \, . \]
if \(t_{\text{worst-bound}}(X_{\text{temp}}) < t_{\text{worst-bound}}(X_r)\)
\[ X_r = X_{\text{temp}} \, ; \]
return \(X_r\).

AAC algorithm

\[ CD_{\text{current}} = \text{calculate length of current contact} ; \]
\[ ts_{\text{pri}} = \text{select a remaining subtask based on priority rules} ; \]
// if \(y_k \times ts_{\text{pri}}\) can be finished within \(CD_{\text{current}}\),
assign a fraction of \(y_k \times ts_{\text{pri}}\) to the current encountered RP.
remaining subtask is \((1 - y_k) \times ts_{\text{pri}}\).

(1) The subtask whose input parameters have not been completely uploaded to the RP assigned by the ALP algorithm (type one);
(2) The subtask whose input parameters have already been uploaded to the RP assigned by the ALP algorithm. However, output results of the subtask have not been received yet (type two).

The following two rules are used to determine the priority of subtasks allocated to the current encountered RP:
(1) Subtasks of type one have a higher priority than those of type two;
(2) In the same type of remaining subtasks, the subtask previously assigned to the RP with largest inter-contact time has the highest priority, since an RS meets RPs with larger inter-contact time less frequently.

It is clear that subtasks of type two may be parallel processed by both the current encountered RP and the RP assigned by the ALP algorithm (two copies of each subtask are parallel processed in the worst case). Therefore, using of the AAC algorithm can result in duplicate task processing.

Remark: When two mobile nodes make contact first time, entries for each other are created in their OCTs with a large initial value of IT and small initial values of CD, CB, and AR. Since this means large expected task latencies, at the beginning, these two nodes seldom schedule their tasks to each other based on the ALP algorithm. They mainly share resources through the AAC algorithm at this stage. As time passes, variables in the OCTs converge to their averages which promotes deep-level resource sharing between them step by step. A timer can be set to delete outdated entries to prevent failure of nodes. In that case, two nodes treat each other as a new node when they encounter again.

5.2 Algorithm for Accidental Contacts

An RS uses the ALP algorithm to partition the original task into subtasks and assigns them to a group of appropriate RPs. However, an RS may encounter an RP that has idle resources but was not scheduled any subtask by the ALP algorithm (due to the randomness of contact opportunities). The RS should still be able to utilize the idle resources in that RP to accelerate its task completion. An algorithm is introduced here to deal with this kind of accidental contacts. It supplements the ALP algorithm and is called “AAC”.

The basic idea of AAC is simple: the RS estimates the duration of the current contact and assigns the encountered RP a subtask that it can finish within that time (since the result of ALP indicates that the current encountered RP is a good candidate, it may take a long time to make contact with it again.). Since estimating contact duration is not the focus of this paper, a simple model in which two nodes move from their current locations to their destinations in line is used. As a result, the duration of their contact is calculated in accordance with their traces, i.e., the period during which their distance apart is within the wireless transmission range of each other (location information can be obtained through GPS or peer-based techniques [13]). It should be noted that other improved methods of estimating contact duration can be smoothly integrated into AAC.

The last problem is which part of the remaining (unfinished) subtasks should be allocated to the current encountered RP. There are two types of remaining subtasks:
according to their amounts of resources. One unit of resources is assumed to process one unit of task execution body in one second.

(1) Super node (SN): node with 100 units of resources.
(2) Common node (CN): node with 20 units of resources.
(3) Poor node (PN): node with no resources.

Without loss of generality, there were one SN, two CNs, and two PNs associated with each landmark. Every node generated a task with an equal probability. Each (sub)task was served according to a FIFO policy by RPs. Five seconds were preserved for mobile nodes to detect each other at the beginning of each contact. The value of $\alpha$ for generating OCTs in Eq. (1) was set to 0.9. The simulation first run without generating any task for 20,000 seconds so that the nodes could establish their OCTs. This is called the training period of OCTs.

Four different methods were compared. The first two were proposed in Sect. 5. Since the AAC algorithm relies on the ALP algorithm, it was not evaluated alone.

(1) ALP: The RS partitions the original task into subtasks and assigns them to a group of RPs in accordance with the proposed ALP algorithm.
(2) ALP-AAC: Apart from the ALP algorithm, the RS uses the AAC algorithm to make use of accidental contact opportunities.
(3) Epidemic: The RS replicates the original task to every encountered RP until it receives the result from one. To reduce resource consumption, an RS cancels duplicated tasks in encountered RPs if it has already received the results of this task.
(4) Best-qualified RP (BQR): The RS assigns the original task to the best-qualified RP, i.e., the one expected to minimize task latency. This is an improved version of the method proposed by Shi et al. [5], [6].

Four methods were compared on the basis of two criteria.

**Task latency (TL):** described in Sect. 3.2.

**Duplication rate (DR):** since both ALP-AAC and Epidemic methods generate duplicate (sub)tasks, this metric measures the ratio between the size of the task being processed and the size of the original task. Small DR is preferred to prevent saturating RPs’ resources. Since ALP and BQR methods do not generate any duplicate task, their values of DR are always one. As described in Sect. 5.2, the ALP-AAC method generates at most two copies of each subtask. As a result, its value of DR is no larger than two in any case.

### 6.1 Parameters of the Mobility Model

The proposed algorithms were first evaluated for different parameters of the mobility model. The average latencies for one task for different node speeds are shown in Fig. 3(a).

<table>
<thead>
<tr>
<th>Simulation area</th>
<th>3000 × 3000 m</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size of landmarks</td>
<td>1000 × 1000 m</td>
</tr>
<tr>
<td>Number of landmarks</td>
<td>4</td>
</tr>
<tr>
<td>Upper left corner of landmarks</td>
<td>(1000, 1000), (1000, 3000), (2000, 2000), (0, 2000),</td>
</tr>
<tr>
<td>Number of SNs</td>
<td>4</td>
</tr>
<tr>
<td>Number of CNs</td>
<td>8</td>
</tr>
<tr>
<td>Number of PNs</td>
<td>8</td>
</tr>
<tr>
<td>Size of execution body, $t_{se}$</td>
<td>10,000 units</td>
</tr>
<tr>
<td>Size of input parameters, $t_{si}$</td>
<td>100 Mbits</td>
</tr>
<tr>
<td>Size of output results, $t_{so}$</td>
<td>100 Mbits</td>
</tr>
<tr>
<td>Link bandwidth</td>
<td>6 Mbps</td>
</tr>
<tr>
<td>Transmission range of nodes</td>
<td>150 m</td>
</tr>
<tr>
<td>Mobility model</td>
<td>Random way point</td>
</tr>
<tr>
<td>Mobility speed</td>
<td>10 (m/s)</td>
</tr>
<tr>
<td>Pause time</td>
<td>0 s</td>
</tr>
<tr>
<td>Deviation probability</td>
<td>0.1</td>
</tr>
<tr>
<td>$\alpha$ for generating OCTs</td>
<td>0.9</td>
</tr>
<tr>
<td>Training period of OCTs</td>
<td>20,000 s</td>
</tr>
</tbody>
</table>

††With the default value of $\alpha$, 20,000 seconds is enough for OCTs to reach the steady state in following simulation. The impacts of different values of $\alpha$ and training period are further discussed in Sect. 6.3.
When the speed was slow, the values of inter-contact time and contact duration were large. That made it harder for an RS to encounter a specified RP. However, it also indicates that if an RS encountered an RP, the large contact duration ensured a high probability of finishing the task within this contact. As a result, both ALP and BQR performed worse because they only assign (sub)tasks to a group of specified RPs. Both ALP-AAC and Epidemic performed better since they enabled an RS to utilize resources in any RP that is encountered first. The average latencies for one task with different deviation probabilities are shown in Fig. 3(b). The average latency increased with the deviation probability. This was because a larger deviation probability results in the mobile nodes moving in a wider area rather than spending most of their time in the landmark area with which they were associated. Therefore, it was more difficult for mobile nodes to make contact with each other.

Figures 4(a) and 4(b) show the duplication rates for one task for different node speeds and different node deviation probabilities. ALP and BQR methods did not generate any duplicate task. Therefore, a single line of value one is used to represent their values in the figures. Since Epidemic blindly replicated the original task to every encountered RP, it had a much larger DR than the other three methods. ALP-AAC only utilized idle resources in a contacted RP. As a result, its DR values were close to one in both figures.

It is obvious from these results that the proposed ALP-AAC method had the smallest task latencies of the four methods for the different parameters of the mobility model. Compared with the Epidemic method, it had dramatically lower DR values. Moreover, the proposed ALP method performed better than the BQR method in most cases even for a constant DR value (one).

6.2 Parameters of Tasks

The proposed algorithms for different parameters of tasks were evaluated. The average latencies for one task for different execution body sizes ($t_{se}$) are shown in Fig. 5(a). The average latency increased with $t_{se}$. This was because mobile nodes need more time to process a larger task, as shown by Eq. (12). Along with the increasing size of $t_{se}$, the workload of the RPs increased. As a result, there was less chance of encountering an RP with idle resources and thus to utilize the AAC algorithm. This is why the performance of the ALP algorithm converged to that of the ALP-AAC algorithm as $t_{se}$ increased. The average latencies for one task for different input parameter sizes ($t_{si}$) and output result sizes ($t_{so}$) are shown in Fig. 5(b). The sizes of $t_{si}$ and $t_{so}$ were assumed to be the same in the simulation. The average task latency increased with $t_{si}$ and $t_{so}$. This was because more time was consumed by uploading the input parameters (downloading the output results).

Figure 6(a) and Fig. 6(b) show duplication rates for different parameters of tasks. It seems counterintuitive that the DR for Epidemic decreased with $t_{si}$ and $t_{so}$, as shown in Fig. 6(b). This was because a larger $t_{si}$ meant a longer time to upload the parameters (taking one or more contacts to
Fig. 6 Duplication rates for different parameters of tasks.

Fig. 7 Task latencies for different value of $\alpha$.

Fig. 8 Duplication rates for different value of $\alpha$.

6.3 Parameters of Generating OCTs

Finally, to determine the impact of accuracy of OCTs on the proposed methods, different parameter settings for generating OCTs are discussed. Figure 7 shows average task latencies for one task under different values of $\alpha$. Since the historical part usually has a higher weight in the moving window average strategy, only values larger than 0.5 were considered. Task latencies for the Epidemic method are presented for the sake of convenience, although they were not influenced by statistics in the OCTs. As discussed in Sect. 3.1, the accuracy of OCTs increases with the value of $\alpha$. Therefore, BQR, ALP, and ALP-AAC performed worse when the value of $\alpha$ is small, since they allocated tasks according to inaccurate statistics in OCTs. The proposed ALP method performed better than the BQR method, even if both of them suffered from inaccurate statistics and did not generate any duplicate task. The proposed ALP-AAC method was more tolerant to inaccurate OCTs, since it used duplication strategies to compensate for inaccurate task allocations. This was proved by Fig. 8 in which the duplication rate of ALP-AAC became larger when the value of $\alpha$ decreased.

The accuracy of OCTs also depends on its training period. As discussed in Sect. 3.1, a larger value of $\alpha$ requires a longer training period to generate accurate OCTs. Simulation results under different training periods were similar to those under different values of $\alpha$ shown in Figs. 7 and 8, since both of them resulted in inaccurate OCTs when their values were small. Therefore, they are not shown in this paper.

It is obvious from these results that accurate user statistics are important to the concept of opportunistic resource sharing. A simple moving window average strategy is used in this paper to illustrate the superiority of our proposal. However, how to extract knowledge from existing data (e.g., life-logs of mobile users) to generate accurate OCTs is still an important open issue that needs further investigation.
7. Conclusion

This paper presented the concept and design of an opportunistic resource sharing mechanism. Mobile nodes make use of contact opportunities between themselves to share resources and accelerate their task completion. Simulation results indicated that the proposed mechanism was efficient in terms of both time and resource consumption. Interesting future work includes investigating discrete task models, improved methods to generate accurate OCTs, and resource sharing with indirectly encountered RPs.

Acknowledgment

This work was supported in part by the National Institute of Information and Communications Technology (NICT), Japan.

References


Appendix

Theorem 1. Application of the same solution vector $X$ to problems P.1 and P.2 limits the error between the objective values to within the maximum inter-contact time of the support of $X$, i.e., $|t(X) - t'(X)| \leq IT_{\text{max-support}}(X)$.

Proof. $X$ is assumed to be a solution vector to task $ts$. Without loss of generality, the $k$-th coordinate of $X, x_k$, is considered. When $x_k$ is not in the support of $X (x_k$ is zero), both $T_L(x_k \times ts)$ defined in problem P.1 and $T_L'(x_k \times ts)$ defined in problem P.2 are zero. Consequently, the difference between their values is zero.

When $x_k$ is in the support of $X$, there are four cases for $T_L(x_k \times ts)$ depending on when the subtask is generated and finished: (1) the subtask is generated when RS is in contact with the $k$-th RP (GC), (2) the subtask is generated when RS is not in contact with the $k$-th RP (GNC), (3) the subtask is finished when RS is in contact with the $k$-th RP (FC), and (4) the subtask is finished when RS is not in contact with the $k$-th RP (FNC). A simple result that is used in the following proof is

$$-1 < A - [A] \leq 0. \quad (A·1)$$

(1) GC & FC

In this case, the time spent waiting for contacting opportunities, $T_{\text{wait}}$, for uploading input parameters and downloading output results is calculated using Eq. (10).

$$TL_k(x_k \times ts) - TL_k(x_k \times ts) = \frac{t_{s_i} \times x_k}{T_{\text{up-one}}} - \frac{1}{2} \left( \frac{t_{s_i} \times x_k}{T_{\text{up-one}}} - \frac{1}{2} \right) + \frac{t_{s_o} \times x_k}{T_{\text{up-one}}} \times 1 \times IT_k,$$

\begin{equation} \tag{A·2} 
\end{equation}

where $IT_k$ is the inter-contact time between RS and the $k$-th RP.

(2) GNC & FNC

In this case, the time spent waiting for contact opportunities, $T_{\text{wait}}$, for uploading input parameters and downloading output results is calculated using Eq. (7).

$$TL_k(x_k \times ts) - TL_k(x_k \times ts) = \left( \frac{t_{s_i} \times x_k}{T_{\text{up-one}}} - 1 \right) \frac{t_{s_i} \times x_k}{T_{\text{up-one}} - 1} + \frac{t_{s_o} \times x_k}{T_{\text{up-one}} - 1} \times IT_k,$$

\begin{equation} \tag{A·3} 
\end{equation}

(3) GC & FNC

In this case, the time spent waiting for contact opportunities, $T_{\text{wait}}$, for uploading input parameters is calculated
using Eq. (10), while that for downloading output results is calculated using Eq. (7).

\[
TL_k'(x_k \times ts) - TL_k(x_k \times ts) = \left( \frac{\sum \sum t_{k}x_{k}}{\sum t_{k}x_{k}} - 1 \right) - \left( \frac{\sum \sum t_{k}x_{k}}{\sum t_{k}x_{k}} - 1 \right) + 1) \times IT_k.
\]

(A-4)

Theorem 2. Suppose that \((x_0, x_1, x_2, ..., x_N)\) is the optimal solution vector \(X_{\text{optimal}}\) to the problem P.I and that \((y_{m1}, y_{m2}, y_{m3}, ..., y_{mN})\) is its corresponding order vector \(X_{\text{order}}\). If \(y_{mk}\) is in the support of \(X_{\text{optimal}}\), then all coordinates before \(y_{mk}\) in \(X_{\text{order}}\) are also in the support of \(X_{\text{optimal}}\).

Proof. Suppose \(y_{mi}\) is a coordinate before \(y_{mk}\) that is not in the support of \(X_{\text{optimal}}\). Without loss of generality, the subtask assigned to the \(m\)-th RP, \(x_{mi} \times ts\) is assumed to determine the value of \(t(X_{\text{optimal}})\). However, a fraction of the subtask can be offloaded from \(x_{mi}\) to \(x_{mi}\) to include \(x_{mi}\) in the support of a new solution vector, \(X_{\text{new}}\). The fraction of offloaded subtask is represented by \(y_{mi}\). The remaining subtask assigned to the \(m\)-th RP is \((1 - y_{mi}) \times x_{mi} \times ts\), and the subtask assigned to the \(m\)-th RP is \(y_{mk} \times x_{mk} \times ts\). As a result, the time consumed by the \(m\)-th provider, \(TL_m((1 - y_{mi}) \times x_{mi} \times ts)\), is reduced. At the same time, the time consumed by the \(m\)-th provider \(TL_{mk}(y_{mk} \times x_{mk} \times ts)\) is increased because no subtask is assigned to it in \(X_{\text{optimal}}\).

Since tasks are assumed to be partitioned seamlessly in this paper, \(y_{mk}\) is a continuous variable. Since \(x_{mi}\) is before \(x_{mk}\) in \(X_{\text{order}}\), there always exists a small enough \(y_{mk}\) to satisfy the inequation

\[
TL_{mk}(y_{mk} \times x_{mk} \times ts) < TL_{mk}(x_{mk} \times ts),
\]

(A-10)

since \(x_{mk} \times ts\) is the subtask assigned to the \(mk\)-th provider. This means that the \(m\)-th provider is able to finish its subtask before the \(mk\)-th provider while the \(n\)-th provider is able to finish its subtask in less time compared with the previous allocation in \(X_{\text{optimal}}\). Since coordinates other than \(x_{mi}\) and \(x_{mk}\) in \(X_{\text{optimal}}\) remain the same, \(X_{\text{new}}\) is a better solution vector than \(X_{\text{optimal}}\), which contradicts the hypothesis. Thus, \(x_{mi}\) is in the support of \(X_{\text{optimal}}\) and the proof is complete.

Theorem 3. Application of \(X_r\) and \(X_{optimal}\) to problem P.I restricts the error between \(t(X_r)\) and \(t(X_{optimal})\) to within twice the maximum inter-contact time of the support of \(X_{optimal}\), \(IT_{\text{max}}\).

Proof. Two inequations can be derived from Theorem 1:

\[
t(X_r) - t(X_{\text{optimal}}) \leq IT_{\text{max}}(X_{\text{optimal}}),\quad t(X_{\text{optimal}}) - t(X_r) \leq IT_{\text{max}}(X_{\text{optimal}}).
\]

(A-11)

Since \(X_{\text{optimal}}\) minimizes the objective value of problem P.I,

\[
\frac{t(X_r) - t(X_{\text{optimal}})}{IT_{\text{max}}(X_{\text{optimal}})} \leq t(X_r).
\]

(A-12)

Finally, since both \(x_{mi}\) and \(x_{mk}\) are the support of \(X_r\),

\[
|t(X_r) - t(X_{\text{optimal}})| = |IT_{\text{max}}(X_{\text{optimal}}) - IT_{\text{max}}(X_{\text{optimal}})| \leq IT_{\text{max}}(X_{\text{optimal}}).
\]

(A-13)

Combining Eqs. (A-11), (A-12), (A-13), (A-14) produces

\[
t(X_{\text{optimal}}) \leq t(X_r) \leq t(X_{\text{optimal}}) + IT_{\text{max}}(X_{\text{optimal}}),
\]

(A-14)

leading to

\[
t(X_r) - t(X_{\text{optimal}}) \leq IT_{\text{max}}(X_{\text{optimal}}) - IT_{\text{max}}(X_{\text{optimal}}) + IT_{\text{max}}(X_{\text{optimal}}) \leq 2 \times IT_{\text{max}}(X_{\text{optimal}}).
\]

(A-15)

The final induction step of Eq. (A-16) is based on Theorem 1. The proof is complete.
Wei Liu received the B.E. and M.E. degree in Software Engineering from Chongqing University, China, in 2006 and 2009. He is currently working toward the Ph.D. degree in Communications and Computer Engineering, Graduate School of Informatics, Kyoto University. His current research interests include green networks and pervasive computing. He received the Excellent Student Award from HP China in 2008.

Ryoichi Shinkuma received the B.E., M.E., and Ph.D. degrees in Communications Engineering from Osaka University, Japan, in 2000, 2001, and 2003, respectively. In 2003, he joined the faculty of Communications and Computer Engineering, Graduate School of Informatics, Kyoto University, Japan, where he is currently an Associate Professor. He was a Visiting Scholar at Wireless Information Network Laboratory (WINLAB), Rutgers, the State University of New Jersey, USA, from 2008 Fall to 2009 Fall. His research interests include network design and control criteria, particularly inspired by economic and social aspects. He received the Young Researchers' Award from IEICE in 2006 and the Young Scientist Award from Ericsson Japan in 2007, respectively. He is a member of IEEE.

Tatsuro Takahashi received the B.E. and M.E. in Electrical Engineering from Kyoto University, Kyoto, Japan, in 1973 and 1975 respectively, and Dr. of Engineering in Information Science from Kyoto University in 1997. He was with NTT Laboratories from 1975 to 2000, making R&D on high speed networks and switching systems for circuit switching, packet switching, frame relaying, and ATM. Since July 1, 2000, he is a Professor, Communications and Computer Engineering, Graduate School of Informatics, Kyoto University. His current research interests include high-speed networking, photonic networks and mobile networks. Prof. Takahashi received the Achievement Award from IEICE in 1996, the Minister of Science and Technology Award in 1998, and the Distinguished Achievement of Contributions Award from IEICE in 2011. He was a Vice President of the ATM Forum from 1996 to 1997, and the Chairman of the Network Systems (NS) Technical Group in the Communications Society of IEICE from 2001 to 2002. Prof. Takahashi is an IEEE Fellow.