sustainability

MDPI

Article

Computation Offloading Algorithm for Arbitrarily Divisible Applications in Mobile Edge Computing Environments: An OCR Case

Bo Li [1], Min He [1,*], Wei Wu [2], Arun Kumar Sangaiah [3] and Gwanggil Jeon [4]

1 School of Information Science and Engineering, Yunnan University, Kunming 650091, China; libo@ynu.edu.cn
2 College of Electronics and Information Engineering, Sichuan University, Chengdu 610065, China; wuwei@scu.edu.cn
3 School of Computing Science and Engineering, Vellore Institute of Technology (VIT), Vellore 632014, India; sarunkumar@vit.ac.in
4 Department of Embedded Systems Engineering, Incheon National University, Incheon 22012, Korea; gjeon@inu.ac.kr
* Correspondence: hemin@ynu.edu.cn

Received: 29 March 2018; Accepted: 8 May 2018; Published: 17 May 2018

Abstract: Divisible applications are a class of tasks whose loads can be partitioned into some smaller fractions, and each part can be executed independently by a processor. A wide variety of divisible applications have been found in the area of parallel and distributed processing. This paper addresses the problem of how to partition and allocate divisible applications to available resources in mobile edge computing environments with the aim of minimizing the completion time of the applications. A theoretical model was proposed for partitioning an entire divisible application according to the load of the application and the capabilities of available resources, and the solutions were derived in closed form. Both simulations and real experiments were carried out to justify this model.

Keywords: mobile edge computing; intelligent 5G services; computation offloading; divisible load theory

1. Introduction

In mobile computing environments, computation offloading, also known as cyber foraging [1], is an important practice to augment resource-constrained devices by partitioning and transferring some of their computation intensive workloads to other computing resources, with the aim of enhancing the capability of those resource-constrained devices and/or to save energy consumption [2–4].

In recent years, with the rapid development of mobile computing and wireless communications, mobile edge computing (MEC) was proposed to provide cloud-like services at the edge of the mobile network in close proximity to mobile users [5,6]. MEC is defined as a distributed mobile cloud computing environment where computing resources (i.e., edge servers) are installed within or near base stations or WiFi access points to provide computing services for mobile users in a limited communication range. The concept of MEC is highly similar to cloudlets [7] or fog computing [8]. Especially, in future 5G communication systems, mobile edge computing is regarded as an important enabling technology to offer computing and storage services for various IoT devices and applications [6,9], and it will be helpful to provide quality of services (QoS) guarantees for various over-the-top (OTT) services [10,11], which will be beneficial for both OTT service providers and network operators [12].

Many technologies have been proposed to enable computation offloading in various mobile computing environments with the aim of addressing the challenges involved when offloading applications from mobile devices to edge services providers. Among all the challenges, how to

partition applications into components and how to allocate available resources for such components to gain certain optimizing targets are the most important topics that have drawn lots of attention [13–16].

In existing partition and allocation related research, offloadable applications are usually regarded as being composed of multiple sections with various granularity levels and dependencies [13]. For example, an entire application may be partitioned at the methods level, the threads level, the classes level, or the tasks level, and the processing of these modules may be constrained by some precedence relations. Usually, such modularly divisible applications are represented as task interaction graphs, whose vertices correspond to the modules, and whose edges represent interactions or precedence relations between modules. Some graph- or network-based theoretical models, as well as algorithms, have already been proposed for partitioning and distributing such applications among available resources to achieve certain objectives, e.g., to reduce completion time, to reduce network overhead, or to save energy.

For modularly divisible applications, the number of modules, the size of each module, and the interactions are assumed to be predefined before making partition and distribution decisions. This assumption holds for many mobile applications. However, except if it is modularly divisible, an application may also be arbitrarily divisible [13,17–19], i.e., the number of portions and the size of each portion are not predefined before making partition and distribution decisions—actually, they are determined by the decisions. Compared with modularly divisible applications, arbitrarily divisible ones are more suitable for being processed among a set of dynamically organized resources: as the number and capabilities of available resources change, an arbitrarily divisible application can be divided into any number and size of portions to match available resources. Applications satisfying this arbitrarily divisible property include data processing of massive experimental data, matrix computations, signal processing applications, etc. [20]. Optical Character Recognition is a common method to convert images of typed, handwritten or printed text into digitized text. Since OCR images can be split into many slices and allocated among a group of processing resources to process in parallel, OCR is a kind of typical arbitrarily divisible application suitable for dynamic mobile computing environments. In [21], by using divisible load theory, the authors derived a hybrid partitioning scheme to solve the optical character recognition problem in a network of workstations, and presented a closed-form expression for the number of neurons assigned to each processor.

For offloading arbitrarily divisible applications in mobile edge computing environments, two important decision problems are about how to partition the applications into smaller sections and how to allocate these smaller sections to available computing units, so that the computation is completed in the shortest possible time. As far as the authors know, up to now, very little research has been carried out for these two basic problems.

In this paper, based on the Divisible Load Theory [20], we proposed a theoretical model for partitioning the arbitrarily divisible application in mobile edge computing environments. A set of linear functions were proposed to model the problem of the partitioning and allocating of an arbitrarily divisible application among a group of available resources, with the goal of minimizing the completion time of the entire application. In the model, all factors related to both communication and computation cost of the application and its segments were taken into account, including the amount of computation required by the application and its segment, the computation capability of the resources, the start-up delay of the resources, the amount of data to be transferred, the bandwidth and network latency between resources. In order to justify the feasibility and performance of the model, both simulations and real experiments were carried out.

The main contributions of this study include:

- We propose a theoretical model for offloading arbitrarily divisible applications in mobile edge computing environments;
- We derive closed-form expressions for the sizes of segments and the minimum makespan of the entire application;

- Based on the closed-form expressions, a heuristic is proposed for partitioning and distributing arbitrarily divisible applications in mobile edge computing environments.
- One method is proposed to estimate these decision parameters in real environments.

2. Related Work

Divisible load applications are a class of tasks whose loads can be partitioned into some arbitrary size, in which each part can be executed independently by a processor [20,22]. In recent years, many researchers have discovered a wide variety of divisible applications in the area of parallel and distributed computing [23], sensor networks [24,25], grid computing [26] and cloud computing [27–30], and have carried out ample research on the scheduling problem of divisible load applications, also called divisible load theory. Divisible load applications may be those applications that are divisible in nature, or they may be an approximation composed of many relatively small independent tasks [31].

In the research of divisible load theory, the network topology is very important to determine the scheduling model. Up to now, various topologies were proposed [22], including single-level tree, multi-level tree, hypercubes, arbitrary graphs, daisy chain networks and meshes. A unified discussion of divisible load scheduling results for star and tree networks was also presented [23].

Since the single level tree network (or star network) can generalize any topology, it can be regarded as a fundamental interconnection topology for the research of divisible applications. For the computing environments considered in this paper, it is assumed that all available surrogate nodes are connected to the client node directly and a single-level tree topology is formed. In the following, we are interested in the research related to schedule divisible applications in single-level tree networks.

A non-linear programming model for scheduling divisible applications composed of M equal-sized independent tasks under the bounded multi-port model and a heuristic called bandwidth-aware task scheduling algorithm was proposed in 2014 [28]. However, the applications considered by the authors cannot be divided arbitrarily, thus they are not suitable for our paper. Furthermore, when calculating the cost of processing some data on other nodes, network latency and start-up time of virtual machines were not taken into account. This is not realistic for distributed opportunistic computing environments. In this paper, network latency and start-up time are considered in the mathematical model.

The scheduling problem for nonlinear divisible applications in a single level tree network with a collective communication model was considered [18]. In that paper, both computation start-up delays and communication start-up delays were considered. However, it was assumed that the computation time function for any given processor is a polynomial equation in load fraction size. This is quite different from the linearly divisible application model in our paper.

Similar to the solutions proposed to split divisible OCR images into slices and allocate them among a group of computing units in this paper, the problem of how to split and schedule divisible images in visual sensor networks was considered [25]. It was also assumed that there are linear correspondences among the number of pixels in a slice, the size of the slice in bits and the computation amount required. An optimal offloading strategy is formally characterized under different networking and communication paradigms. However, network latency and start-up time were not considered.

A scheduling model to allocate a divisible application with many trunks to N available computing workers in a sequence fashion was proposed [29], so that all computing workers complete their computations at the same time. The problem of distributing a divisible application in sequence among multiple processor nodes with limited buffer sizes was also considered [32]. However, in our paper, it is assumed that the fractions are processed by both the master and computing workers in parallel.

For network computing environments, the process of communication or computation demands an initial start-up time. To schedule divisible tasks in a bus architecture with multiple processor and a bus control unit (BCU), the start-up costs could be included in both the communication and computation times of the loads [33]. The BCU was assumed to transmit load fractions to the processors one by one.

This is quite different from the collective communication model in this paper that can transmit load fractions in parallel.

Different from existing research that focuses on the partitioning and allocation of one single divisible application, some research has investigated the problem of scheduling multiple divisible applications. The problem of scheduling multiple divisible applications in single-level tree networks was investigated and an efficient static scheduling algorithm to near-optimally distribute multiple loads among all processors so that the overall processing time of all jobs is minimized [34]. Besides, the problem of scheduling a group of divisible applications with different deadline and cost properties into a homogeneous cloud computing environment, and a closed-form solution for the load fractions to be assigned to each processor was proposed [30]. Based on those results, both computation and energy cost could be considered [35]. However, in these two papers, communication cost was not taken into account.

Compared with existing research, this paper proposed a theoretical model for partitioning the arbitrarily divisible application in mobile edge computing environments. In this model, both communication and computation cost of the application and its segments were considered and closed-form expressions for the sizes of segments and the minimum makespan of the entire application were derived. Based on the model, a heuristic was proposed to partition and allocate segments of the arbitrarily divisible application in mobile edge computing environments.

Different from existing research related to scheduling divisible applications in single-level tree networks that are reported in [28] and other references, in this paper, for OCR-like arbitrarily divisible applications, different factors were taken into account (see Table 1), and the mathematical model proposed is more suitable to describe the processing of OCR-like arbitrarily divisible applications in mobile computing environments.

Table 1. Differences from existing research.

References	The Model Considered in the Reference(s)	The Model Considered in This Paper
[28]	(1) Workload can be divided into M equal-sized independent tasks. (2) When calculating the cost of processing, network latency and start-up time of virtual machines were not considered.	(1) Workload can be divided arbitrarily.
[18]	The computation time function for any given processor is a polynomial equation in load fraction size.	The computation time function for divisible application is linearly proportional to the load fraction size.
[25]	Network latency and start-up time were NOT considered.	Network latency and start-up time of virtual machines were taken into account in the mathematical model.
[29,32]	A divisible application is scheduled to N available computing workers in a sequence fashion.	A divisible application is scheduled to N available computing workers in parallel.
[33]	Load fractions are transmitted from the control unit to available processors one by one.	Load fractions are transmitted from the client node to available surrogates in parallel by using the collective communication model.
[30,34]	Focus on static scheduling of a group of applications at the same time.	Focus on dynamic scheduling of one single divisible application.
[35]	(1) Both computation and energy cost were considered. (2) Communication cost was not taken into account.	(1) Only computation cost is considered. (2) Communication cost-related parameters (bandwidth and latency) were taken into account.

Different from [21] in which OCR applications were used for case study of the proposed hybrid partitioning scheme for the multilayer perceptron network trained using back-propagation algorithm on network of workstations, in this paper, OCR applications were used for a case study of the offloading method of arbitrarily divisible workloads in mobile edge computing environments. We also

derived a closed-form expression for the partition and allocation of the workloads to available computing resources.

3. Model and Algorithm for Offloading Arbitrarily Divisible Applications

Optical Character Recognition is a kind of typical arbitrarily divisible application suitable for dynamic mobile computing environments. This section presents an example OCR application, and illustrates the processes to partition it into different sections and distribute them to resources. Then, based on the divisible load theory, a theoretical model was proposed for making offloading decisions for such arbitrarily divisible applications.

3.1. Optical Character Recognition: A Typical Arbitrarily Divisible Application

Optical Character Recognition is a kind of typical arbitrarily divisible application suitable for dynamic mobile computing environments. It is a common method to convert images of typed, handwritten or printed text into machine-encoded text, so that it can be electronically edited, searched, stored, and used in machine processes, such as machine translation and augmented reality. Figure 1 presents an example of partitioning an OCR image as the number of available resources changes. In the example, as the number of resources changes from one, to two, to four, the task is partitioned into one, two and four pieces; each piece is for one resource. In this example, we also assume the parameters of the resources are the same, thus the image is partitioned equally. If the parameters of the resources change to be not the same, the partition can be accordingly adjusted. Up to now, some networked computing systems have been reported to evaluate their performance by using OCR instances [36].

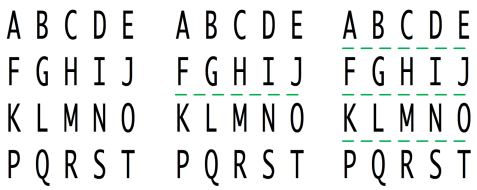

Figure 1. An example of an OCR image partition when the number of available resources is one (**left**), two (**middle**), and four (**right**).

A usage scenario for processing OCR application shown in Figure 1 is as follows. For example, when a group of people hang out together and one of them wants to recognize and translate a tourist guiding file in a foreign language into his/her native language, he or she may use a smart phone to take a picture of the file and then process it on the smart phone. When the size of the file is large, the recognition and translation process will take a long time and hurt the quality of experiences of the users. Assume that there are some computing devices in the vicinity that are connected through some wireless network and can work cooperatively; we can partition the picture into many pieces and send some of them to other available devices to process in parallel. In this way, the processing time will be much shorter and the quality of user experience will be much better.

In the OCR example illustrated in Figure 1, it is assumed that the partition of the OCR image is not completely arbitrary—it is nearly arbitrary, in that it is necessary to avoid partitioning the image across characters, to guarantee that different partition solutions can generate the same final result for the entire image.

3.2. Model and Algorithm for Offloading OCR-Like Applications

The computing environment to be considered here consists of one mobile client node and a set K of available resources (also known as *surrogate nodes* or *edge servers*), as shown in Figure 2. Without loss of generality, we assume that the nodes are indexed from 0 (the client node) to $k = |K|$. Assume the client node acquires an image as shown in Figure 1 and needs to convert it into text. The client node

may process the image itself. However, it can also split the image into many fractions and distribute parts of them to the surrogate nodes to process.

In order to make the best use of those available surrogate nodes, how to partition and allocate a divisible OCR image among the client node and the surrogate nodes to decrease the total completion time becomes an important problem. Notably, in Figure 1, the application is divided into equally sized fractions. However, this partition will result in poor performance for environments with heterogeneous processors and links. In the following, we will provide a mathematical model for this scheduling problem in heterogeneous environments. The notations and definitions that are used throughout this paper are shown in Table 2.

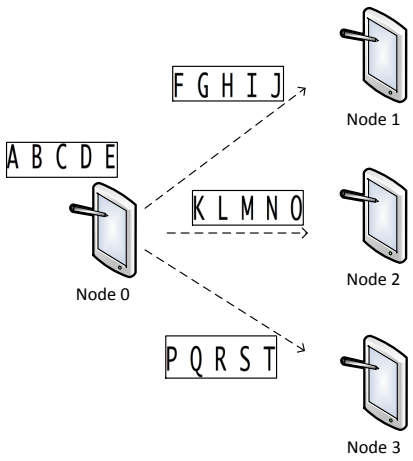

Figure 2. An example of allocating the divisible OCR image in Figure 1 to process in a mobile edge computing environment consisting of one client (node 0) and three edge servers (nodes 1 to 3).

For a typical OCR image of x pixels, assume it is split into $k+1$ horizontal (or vertical) slices with sizes $x_0, x_1, ..., x_k$. For a slice with size x_i, assume there is a linear function $y_{ij} = f_j(x_i) = a_j x_i + b_j$ that can be used to estimate the processing time y_{ij} when the slice is processed on node j. In this function, a_j and b_j are dependent on node j, not on x_i, meaning that $f_j()$ is determined by node j. Given the parameters of the nodes, the sizes of the slices, and the mapping of the slices to the nodes, we can calculate the processing time of every slice and the final completion time of the entire image.

When an OCR segment is processed on the client node, the completion time will be determined by the size of the segment x_i, the start-up time F_0 and the computation capability of the node P_0. Here, the start-up time is defined as a constant time interval needed to prepare for the processing of an OCR segment. For each node, its start-up time is constant, i.e., it will not change as the node processes different OCR images.

When an OCR segment is processed on a surrogate node, the entire process includes three phases:

(1) The client node sends the OCR segment to the surrogate node;
(2) After receiving the entire OCR segment, the surrogate node start processing the segment;
(3) The surrogate node returns the result to the client node.

The entire completion time of the segment is the sum of processing time of the three phases.

In existing research, it is usually assumed that the client node distributes the load fractions to other nodes one by one. However, in this paper, as in [18], it is assumed that the client node is equipped with a front-end (also known as a communication co-processor) supporting collective communication model. After using the front-end, the client node may process its computation load and communicate with other nodes at the same time. By using this collective communication model, the client node can distribute/receive data from/to the surrogate nodes concurrently. For example, as soon as the partition decision is made, the client node may distribute the load fractions to corresponding nodes concurrently by using the broadcast model.

Assume there are k surrogates. For surrogate $S_i, i = 1 \rightarrow k$, its computation capability is P_i MIPS (millions of instructions per second), its start-up time is F_i seconds, the bandwidth connecting to the client node is B_i MB/s (megabyte per second), the round trip network latency is L_i seconds, and the computation amount of the OCR segment is C_i MI (millions of instructions). Considering the fact that the computation amount of recognizing an OCR image is usually linearly proportional to the size of the image, let $D_i = \alpha * C_i$, where α is defined as the ratio of size D_i pixels of the image to its computation amount C_i. For different OCR applications, α may change, depending on the image itself and the technologies for pro-processing, character recognition, and post-processing.

When scheduling divisible applications in distributed computing environments, the start-up costs in both the communication and computations times must be considered. Here, these overhead factors are considered as additive components. Assume the start-up cost in the computation time of node i is defined as F_i. For the start-up cost in the communication time of the link between node 0 and node i, assume it is constant. Considering the fact that the network latency of the link between node 0 and node i is also regarded as a constant, we use L_i to express the sum of the round trip latency and the start-up times of the link between node 0 and node i. In the following, for simplicity, we call L_i the latency of the link, instead of calling it the sum of the latency and the start-up time of the link.

Given the information as above, when an OCR segment with computation amount C_i is transferred from the client node to surrogate node S_i to process, we can estimate the time needed in the three phases, and thus can estimate the entire processing time: In phase (1), the time needed will be the transmission time of the image plus the one way network latency, i.e., be $D_i/B_i + L_i/2 = \alpha * C_i/B_i + L_i/2$; In phase (2), the time needed will be the start-up time plus the recognition time of the OCR segment, i.e., be $F_i + C_i/P_i$; In phase (3), the time needed will be the transmission time of the result plus the one way network latency. Usually the result of an OCR application is in plain text and, compared with the size of the image, its size is very small. So, when calculating the time used to transfer the result, the size of the result will be neglected, and the time will be dominated by the one way network latency. So, the time needed for phase (3) can be expressed as $L_i/2$.

Table 2. Notations and definitions.

Notation	Definition
C (MI)	the total computation amount of the divisible application
C_i (MI)	the computation amount allocated for node $i, i = 0 \rightarrow k$
α	the ratio of the computation amount and the size of data
k	the number of surrogate nodes available
P_i (MIPS)	the computation capability of node $i, i = 0 \rightarrow k$
B_i (MB/s)	the bandwidth of the link between nodes i and node 0, $i = 1 \rightarrow k$
F_i (s)	the start-up cost in communication time of node $i, i = 0 \rightarrow k$
L_i (s)	the round-trip network latency of the link between nodes 0 and $i, i = 1 \rightarrow k$

Finally, assume an OCR application is processed in an opportunistic environment with one client node and k surrogate nodes. In order to process the application as soon as possible, the client node will partition and allocate the application to all available resources in a load balance way. Assuming all the results will be returned to the client node at the same time, we can get the following formula:

$$\begin{aligned} \frac{C_0}{P_0} + F_0 &= \frac{C_1}{P_1} + \frac{\alpha * C_1}{B_1} + F_1 + L_1 \\ &= \frac{C_2}{P_2} + \frac{\alpha * C_2}{B_2} + F_2 + L_2 \\ &= \ldots \\ &= \frac{C_k}{P_k} + \frac{\alpha * C_k}{B_k} + F_k + L_k \end{aligned} \tag{1}$$

where $C_0 + C_1 + \ldots + C_k = C$.

From Formula (1), following formula can be derived:
For $\forall i \in 1 \rightarrow k$

$$\begin{aligned} \frac{C_i}{P_i} + \frac{\alpha * C_i}{B_i} + F_i + L_i &= \frac{C_i * B_i + \alpha * P_i * C_i}{P_i * B_i} + F_i + L_i \\ &= \frac{(B_i + \alpha * P_i)C_i}{P_i * B_i} + F_i + L_i \\ &= \beta_i * C_i + F_i + L_i \end{aligned} \tag{2}$$

where $\beta_i = (B_i + \alpha * P_i)/(P_i * B_i)$.

Considering Formulas (1) and (2), we have

$$\begin{aligned} C_0 &= P_0 * \beta_i * C_i + P_0 * (F_i + L_i - F_0) \\ &= \gamma_i * C_i + \delta_i \qquad (3) \\ C_i &= (C_0 - \delta_i)/\gamma_i \qquad (4) \end{aligned}$$

where $\gamma_i = P_0 * \beta_i$ and $\delta_i = P_0 * (F_i + L_i - F_0)$.

Now, by introducing Formula (4) into $C_0 + C_1 + ... + C_k = C$, we have

$$\begin{aligned} C_0 + C_1 + ... + C_k = {} & C_0 + (C_0 - \delta_1)/\gamma_1 \\ & + (C_0 - \delta_2)/\gamma_2 \\ & + ... \\ & + (C_0 - \delta_k)/\gamma_k \end{aligned}$$

and

$$C_0 = \frac{C + (\delta_1/\gamma_1 + \delta_2/\gamma_2 + ... + \delta_k/\gamma_k)}{1 + 1/\gamma_1 + 1/\gamma_2 + ... + 1/\gamma_k} \tag{5}$$

Finally, we have

$$\begin{aligned} makespan &= C_0/P_0 + F_0 \\ &= \frac{C + (\delta_1/\gamma_1 + \delta_2/\gamma_2 + ... + \delta_k/\gamma_k)}{P_0(1 + 1/\gamma_1 + 1/\gamma_2 + ... + 1/\gamma_k)} + F_0 \end{aligned} \tag{6}$$

where *makespan* is defined as the time difference between the start and finish of the offloaded task.

Based on above models, Algorithm 1 was proposed here for partitioning a divisible OCR application among a group of resources.

Algorithm 1: Algorithm for adaptively partitioning and allocating a divisible OCR application.

input : Application Info: C, α
Resource Info: $P_0, F_0, P_i, F_i, B_i, L_i, i = 1 \rightarrow k$
output: Offloading decision, $C_i, i = 0 \rightarrow k$ and *makespan*

(1) Find out the number k of available resources and their parameters $P_i, F_i, B_i, L_i, i = 1 \rightarrow k$;
(2) Calculate and partition the task for the client node based on Formula (5);
(3) Calculate and partition the task for each surrogate based on Formula (4);
(4) Calculate the makespan based on Formula (6).

In the above theoretical analysis, the sizes of slices can be calculated by Formula (4). However, in some cases, if the image is partitioned exactly according to the resultant sizes, the boundary of slices may cross some characters. This will impact the final recognition result of the entire image. To avoid

such cases, try to partition the image into slices according to the sizes in Formula (4), at the same time, do not cross characters. In practice, this can be implemented by considering the line spacing of the image and trying to make the size of the slices be as close as possible to the theoretical results.

4. Experiments and Results

In this section, we present a set of experiments to validate the mathematical model and to study the performance of the proposed algorithm in both simulation and real computing environments.

4.1. Simulation Environments

In simulation, it is assumed that the computing environment is composed of a mobile client node and some surrogate nodes in the vicinity, connected via wireless network. When an OCR task appears on the client node, the client node will try to find out the number and the parameters of available surrogates, and then, it will decide how to partition the task and allocate the subtasks to available surrogates, so as to make the maximum completion time of the subtasks as minimum as possible.

In simulation, it is assumed that the maximum number of available surrogate nodes is 5, and the parameters for the application and the nodes are listed in Table 3. For the surrogate nodes, their performance parameters, including the computation capacity (P_i), the bandwidth to the client node (B_i), and the network latency (L_i), are random variables uniformly distributed in given ranges. In Table 3, $U([x_1, x_2, ..., x_n])$ represents a uniform distribution within x_1, x_2, to x_n. For example, B_i subjects to $U([2, 4, 6, 8, 10])$, meaning that the bandwidth from any one of those k surrogate nodes to the client node is chosen randomly and uniformly within 2, 4, 6, 8, and 10 MB/s. Since the parameters for every surrogate node are chosen independently, for different surrogate nodes, their parameters are also different. So, these variable parameters can be used to describe the characteristics of surrogate nodes in a dynamic computing environment. When all the parameters of the application and the resources are known, we can calculate the partition and the makespan based on the model and Formula in Section 3.

Table 3. Parameters for simulation.

Parameter	Description	Values
C	the total computation amount of the divisible application	100 MI
α	the ratio of the computation amount and the size of data	0.2
k	the number of surrogate nodes available	0 to 5
P_0	The computation capacity of the client node	5 MIPS
$P_i, i = 1 \rightarrow k$	the computation capability of node i	$U([5, 10, 15, 20])$ MIPS
$B_i, i = 1 \rightarrow k$	the bandwidth of the link between nodes i and 0	$U([2, 4, 6, 8, 10])$ data units per second MB/s
$L_i, i = 1 \rightarrow k$	the round-trip latency between nodes 0 and i	$U([0.1, 0.2, 0.3, 0.4, 0.5])$ second

$U([x_1, x_2, ..., x_n])$ represents a uniformly distribution within x_1, x_2, ..., x_n.

In the following example, assume the processing capacity, bandwidth and network latency for the surrogates are [20 20 20 15 5] MIPS, [10 10 10 8 2] MB/s, and [0.5 0.5 0.5 0.4 0.1] second individually, i.e., for the ith surrogate, its processing capacity, bandwidth and latency are the value of the ith element of the vectors. For example, the processing capacity, bandwidth and latency of the first surrogate node are 20 MIPS, 10 MB/s, and 0.5 s individually.

In order to study the influence of the number of surrogate nodes on the partition of the task and the makespan, at first, the surrogate nodes were sorted decreasingly according to the value of γ in Formula (4). Then, assume the number of available surrogate nodes increases from zero (i.e., only the client node itself available), to one (i.e., only one surrogate node is available), and finally to five (i.e., all surrogate node are available).

Figures 3 and 4 present the partition and makespan results when different number of surrogates are available. In Figure 3, as the number of resource is 1, i.e., the client node itself, the whole application is allocated to the client node to execute, thus the amount of computation on the client node is 100 MI. In the second column, as the first surrogate becomes available and the number of resources increases

to 2, the application is partitioned into two parts: 27.8 MI on the client node and 72.2 MI on the first surrogate. In the third column, as the second surrogate becomes available, the application is partitioned into three parts: 17 MI on the client node and 41.5 MI on each surrogate node. Here, because the two surrogates have the same parameters, this leads to the same partition for them. The partitions with other numbers of available resources are illustrated in corresponding columns. Finally, as the number of available resources increases to 6, we got the partition as shown in the last column.

Figure 4 presents the makespan results as the number of available resources changes. As can be seen, as the number of available resource increases, the makespan of the application will decrease. For each column, the makespan result is consistent with the corresponding partition result in Figure 3 divided by the processing capacity of the client node. This can be concluded based on Formula (6).

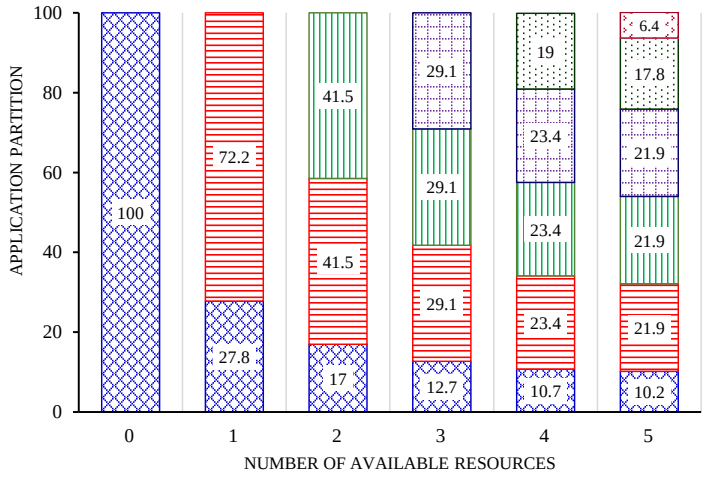

Figure 3. An example of an OCR application partition against different number of available resources, without considering network latency.

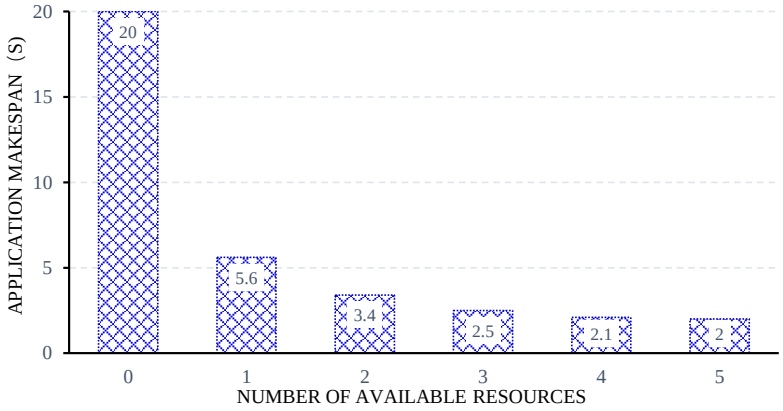

Figure 4. The makespan of the application executed on different number of available resources, without considering network latency.

4.2. Real Experiments

A real experimental environment was built up to verify the feasibility and the performance of the model proposed in Section 3, in which an android phone acting as the client node, and another two laptops acting as the surrogate nodes, were connected via an 802.11b Wireless access point. The parameters of these devices are listed in Table 4.

Table 4. Real experimental environments.

Device	Model	CPU	RAM	OS
client node	Huawei H30	Kirin910 1.6 GHz	1 GB	Android 4.0
surrogate node 1	Lenovo M40	I3-4030U 1.9 GHz	2 GB	Windows 7
surrogate node 2	Lenovo N50	I5-5200U 2.2 GHz	4 GB	Windows 7

In practice, in order to adopt the partition and allocation model to process a real OCR task, except for the size of the OCR image being known, any other parameters, including the computation amount of the task, and the parameters of the resources available (the computation capability, the bandwidth, and the latency) are deterministic. However, in real environment, it is not easy to predict them accurately. To handle this problem, a method that is a compromise is proposed as follows.

In the mathematical model in Section 3, it is assumed that, for a typical divisible OCR image, when it is partitioned into n (>1) segments of size x_1, x_2, to x_n individually, and every segment is allocated to be processed on different resource, the completion time t_i for the segment with size x_i can be estimated by using a linear function, i.e.,

$$\forall i = 0 \to k, t_i = f_i(x_i) = a_i x_i + b_i \tag{7}$$

Considering that the optimization object is to make the completion time of all the segments equal, the following equations can be constructed:

$$f_i(x_i) = f_j(x_j), \forall i, j \in 0 \to k \tag{8}$$

$$x = \sum_{i=0}^{k} x_i \tag{9}$$

By introducing Equation (7) into Equations (8) and (9), we can get:

$$x_0 = (x - \sum_{i=1}^{k} \frac{b_0 - b_i}{a_i}) / \sum_{i=0}^{k} a_0/a_i \tag{10}$$

$$x_i = a_0 x_0 / a_i + (b_0 - b_i)/a_i \tag{11}$$

$$t = a_0 x_0 + b_0 \tag{12}$$

where x_0 is the size of sub image processed on the client node, and x_i the size on the ith resource available, and t the completion time of the task.

For the method as above, how to get the value of a_i in Equation (7) is the most important step. In practice, a group of OCR files can be used as training files and processed on every available resource. In this way, a_i can be statistically calculated. After this training process, when a new OCR task arrives, we can partition and allocate it to available resources according to Equations (10) and (11), and estimate its completion time according to Equation (12).

In experiments, a set of training tasks were firstly processed on every resource. Assume the size of the OCR images ranges from 100×100 to 500×500 pixels. At first, these images are recognized on the smart phone, and their completion times were recorded. Then these images were transferred individually from the smart phone to the other two laptops to process, and the time points at which the results returning to the smart phone were recorded as the completion time. Finally, it is statistically found that $t_0 = 0.19x + 2850$, $t_1 = 0.072x + 1890$ and $t_2 = 0.039x + 1630$, in which x representing the size of an OCR image, and t_0, t_1, and t_2 representing the completion time of recognizing the image on the smart phone, on surrogate 1 and on surrogate 2 individually.

After predicting, as a new OCR task arrives at the smart phone, the scheduler will decide how to partition and allocate the task according to Equations (10) and (11). Figure 5 presents the completion time of the OCR tasks with sizes ranging from 100×100 to 500×500 pixels. When the tasks were

executed only on the smart phone, as the sizes of the tasks increase, their makespan increases quickly (see the blue curve). However, when another two laptops were used for processing, we can partition the tasks according to Equations (10) and (11), and predict their makespan according to Equation (12) (see the dashed green curve). Finally, the actual makespans of the tasks are illustrated in the red curve. As can be seen, when another two surrogate nodes are used for processing, the completion time of the OCR images become much lower.

In addition, in Figure 5, as the sizes of the images increase, generally speaking, the actual makespan results in the red curve and the blue one increase linearly. However, this linearity cannot be guaranteed for every instance. This phenomenon can be explained as follows: for an OCR image, the processing requirement is related not only to the size of the image, but also to the complexity of the processing needed. If the size of the OCR image increases and the complexity of the processing needed decreases, the completion time may decrease. So, as the size of the images increase from small to large, the completion time is expected to increase linearly with fluctuation.

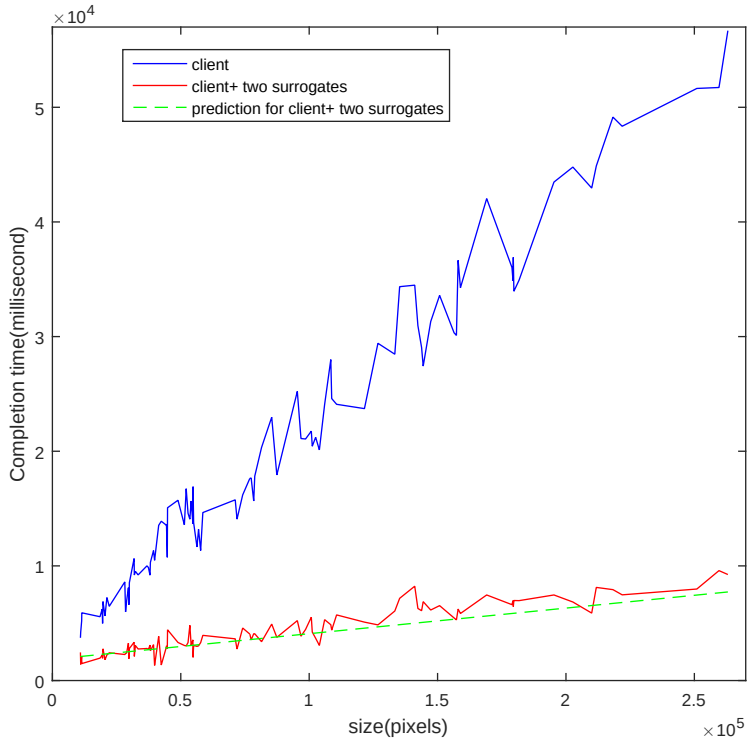

Figure 5. The makespan of the adaptive partition and allocation algorithm against the size of the tasks.

5. Conclusions

In this paper, OCR applications, as typical of divisible applications, were considered to be processed in a computing environment consisting of a group of mobile devices connected with single level star topology wireless network. By using the proposed scheduling model, the load of a divisible application can be partitioned and allocated based on both the communication and computation costs, so that the final completion time of the whole application can be made as short as possible. Based on the results of both the simulations and real experiments, it can be seen that the model proposed is very effective and practical for partitioning and allocating divisible applications in opportunistic computing environments. This model can be extended for scheduling divisible applications in other similar computing environments.

In this study, we focus on the benefit of offloading in terms of completion time, without considering the energy consumption that may be incurred. In future work, energy consumption can also be considered as an important factor for making offloading decisions.

Author Contributions: B.L., M.H., W.W. and G.J. conceived and designed the experiments; B.L. and M.H. performed the experiments, analyzed the data; W.W., A.K.S. and G.J. contributed analysis tools; B.L. and M.H. wrote the paper.

Acknowledgments: This work was partially supported by National Natural Science Foundation of China (No. 61562092, 61463049).

Conflicts of Interest: The authors declare no conflict of interest. The founding sponsors had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, and in the decision to publish the results.

Abbreviations

The following abbreviations are used in this manuscript:

MEC	Mobile Edge Computing
OCR	Optical Character Recognition
IoT	the Internet of Things

References

1. Balan, R.; Flinn, J.; Satyanarayanan, M.; Sinnamohideen, S.; Yang, H.I. The case for cyber foraging. In Proceedings of the 10th Workshop on ACM SIGOPS European Workshop, Saint-Emilion, France, 1 July 2002; ACM: New York, NY, USA, 2002; pp. 87–92.
2. Li, B.; Pei, Y.; Wu, H.; Shen, B. Heuristics to allocate high-performance cloudlets for computation offloading in mobile ad hoc clouds. *J. Supercomput.* **2015**, *79*, 3009–3036. [CrossRef]
3. Abolfazli, S.; Sanaei, Z.; Ahmed, E.; Gani, A.; Buyya, R. Cloud-based augmentation for mobile devices: Motivation, taxonomies, and open challenges. *IEEE Commun. Surv. Tutor.* **2014**, *16*, 337–368. [CrossRef]
4. Lee, B.D.; Lim, K.H.; Choi, Y.H.; Kim, N. An adaptive computation offloading decision for energy-efficient execution of mobile applications in clouds. *IEICE Trans. Inf. Syst.* **2014**, *97*, 1804–1811. [CrossRef]
5. Mao, Y.; You, C.; Zhang, J.; Huang, K.; Letaief, K. A Survey on Mobile Edge Computing: The Communication Perspective. *IEEE Commun. Surv. Tutor.* **2017**, *19*, 2322–2358, doi:10.1109/COMST.2017.2745201. [CrossRef]
6. Chen, M.; Hao, Y. Task Offloading for Mobile Edge Computing in Software Defined Ultra-dense Network. *IEEE J. Sel. Areas Commun.* **2018**, doi:10.1109/JSAC.2018.2815360. [CrossRef]
7. Lewis, G.A.; Echeverria, S.; Simanta, S.; Bradshaw, B.; Root, J. Cloudlet-based cyber-foraging for mobile systems in resource-constrained edge environments. In Proceedings of the ICSE Companion 2014 Companion Proceedings of the 36th International Conference on Software Engineering, Hyderabad, India, 31 May–7 June 2014; pp. 412–415.
8. Bonomi, F.; Milito, R.; Zhu, J.; Addepalli, S. Fog Computing and Its Role in the Internet of Things. In Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing (MCC '12), Helsinki, Finland, 17 August 2012; ACM: New York, NY, USA, 2012; pp. 13–16, doi:10.1145/2342509.2342513. [CrossRef]
9. Neto, J.; Yu, S.; Macedo, D.; Nogueira, J.; Langar, R.; Secci, S. ULOOF: A User Level Online Offloading Framework for Mobile Edge Computing. *IEEE Trans. Mob. Comput.* **2018**, doi:10.1109/TMC.2018.2815015. [CrossRef]
10. Giatsoglou, N.; Ntontin, K.; Kartsakli, E.; Antonopoulos, A.; Verikoukis, C. D2D-Aware Device Caching in mmWave-Cellular Networks. *IEEE J. Sel. Areas Commun.* **2017**, *35*, 2025–2037, doi:10.1109/JSAC.2017.2720818. [CrossRef]
11. Datsika, E.; Antonopoulos, A.; Zorba, N.; Verikoukis, C. Software Defined Network Service Chaining for OTT Service Providers in 5G Networks. *IEEE Commun. Mag.* **2017**, *55*, 124–131, doi:10.1109/MCOM.2017.1700108. [CrossRef]
12. Antonopoulos, A.; Kartsakli, E.; Perillo, C.; Verikoukis, C. Shedding Light on the Internet: Stakeholders and Network Neutrality. *IEEE Commun. Mag.* **2017**, *55*, 216–223, doi:10.1109/MCOM.2017.1600417. [CrossRef]
13. Liu, J.; Ahmed, E.; Shiraz, M.; Gani, A.; Buyya, R.; Qureshi, A. Application partitioning algorithms in mobile cloud computing: Taxonomy, review and future directions. *J. Netw. Comput. Appl.* **2015**, *48*, 99–117. [CrossRef]
14. Liu, L.; Chang, Z.; Guo, X.; Mao, S.; Ristaniemi, T. Multiobjective Optimization for Computation Offloading in Fog Computing. *IEEE Internet Things J.* **2018**, *5*, 283–294, doi:10.1109/JIOT.2017.2780236. [CrossRef]

15. Wu, H. Multi-Objective Decision-Making for Mobile Cloud Offloading: A Survey. *IEEE Access* **2018**, *6*, 3962–3976, doi:10.1109/ACCESS.2018.2791504. [CrossRef]
16. Huang, C.M.; Chiang, M.S.; Dao, D.T.; Su, W.L.; Xu, S.; Zhou, H. V2V Data Offloading for Cellular Network Based on the Software Defined Network (SDN) Inside Mobile Edge Computing (MEC) Architecture. *IEEE Access* **2018**, *6*, 17741–17755, doi:10.1109/ACCESS.2018.2820679. [CrossRef]
17. Camp, T.; Boleng, J.; Davies, V. A survey of mobility models for ad hoc network research. *Wirel. Commun. Mob. Comput.* **2002**, *2*, 483–502. [CrossRef]
18. Suresh, S.; Kim, H.; Run, C.; Robertazzi, T.G. Scheduling nonlinear divisible loads in a single level tree network. *J. Supercomput.* **2012**, *61*, 1068–1088. [CrossRef]
19. Wang, K.; Robertazzi, T.G. Scheduling divisible loads with nonlinear communication time. *IEEE Trans. Aerosp. Electron. Syst.* **2015**, *51*, 2479–2485. [CrossRef]
20. Bharadwaj, V.; Ghose, D.; Robertazzi, T. Divisible load theory: A new paradigm for load scheduling in distributed systems. *Clust. Comput.* **2003**, *6*, 7–17. [CrossRef]
21. Suresh, S.; Omkar, S.N.; Mani, V. Parallel implementation of back-propagation algorithm in networks of workstations. *IEEE Trans. Parallel Distrib. Syst.* **2005**, *16*, 24–34, doi:10.1109/TPDS.2005.11. [CrossRef]
22. Othman, S.G.M. Comprehensive Review on Divisible Load Theory: Concepts, Strategies, and Approaches. *Math. Probl. Eng.* **2014**, *2014*, 460354.
23. Beaumont, O.; Casanova, H.; Legrand, A.; Robert, Y.; Yang, Y. Scheduling divisible loads on star and tree networks: Results and open problems. *IEEE Trans. Parallel Distrib. Syst.* **2005**, *16*, 207–218. [CrossRef]
24. Li, X.; Liu, X.; Kang, H. Sensing workload scheduling in sensor networks using divisible load theory. In Proceedings of the IEEE GLOBECOM 2007, Washington, DC, USA, 26–30 November 2007; pp. 785–789.
25. Redondi, A.; Cesana, M.; Tagliasacchi, M.; Filippini, I.; Dán, G.; Fodor, V. Cooperative image analysis in visual sensor networks. *Ad Hoc Netw.* **2015**, *28*, 38–51. [CrossRef]
26. Zhu, T.; Wu, Y.; Yang, G. Scheduling divisible loads in the dynamic heterogeneous grid environment. In Proceedings of the 1st International Conference on Scalable Information Systems, Hong Kong, China, 1 June 2006; Volume 152, p. 8.
27. Hu, M.; Luo, J.; Veeravalli, B. Optimal provisioning for scheduling divisible loads with reserved cloud resources. In Proceedings of the 2012 18th IEEE International Conference on Networks, Singapore, 12–14 December 2012; pp. 204–209.
28. Lin, W.; Liang, C.; Wang, J.Z.; Buyya, R. Bandwidth-aware divisible task scheduling for cloud computing. *Softw. Pract. Exp.* **2014**, *44*, 163–174. [CrossRef]
29. Khan, L.I.L. Implementation and performance evaluation of a scheduling algorithm for divisible load parallel applications in a cloud computing environment. *Softw. Pract. Exp.* **2015**, *45*, 765–781.
30. Abdullah, M.; Othman, M. Cost-based multi-QoS job scheduling using divisible load theory in cloud computing. *Procedia Comput. Sci.* **2013**, *18*, 928–935. [CrossRef]
31. Veeravalli, B.; Viswanadham, N. Suboptimal solutions using integer approximation techniques for scheduling divisible loads on distributed bus networks. *IEEE Trans. Syst. Man Cybern. Part A Syst. Humans* **2000**, *30*, 680–691. [CrossRef]
32. Li, X.; Bharadwaj, V.; Ko, C. Divisible load scheduling on single-level tree networks with buffer constraints. *IEEE Trans. Aerosp. Electron. Syst.* **2000**, *36*, 1298–1308.
33. Veeravalli, B.; Li, X.; Ko, C.C. On the influence of start-up costs in scheduling divisible loads on bus networks. *IEEE Trans. Parallel Distrib. Syst.* **2000**, *11*, 1288–1305. [CrossRef]
34. Li, X.; Veeravalli, B. PPDD: Scheduling multi-site divisible loads in single-level tree networks. *Clust. Comput.* **2010**, *13*, 31–46. [CrossRef]
35. Abdullah, M.; Othman, M. Scheduling Divisible Jobs to Optimize the Computation and Energy Costs. *Int. J. Eng. Sci. Invent.* **2015**, *4*, 27–33.
36. Yang, K.; Ou, S.; Chen, H.H. On effective offloading services for resource-constrained mobile devices running heavier mobile Internet applications. *IEEE Commun. Mag.* **2008**, *46*, 56–63. [CrossRef]