Research Article

International Journal of Distributed Sensor Networks

International Journal of Distributed Sensor Networks
2017, Vol. 13(11)

DOI: 10.1177/1550147717742073
journals.sagepub.com/home/ijdsn
SAGE

A cost- and performance-effective approach for task scheduling based on collaboration between cloud and fog computing

Xuan-Qui Pham, Nguyen Doan Man, Nguyen Dao Tan Tri, Ngo Quang Thai and Eui-Nam Huh

Abstract

The rapid development of Internet of Things applications, along with the limitations of cloud computing due mainly to the far distance between Internet of Thing devices and cloud-based platform, has promoted a newly distributed computing platform based on collaboration between cloud computing and fog computing. Fog computing helps to reduce transmission latency and monetary cost for cloud resources, while cloud computing helps to fulfill the increasing demands of large-scale compute-intensive offloading applications. In this article, we study the tradeoff issue between the makespan and cloud cost when scheduling large-scale applications in such a platform. We propose a scheduling algorithm called Cost-Makespan aware Scheduling heuristic whose major objective is to achieve the balance between the performance of application execution and the mandatory cost for the use of cloud resources. Additionally, an efficient task reassignment strategy based on the critical path of the directed acyclic graph modeling the applications is also proposed to refine the output schedules of the Cost-Makespan aware Scheduling algorithm to satisfy the user-defined deadline constraints or quality of service of the system. We also verify our proposal by extensive simulations, and the experimental results show that our scheduling approach is more cost-effective and achieves better performance compared to others.

Keywords

Internet of Things, cloud computing, fog computing, task scheduling, deadline constraint

Date received: 29 March 2017; accepted: 11 October 2017

Handling Editor: Eduardo Cerqueira

Introduction

Recently, the Internet of Things (IoT) is one of the major revolutions in information and communication technology (ICT). The IoT and its associated technologies, such as machine-to-machine (M2M) technology, extend the Internet connectivity beyond traditional smart devices like smartphones, tablets to a diverse range of devices, and everyday things (e.g. objects, machines, vehicles, buildings) to perform a variety of services and applications (e.g. health care, medicine treatment, traffic control, energy management, vehicular networking). These connected devices are generating an unprecedented amount of data, which needs to be stored, processed, and analyzed for deriving valuable insights as well as properly accessed by end users and/or client applications. Together with it, the quantity and the scale of services and applications are increasing

Department of Computer Science and Engineering, Kyung Hee University, Yongin-si, Korea

Corresponding author:
Eui-Nam Huh, Department of Computer Science and Engineering, Kyung Hee University, Yongin-si 17104, Korea.
Email: johnhuh@khu.ac.kr

rapidly, which may require processing capabilities beyond what could be offered by the most powerful smart devices. Meanwhile, cloud computing, in which dynamically scalable and often virtualized resources are provided as a service over the Internet, may offer a significant complement to IoT. The intrinsic limitations of lightweight smart devices (e.g. battery life, processing power, storage capacity, network resources) can be alleviated by offloading compute-intensive, resource-consuming tasks up to a powerful computing platform in the cloud, leaving only simple jobs to the capacity-limited smart devices.

However, when IoT meets cloud, many challenges arise. According to Information Handling Services (IHS) Markit company, the IoT market will grow from an installed base of 15.4 billion devices in 2015 to 30.7 billion devices in 2020 and 75.4 billion in 2025.[1] With the predicted explosion in the number of connected devices, traditional centralized cloud-based architectures, in which computing and storage resources are concentrated in a few large data centers, will not be able to handle the IoT's data and communication needs anymore. It is mainly caused by the far distance between the cloud and IoT devices. The transmission of huge amount of data or service requests from IoT devices to the cloud over the Internet will not only pose heavy burden to network performance and network bandwidth but also result in unbearable transmission latency and degraded quality of service (QoS) to end users. Moreover, persistent connectivity to the cloud may not always be available for IoT devices or simply too expensive, especially in 3G network.[2] On the other hand, thanks to the advances in hardware and software technology, many network edge devices and even user terminals (e.g. routers, gateways, workstations, PC) are getting more and more powerful in terms of processing, storage, and communication capabilities. These resources are not always utilized by their owners. It results in recent attempts to push the cloud computing capabilities to the network edge.

Fog computing,[3] which was first proposed by Cisco, represents a new paradigm for cloud computing which can transform the network edge into a distributed computing infrastructure for IoT applications. The idea of fog computing is to extend the cloud to be closer to the things that produce and act on IoT data; hence, fog computing is usually referred as "cloud to the ground." Instead of forcing all processing to the cloud, fog computing aims to process part of the applications' workload locally on network edge devices or end-user clients, which are called fog servers or fog nodes. These fog nodes are deployed at the edge of networks (e.g. park, bus terminal, shopping center) to pre-store cloud data and serve as a near-end computing proxies between the front-end IoT devices and the back-end cloud servers. Putting resources at the edge of networks only one or two hops from the data sources allows fog nodes to perform low latency processing while latency-tolerant and large-scale tasks can still be efficiently processed by the cloud. On the other hand, the benefit of scalability and on-demand services in the cloud can help the fog to fulfill the increasing demands of large-scale compute-intensive offloading applications if the resources of fog computing are not sufficient. From this point of view, fog computing is not aimed to replace cloud computing, but to complement it in a new computing paradigm, cloud–fog computing.[4–7]

In practice, many offloading applications are composed of multiple modules performing different tasks. These tasks can be either independent of each other or mutually dependent.[8] In the former case, all tasks can be offloaded simultaneously and processed in parallel. In the latter case, however, the application is made up of tasks that need input from some other tasks and parallel offloading may not be applicable. The latter application is usually modeled as a workflow, which is represented by a directed acyclic graph (DAG), where the set of vertices represents different tasks in the application and the set of edges specifies their precedence constraints. An example of DAG-based application is complex event processing (CEP) which is an emerging big data analytics technique for the IoT. CEP-based systems interpret input data as event streams and accept a user-defined analytics workflow composed of CEP queries (tasks) that need to be performed on these streams in order to derive semantically enriched "complex" event from a series of simpler events.[9,10] With the growing prevalence of cloud–fog computing system, it is inevitable that such DAG-based applications will be naturally distributed and embedded in an environment with numerous connected computing devices with heterogeneous capabilities. As streaming data travel from its point of origin (e.g. sensors) toward cloud platform, it passes through many devices from the network edge to the cloud, each of which is a potential target of computation offloading. The main challenge lies in scheduling application tasks in a pool of processing nodes in cloud and fog environment considering inter-task dependencies to optimize some predefined objective. Previously, many scheduling algorithms were proposed for heterogeneous computing, whose main objective is to minimize the execution time of tasks, without worrying about monetary charges of using computing resources.[11–14] However, with the advent of cloud computing, in which part of the application execution is outsourced to the computing resources of different cloud providers (CPs) and cloud customers (CCs) are charged based on the number of virtual machines (VMs) and hours of usage, some recent efforts have been made to reduce the cost of using cloud service.[15–22] A task schedule, which can minimize the completion time of the workflow but corresponds to a large

amount of monetary cost, is not an optimal solution for CCs.

In this article, we consider task scheduling in a newly distributed computing platform based on collaboration between cloud computing and fog computing for executing large-scale offloading DAG-based applications. Specifically, the computing platform is originated from the infrastructure of fog nodes (e.g. routers, gateways, workstations, IP video cameras) at the premise of CCs and extended by the virtualized hosts (cloud nodes) provided as services in cloud computing. We then propose a novel application scheduling heuristic, Cost-Makespan aware Scheduling (CMaS) algorithm to solve the scheduling problem in this platform. Besides overall efficiency, our proposal also takes into account network condition in the cloud-fog environment and monetary cost charged to CCs. Thus, the major objective of the CMaS algorithm is to achieve a good tradeoff between the application execution time and the cost for the use of cloud resources. Moreover, we present a task reassignment strategy which is responsible for refining the output schedules of the CMaS algorithm to satisfy the user-defined deadline constraints, hence improve the QoS of the system. Our approach is experimentally evaluated and compared with some existing ones. The results prove that our method can guarantee QoS and has better cost-effectiveness than other approaches.

The remainder of the article is organized as follows. In section "Related work," we introduce some related works to the task scheduling problem that have been studied so far. In section "Motivation scenario," we give a motivation scenario, where our proposal model can be practical to cloud–fog collaboration for task scheduling. Then system architecture is described in section "System architecture." In section "Problem model and solution," we detail the problem formulation and present our proposed method. Then we describe some experimental results in section "Implementation and analysis," followed by our conclusions and suggestions for future work in section "Conclusion."

Related work

There have been various studies that attempt to solve task scheduling problem in heterogeneous computing systems, where the sequence of tasks (workflow) is popularly presented by a DAG.[11–14] Because DAG task scheduling is an non-deterministic polynomial-time complete (NP-complete) problem,[11] it is expected to be solved by heuristic algorithms for finding approximate optimal solutions. The heterogeneous earliest finish time (HEFT)[12] algorithm is the most popular and widely used algorithm. The HEFT includes two main phases: a task prioritizing phase for computing the priorities of all tasks based on upward rank value and a processor selection phase for selecting the task with the highest upward rank value at each step and assigning the selected task to the processor which minimizes the task's finish time. Arabnejad and Barbosa[13] introduce predict earliest finish time (PEFT), in which task priority and processor selection are based on an optimistic cost table (OCT) representing for each pair (task, processor) the minimum processing time of the longest path from the current task to the exit node. Meanwhile, Wang et al.[14] propose two innovative policies, entry task duplication selection policy and idle time slots (ITS) insertion-based optimizing policy in order to improve the efficiency of the scheduling algorithm. However, these algorithms for heterogeneous systems intend to minimize makespan and do not look attentively at the problem of dispatching large tasks in a cloud setting or consider monetary cost of using computing resources.

The emerging of cloud computing motivates the extensive studies on the cost and performance-aware application scheduling models.[15–22] For example, Li et al.[16] present a cost-conscious scheduling heuristic, called CCSH, which is an extension of the HEFT algorithm. In this method, the effect of the monetary cost for the use of cloud services is represented by the input cost-conscious factor which is used as a weight to calculate the earliest finish time (EFT) of each task. Panda and Jana[17] propose a method combining cloud min–min scheduling (CMMS)[18] and profit based task scheduling (PBTS) to minimize makespan and cost simultaneously and maximize the average utilization of cloud resources. However, the constraints of customers on QoS or budget are aborted in these approaches. Meanwhile, some works address deadline-constrained task scheduling on the cloud. Den Bossche et al.[20] address task scheduling problem in a hybrid cloud model, which is built from a private cloud and multiple public clouds. The authors introduce a cost-efficient approach to determine the most appropriate infrastructure (public or private cloud) to execute the incoming applications. Decisions are based on the ability to meet the deadline requirement of each application as well as the cost-savings. However, in this work, each workflow is processed on either the public cloud or the private cloud, which is different from our framework where tasks of the workflow can be dispersed on both VMs of CPs and fog nodes at the local system of CCs. Chopra and Singh[21] propose a deadline and cost-based scheduling heuristic in hybrid cloud, in which subdeadline is assigned to each task in workflow application and migrate the tasks to public cloud when deadline is not met at private cloud. However, the tradeoff between the cost and schedule length is not properly addressed in this work.

Table 1. Existing scheduling approaches and ours.

Algorithm	Target system	Application type	Minimum makespan	Minimum cost	Tradeoff	Deadline constraint
Topcuoglu et al.[12]	Heterogeneous	Workflow	Yes	No	No	No
Arabnejad and Barbosa[13]	Heterogeneous	Workflow	Yes	No	No	No
Wang et al.[14]	Heterogeneous	Workflow	Yes	No	No	No
Sinnen and Sousa[30]	Heterogeneous	Workflow	Yes	No	No	No
Li et al.[18]	Cloud	Workflow	Yes	No	No	No
Huerta-Canepa and Lee[2]	Cloud and thin client	Bag-of-tasks	Yes	No	No	No
Zeng et al.[15]	Cloud	Workflow	Yes	Yes	Yes	No
Li et al.[16]	Cloud	Workflow	Yes	Yes	Yes	No
Panda and Jana[17]	Cloud	Workflow	Yes	Yes	Yes	No
Hung and Huh[19]	Cloud and thin–thick client	Workflow	Yes	Yes	Yes	No
Den Bossche et al.[20]	Cloud	Bag-of-tasks	Yes	Yes	No	Yes
Chopra and Singh[21]	Cloud	Workflow	Yes	Yes	No	Yes
Calheiros and Buyya[22]	Cloud	Bag-of-tasks	Yes	Yes	No	Yes
Zeng et al.[28]	Fog	Bag-of-tasks	Yes	No	No	No
Nan et al.[29]	Cloud and fog	Bag-of-tasks	Yes	Yes	Yes	No
Our approach	Cloud and fog	Workflow	Yes	Yes	Yes	Yes

Fog computing, characterized by extending cloud computing to the edge of network, has recently received considerable attention. The seed contribution on fog computing is the one presented by Bonomi et al.,[3] in which the authors provide insights related to the main requirements and characteristics of fog computing and highlight the importance of cloud–fog interplay and the role of fog computing in the context of IoT. Based on the collaboration between cloud and fog computing, Alsaffar et al.[23] present an architecture of IoT service delegation and resource allocation, in which user requests can be efficiently managed and delegated to appropriate cloud or fog based on linearized decision tree which considers tree conditions (service size, completion time, and VM capacity). Souza and colleagues[24,25] address QoS-aware service allocation problem in a combined cloud–fog architecture consisting a dual-layer fog aiming to diminish the cloud access delay in IoT scenarios. Meanwhile, Deng et al.[7] investigate the tradeoff between power consumption and delay in a cloud–fog computing system. Simulation and numerical results from this work demonstrate that the fog can significantly complement the cloud with much-reduced communication latency. It results in a reduced revenue loss associated with the wide area network (WAN) and network bandwidth cost for a CP. Hence, the problem of minimizing the operational cost of cloud data centers with the help of fog devices has received much attention from researchers.[26,27] Besides, there are some early works that try to deal with task scheduling problem regarding fog computing. Zeng et al.[28] target at minimizing the task completion time while providing a better user experience by designing an efficient resource management strategy in a fog computing supported software-defined embedded system. Meanwhile, Nan et al.[29] consider a three-tier cloud of things (CoT) system and design an adaptive decision-making algorithm called unit-slot optimization for distributing the incoming data to the corresponding tiers that can provide cost-effective processing while guaranteeing average response time. However, these works consider bag of independent tasks or unpredictable batch workloads instead of workflow.

To the best of our knowledge, DAG-based workflow scheduling problem does not seem to be studied so far in the context of cloud–fog. Our work serves as a starting point to determine an optimal schedule for large-scale compute-intensive applications based on the collaboration between cloud and fog computing. The main contribution of this work is the development of a heuristic algorithm to achieve a good tradeoff between the application time and the cost for the use of cloud resources under user-defined deadline constraints. For ease of understanding, we present an overview of common scheduling approaches along with ours in Table 1.

Motivation scenario

As fog computing is gaining more popularity, extending cloud computing, and services to the edge of network, we can envision some situations where IoT devices can benefit from using fog computing environment. The following scenario (Figure 1) exemplifies the utilization of a collaborated cloud and fog computing in optimizing task scheduling.

An human resources manager of a company is organizing a meeting for the report on his employees' behavior statistic. In order to get insightful information from the report, the manager uses his Google glass with built-in camera to take pictures from the presentation

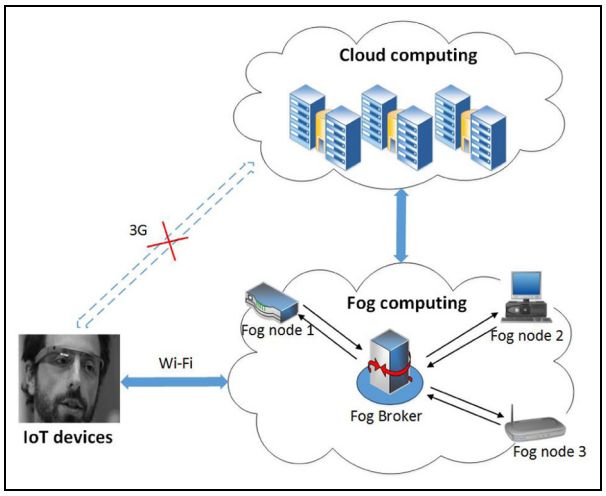

Figure 1. Motivation scenario.

slide and then sends them to the cloud to extract and analyze user activity patterns. The achieved data are used to analyze value information, for example, site traffic, Internet access time, issues of interest, work achievements, on an hourly, daily, weekly, or even yearly basis. These patterns are utilized to develop intrusion detection expert system, performance evaluation program, or other recommendation systems.

Although the glass can be connected to the Internet and use search engine to find information, its bandwidth is too limited for many captured pictures to be uploaded. Fortunately, the glass is preconfigured to be connected with the company's fog nodes in the local network, which own much faster CPU speed, larger RAMs, bigger storage, and especially better Internet connectivity. It is clearly reasonable to let the glass transfer those photos to these fog nodes and rely on them to upload to some cloud services for further processing and looking up for the related information. Since the records of user behavior statistic, which are collected on a daily basis and stored as logs on the cloud, can be up to several hundreds of GBs, it would take long time and be costly to look up some specific data as well as to process them. One of the ways to address this is by splitting the log into smaller chunks, which are then processed in parallel by both a new network made up of the company's fog nodes and cloud VMs. It not only speeds up processing time but also reduces the cost of using cloud resources because it is costly if using solely cloud resources. Eventually, the processed data will be collected at a fog node acting as a broker, where they are combined and forwarded to the user device. With this so-called cloud–fog collaboration, the manager can achieve his desired information in shorter time than the time he could have spent if using only his Google glass.

The above scenario is one of the potential examples, where our proposed model can be applied to utilize the joint work between cloud computing and fog computing. Such collaboration promisingly increases the chance of using resources efficiently. By utilizing multiple fog nodes' relay to enhance the data processing and distribution from cloud networks, lightweight smart device can overcome its computing and network limitation. In this article, we aim at designing a network architecture, based on the collaboration between cloud and fog computing, attempting to solve the following issue: minimizing the makespan of large-scale compute-intensive applications to satisfy QoS requirement in a highly distributed platform of cloud–fog computing while considering the network contention and cloud cost.

System architecture

In our framework, we assume that fog computing system located at the premise of CCs has the role as a service provider (fog provider) to provide the services of application processing to a specific number of IoT device users. Our system architecture has three layers in a hierarchy network as represented in Figure 2.

The bottommost layer consists of user IoT devices, which can be smartphones, tablets, wearable devices, thin-client, smart home appliances, wireless sensor nodes, and so on. They send requests to the upper layers for application execution.

The middle layer represents fog computing environment. The primary components of this layer are intelligent fog devices (e.g. routers, gateways, switches, access points) that have the capability of computing, networking, and storage. They are called fog nodes, which are deployed in the vicinity of end users to receive and process part of a workload of users' requests with the local short-distance high-rate connection. Also, they are connected to the cloud so as to benefit from a massive pool of redundant resources of the cloud on demand.

The uppermost layer represents cloud computing environment, which hosts a number of heterogeneous cloud nodes or VMs of different cloud service providers. The cloud nodes provide outsourced resources to execute the workload dispatched from the fog layer.

In the fog layer, there is a fog device acting as a resource management and task scheduler component, which is called *fog broker*. The broker (1) receives all requests of users; (2) manages available resources on the cloud and fog nodes (e.g. processing capacity, network bandwidth) as well as processing and communication costs together with results of data query returned from nodes; and (3) accordingly creates the most appropriate schedule for an input workflow to decide which part of the workflow will run on which resources. The major components of *fog broker* are described in detail as follows.

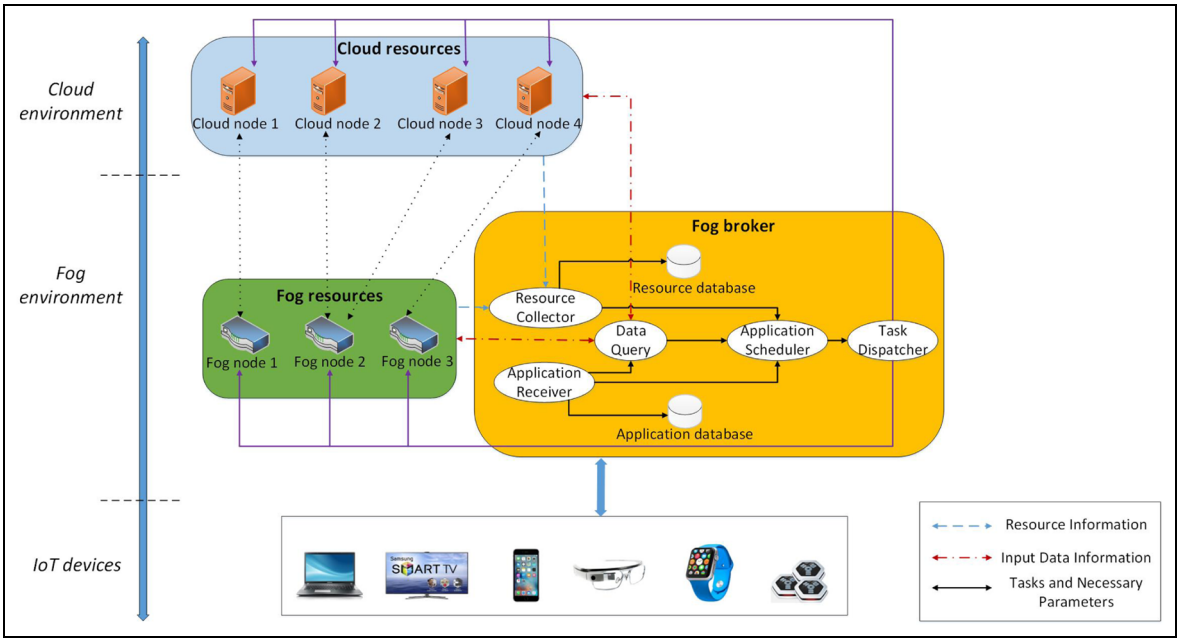

Figure 2. System architecture.

Application receiver is the component that is responsible for providing a user interface for application submission. Each application reaching to *fog broker* is specified by all of the relevant parameters and information such as the number of tasks, the workload of each task, the amount of input data, which are stored into the *application database*.

Next *data query* requires *application receiver* for all information about the input data of the latest application and then creates the database queries to all data storages under the management of *fog broker* to find the necessary amount of data distributed in the computing infrastructure. From the returned querying results, the locations of all input data are disclosed and available for the following application scheduling phase.

Resource collector is responsible for collecting and managing information about the execution rates and data transfer rates of all processing nodes or the billing policy of CPs and storing it in *resource database*. Data in the *resource database* are updated frequently along with the resource supplementation or removal on the computing infrastructure as well as the changes in the billing policy of each CP. This guarantees that the final application schedule produced by *fog broker* is appropriate with the latest updates on the computing resources of CCs and therefore achieves the higher accuracy.

Based on the profiles about processing capacity and network bandwidth of all computing nodes as well as monetary costs for using cloud resources together with results of data query returned from nodes, the *application scheduler* analyzes the application, finds the best schedule and then transfers the output schedule to *task dispatcher*, which in turn dispatches the tasks of each application and sufficient parameters and information to the appropriate computing nodes (cloud or fog).

Problem model and solution

Task scheduling on a target system that has network topology is defined as the problem of allocating the tasks of an application to a set of processors with various processing capabilities so as to minimize the makespan of the schedule. Thus, the input of task scheduling involves a task graph and a processor graph, and the output is a schedule representing the assignment of a processor to each task as shown in Figure 3. In this section, we explain how to formulate the problem and then propose a task scheduling method to solve the mentioned problem.

Problem model

Task graph. A task graph is represented by a DAG, $G = (V, E)$, where the set of vertices $V = \{v_1, v_2, ..., v_n\}$ denotes the set of parallel tasks, and each edge $e_{ij} = (v_i, v_j) \in E$ represents the communication or precedence constraint between two tasks v_i and v_j. Each task $v_i \in V$ has positive workload w_i representing the amount of computing works (e.g. the number of instructions) which have to be processed at the computing resources. And each edge $e_{ij} \in E$ has nonnegative weight c_{ij} which represents the amount of communication data transferred from task v_i and used as input

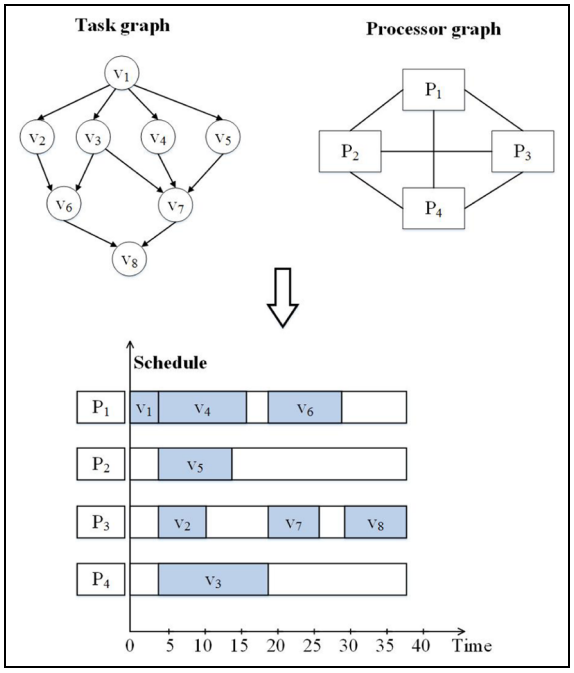

Figure 3. Illustration of task scheduling problem.

data for task v_j. The set of all direct predecessors and successors of v_i are denoted as $pred(v_i)$ and $succ(v_i)$, respectively. A task v_i without any predecessors, that is, $pred(v_i) = 0$, is called an entry task v_{entry}. And a task without any successors, that is, $succ(v_i) = 0$, is called an exit task v_{exit}.

We assume that a task cannot begin its execution until all of its inputs have been gathered sufficiently. The input of a task is not only from the preceding tasks but also from other data sources (i.e. data storages) on both cloud and fog infrastructure and only determined after completing the execution of the preceding tasks. Each task appears only once in the schedule.

Processor graph. A processor graph $PG = (N, D)$ denotes the topology of a cloud–fog network, where the set of vertices $N = \{P1, P2, ..., P_n\}$ denotes the set of processors, each of which is a cloud or fog node, and an edge $d_{ij} \in D$ denotes a link between processor P_i and P_j. Let N_{cloud} and N_{fog} denote the set of cloud nodes and the set of fog nodes, respectively. Hence, $N = N_{cloud} \cup N_{fog}$. Each processor P_i has processing rate p_i and bandwidth bw_i on the link connecting it to other processors.

We assume that due to physical constraints on networking devices, the processing rate of fog nodes is often smaller than that of cloud nodes, which are high-performance VMs in cloud data center. In addition, fog nodes are located in the local system, due to the higher stability of local area network (LAN) than WAN, the bandwidth between processors in fog environment is higher than that of cloud environment.

Proposed method

Given a task graph $G = \{V, E\}$ and a processor graph with network topology $PG = (N, D)$, our method chooses the most appropriate schedule to execute the tasks. Our proposed task scheduling mechanism is composed of three major phases. In the *task prioritizing* phase, all tasks of a specific application are set the corresponding priority levels. Next, the *node selection* phase assigns each task to an appropriate processing node (cloud or fog) to achieve the optimal value of a utility function, which determines the tradeoff between schedule length and cloud cost. Finally, for QoS requirement, a strategy is proposed for *task reassignment* phase, in which the output schedule from the second phase is refined to satisfy the user-defined deadline. It is done by replicating the best node selection for the tasks on the critical path to reduce the summation of the completion time of the application.

Task prioritizing phase. In this phase, tasks are ordered by their scheduling priorities that are based on upward ranking.[12] Basically, the upward rank of a task v_i is the length of the critical path from v_i to the exit task, including the computation time of task v_i. Let $pri(v_i)$ be the priority value of task v_i and be recursively defined by

$$pri(v_i) = \begin{cases} \overline{w(v_i)} + \max\limits_{v_j \in succ(v_i)} \left[\overline{c(e_{ij})} + pri(v_j)\right] & \text{if } v_i \neq v_{exit} \\ \overline{w(v_i)} & \text{if } v_i \equiv v_{exit} \end{cases} \quad (1)$$

where $\overline{w(v_i)}$ is the average computation time of task v_i and $\overline{c(e_{ij})}$ is the average communication time between two tasks v_i and v_j, correspondingly

$$\overline{w(v_i)} = \frac{w_i}{\left(\sum\limits_{P_n \in N} p_n\right)/n} \quad (2)$$

$$\overline{c(e_{ij})} = \frac{c_{ij}}{\left(\sum\limits_{P_n \in N} bw_n\right)/n} \quad (3)$$

with n is the number of computing nodes in the cloud–fog environment.

Finally, we sort all tasks by nonincreasing order of priority level, which provides a topological order of all tasks and thus preserves the precedence constraints between two specific tasks.

Node selection phase. A task v_i only begins its execution after all preceding tasks of v_i are completed. Let $DTT(v_i)$ be the finish time of the last preceding task of v_i. It is also the time when all input data of v_i are ready

to be transferred to the selected node for executing the task v_i. Therefore, it is called data transfer time of task v_i and defined by

$$DTT(v_i) = \max_{v_j \in pred(v_i), P_m \in N} \left[t_f(v_j, P_m) \right] \quad (4)$$

where $t_f(v_j, P_m)$ is the finish time of task v_j on node P_m. For the entry task, $DTT(v_{entry}) = 0$.

Suppose task v_i is assigned to node P_n. Let $c(e_i^{mn})$ be the communication time for transferring data from node P_m to node P_n to execute task v_i, then $c(e_i^{mn})$ is defined as follows

$$c(e_i^{mn}) = \begin{cases} \left(d_i^m + \sum_{v_j \in exec(P_m)}^{v_j \in pred(v_i)} c_{ji} \right) * \left(\frac{1}{bw_m} + \frac{1}{bw_n} \right) & \text{if } m \neq n \\ 0 & \text{if } m = n \end{cases} \quad (5)$$

where d_i^m is the amount of data already stored at processor P_m for executing task v_i and $exec(P_m)$ is the set of tasks executed at node P_m.

The ready time of task v_i on processor P_n, denoted by $t_{dr}(v_i, P_n)$, is the time when all necessary input data of v_i that were sent from data storages on either cloud nodes or fog nodes have arrived at the target processing node P_n. Thus, $t_{dr}(v_i, P_n)$ is defined by

$$t_{dr}(v_i, P_n) = DTT(v_i) + \max_{P_m \in N} \left(c(e_i^{mn}) \right) \quad (6)$$

Let $EST(v_i, P_n)$ and $EFT(v_i, P_n)$ be the earliest start time and the earliest finish time of task v_i on node P_n. Let $w(v_i, P_n)$ denote the execution time of task v_i on node P_n. The $EST(v_i, P_n)$ is defined by searching the earliest ITS on node P_n that is capable of holding the execution time of task v_i. The following scheduling condition formulates that:

Condition 1. Let $[t_A, t_B], t_A, t_B \in [0, \infty]$ be an idle time interval on processor $P_n \in N$ in which no task is executed. A free task $v_i \in V$ can be scheduled on P_n within $[t_A, t_B]$ if

$$max\{t_A, t_{dr}(v_i, P_n)\} + w(v_i, P_n) \leq t_B \quad (7)$$

For the found interval $[t_A, t_B]$, the $EST(v_i, P_n)$ and $EST(v_i, P_n)$ are computed as follows

$$EST(v_i, P_n) = max\{t_A, t_{dr}(v_i, P_n)\} \quad (8)$$

$$EFT(v_i, P_n) = EST(v_i, P_n) + w(v_i, P_n) \quad (9)$$

Besides, the algorithm also considers the monetary cost that the CCs are charged for the use of cloud resources such as computing processors, network resources, storage, and memory. The computing infrastructure is composed of fog nodes at the premise of CCs and extended by the cloud nodes provided as services in cloud computing. Hence, let $cost(v_i, P_n)$ be the monetary cost for executing task v_i on node P_n. If P_n is a cloud node, the monetary cost includes processing cost, storage cost, memory cost of task v_i on node P_n, and communication cost for the amount of outgoing data from other cloud nodes to the target node P_n to process task v_i. In contrast, if P_n is a fog node, the CCs only need to pay for transferring the outgoing data from cloud nodes to the target fog node in the local system. Therefore, the $cost(v_i, P_n)$ is defined as follows

$$cost(v_i, P_n) = \begin{cases} c_{proc}^{(v_i, P_n)} + c_{str}^{(v_i, P_n)} + c_{mem}^{(v_i, P_n)} + \sum_{P_m \in N_{cloud}} c_{comm}^{(v_i, P_m)} & \text{if } P_n \in N_{cloud} \\ \sum_{P_m \in N_{cloud}} c_{comm}^{(v_i, P_m)} & \text{if } P_n \in N_{fog} \end{cases} \quad (10)$$

In equation (10), each cost is calculated as follows.

Processing cost is expressed as

$$c_{proc}^{(v_i, P_n)} = c_1 * w(v_i, P_n) \quad (11)$$

where c_1 is the processing cost per time unit of workflow execution on node P_n.

Let c_2 be the storage cost per data unit and let st_i be the storage size of task v_i on node P_n. Then the storage cost of task v_i is as follows

$$c_{str}^{(v_i, P_n)} = c_2 * st_i \quad (12)$$

We next compute the cost of using the memory of node P_n for task v_i as follows

$$c_{mem}^{(v_i, P_n)} = c_3 * s_{mem} \quad (13)$$

where s_{mem} be the size of the memory used and c_3 is the memory cost per data unit.

Let c_4 be the amount of money per data unit for transferring outgoing data from cloud node P_m, and then the communication cost is calculated as follows

$$c_{comm}^{(v_i, P_m)} = c_4 * \left(d_i^m + \sum_{v_j \in exec(P_m)}^{v_j \in pred(v_i)} c_{ji} \right) \quad (14)$$

From this cost, we can define a utility function which computes the tradeoff between the cost and EFT as follows

$$U(v_i, P_n) = \frac{\min_{P_k \in N} [cost(v_i, P_k)]}{cost(v_i, P_n)} * \frac{\min_{P_k \in N} [EFT(v_i, P_k)]}{EFT(v_i, P_n)} \quad (15)$$

Then, the task v_i is assigned to the node P_n, which provides the maximal value of the tradeoff $U(v_i, P_n)$. After task v_i is scheduled on node P_n, the actual finish time of task v_i, $AFT(v_i)$, is equal to the $EFT(v_i, P_n)$

value. After all tasks in a graph are scheduled, the schedule length or makespan will be the actual finish time of the exit task.

The CMaS algorithm is presented in Algorithm 1.

Deadline-based task reassignment phase. For the application scheduling problems in real world, the user-defined deadline is important information to improve the QoS performance of the system. In other words, when a user submits an application request to the application scheduler, which is represented by *fog broker* in our work, the output schedule must guarantee that the application execution does not finish later than a predefined deadline. To limit the deadline violation, the *fog broker* in the fog computing layer needs to control the output schedules after the *node selection* phase so that the application execution can meet the user-defined deadline with the possibly lowest amount of the monetary cost for cloud resources. Thus, we present a strategy in Algorithm 2, namely deadline-based task reassignment algorithm, to improve the completion time of application execution as deadline constraints. The major idea of this strategy is that the makespan of a schedule is significantly influenced by the total execution time of tasks on the critical path of an application graph.[31] The critical path refers to the longest execution path between the entry task and the exit task of the application and the tasks on this path is called critical tasks.

In our algorithm, the improvement on the makespan of a schedule can be found by reassigning the critical tasks to better processing nodes which can reduce the completion time of each critical task (lines 11–23). For this reason, we distribute evenly the distance between the overall deadline and the makespan of the current schedule (i.e. the AFT value of the exit task) to each critical task (line 10), which has not been reassigned yet, to find the subdeadline, that is, the expected AFT, for that task (line 12). The processing nodes which guarantee that the EFT value of a critical task is not greater than task's subdeadline are considered as the candidate nodes to which this task should be assigned (line 14). Based on the list of the candidate nodes, the node selection for task execution still relies on the maximum value of the utility function $U(v_i, P_n)$ mentioned in the previous section (line 16). However, if the candidate list is empty, that is, no processing node can complete the task earlier than the subdeadline, the task is scheduled to the node which provides the minimal EFT value (line 18). As all the critical tasks were rescheduled, the rest of the application, the non-critical tasks, may also be reassigned to the new nodes corresponding to the updates on the critical tasks (line 24). After this step, since the AFT of a critical task may decrease, this task may be no longer on the critical path of the application. Hence, we need to find the new critical path (line 9) and the task reassignment algorithm is deployed again on the new path.

In practical terms, it is very difficult or impossible for any scheduling algorithm to produce the application schedules which can meet all tight deadlines. Therefore, the major objective of task reassignment phase is to raise schedules which can either meet the deadline constraints or minimize the difference between the overall makespan and the deadline in case of very tight deadline. Specifically, the task reassignment phase will be carried on until the deadline is satisfied ($makespan(current) \leq deadline$) or there is no schedule providing better makespan than the latest schedule that has just been found in the same phase ($makespan(current) \geq old\_makespan$) (line 26).

Implementation and analysis

This section presents experiments that analyze many aspects of our approach via numerical simulations. First, to justify the efficiency of the proposed CMaS algorithm, we have also designed several other task scheduling algorithms to compare their performances in terms of schedule length, monetary cost, and trade-off level. The compared methods include the Greedy for Cost (GfC), which is to minimize workflow execution cost by assigning each task to the most cost-saving processing node; the well-known HEFT,[12] which attempts to schedule interdependent tasks at minimum execution time on a heterogeneous environment; and the CCSH,[16] which deliberates both the EFT of tasks and the cloud cost paid by CCs. Then we evaluate the effectiveness of the proposed task reassignment strategy on controlling the application execution so as to meet QoS requirement as well as cost-savings. These scheduling algorithms are applied into the *application scheduler* component of *fog broker*, respectively.

Experimental settings

The experimental setup for our model's evaluation is shown in Table 2. The simulations are developed in Java with JDK-8u112-x64 and Netbeans-8.0.1 using CloudSim,[32] which is a framework for modeling and simulation of cloud computing infrastructures and services. In our simulation, we use MIPS (million instructions per second) to represent the processing capacity of processors.

Our first experiment covers a random task graph $G = (V, E)$ with the increase of sizes from 20 to 100 and a heterogeneous processor graph $PG = (N, D)$ which is the combination of 25 cloud nodes with the different configurations and 15 fog nodes at the fog computing system. As assumed, the processing rate of fog nodes is set smaller than that of cloud nodes and the bandwidth in the fog environment is higher than that of cloud

Algorithm 1. Cost-Makespan aware Scheduling (CMaS) algorithm.

Input: Task graph $G(V, E)$ and processor graph $PG = (N, D)$ ($N = N_{cloud} \cup N_{fog}$)
Output: A task schedule
Compute the priority level $pri(v_i)$ of each task $v_i \in V$ by traversing graph upward, starting from v_{exit}.
Sort all tasks of V into list L by nonincreasing order of priority levels.
for all $v_i \in L$ **do**
  for all $P_n \in N$ **do**
    Compute $EST(v_i, P_n)$, $EFT(v_i, P_n)$ and $cost(v_i, P_n)$
    Compute the utility function $U(v_i, P_n)$
  end for
  Assign task v_i to the processor P_n that maximizes $U(v_i, P_n)$ of task v_i
end for

Algorithm 2. Deadline-based task reassignment algorithm.

Input: Task graph $G = (V,E)$, processor graph $PG = (N,D)$, and an application schedule S for $G(V,E)$
Output: A new task schedule
$current \leftarrow S$
repeat
  $old\_makespan \leftarrow current.makespan$
  for all $v_i \in G$ **do**
    $v_i.isReAssigned \leftarrow false$
  end for
  $CP = getCriticalPath(G = (V, E))$ {CP is an array of critical tasks}
  $distance \leftarrow \dfrac{old\_makespan - deadline}{length\_of\_CP}$
  for all $v_i \in CP$ **do**
    $subdeadline(v_i) \leftarrow AFT(v_i) - distance$
    if $subdeadline(v_i) \geq 0$ **then**
      $candidates \leftarrow getListCandidate(G = (V, E), PG = (N, D))$ {*candidates* is the set of processing nodes P_n satisfying $EFT(v_i, P_n) \leq subdeadline(v_i)$}
      if *candidates* is not empty **then**
        $best\_node(v_i) \leftarrow \arg\max_{P_n \in candidates} [U(v_i, P_n))]$
      else
        $best\_node(v_i) \leftarrow \arg\min_{P_n \in N} [EFT(v_i, P_n)]$
      end if
      $AFT(v_i) \leftarrow EFT(v_i, best\_node(v_i))$
      $v_i.isReAssigned \leftarrow true$
    end if
  end for
  Scheduling tasks v_i ($v_i \notin CP$ & $v_i.isReAssigned=false$) and recomputing the parameters for the critical tasks on G(V,E)
  $current \leftarrow$ new schedule of G(V,E)
until $current.makespan \leq deadline$ or $current.makespan \geq old\_makespan$

Table 2. Simulation setup.

System	Intel® Core™ i5-4670 CPU, 3.40 GHz
Memory	16 GB
Simulator	Cloudsim 3.0.3
Operating system (OS)	Windows 8.1 Professional
Topology model	Fully connected

Table 3. Characteristics of the cloud and fog environment.

Parameter	Fog environment	Cloud environment
Number of processors	15	25
Processing rate	[10, 500] MIPS	[250, 1500] MIPS
Bandwidth	1024 Mbps	10, 100, 512, 1024 Mbps

MIPS: million instructions per second.

environment. The characteristics of the cloud and fog environment are shown in Table 3. And the cost of using cloud resources is specified in Table 4. The I/O data of a task have a size from 100 to 500 MB.

Note that in our experiment, we generate 10 random DAGs with different edges and node weights for each task graph size. Then we run each method to schedule these task graphs and calculate the average schedule

Table 4. Cost of using cloud resources.

Parameter	Value
Processing cost per time unit	[0.1, 0.5]
Communication cost per data unit	[0.3, 0.7]
Unit cost of memory used	[0.01, 0.1]
Unit cost of storage used	[0.05, 0.2]

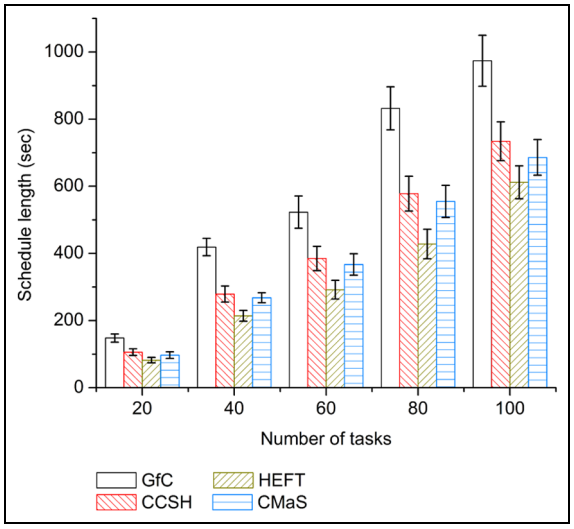

Figure 4. Schedule length comparison.

length and the average monetary cost for the changing number of tasks. The error bars in the following figures indicate 95% confidence interval.

Evaluation of CMaS algorithm

Figure 4 shows that in terms of schedule length, GfC algorithm gets the worst case, HEFT algorithm obtains the best result while CCSH and CMaS are in the middle. Specifically, our method is 5.53% better than CCSH. And compared with GfC, our method even achieves a far better performance, about 32.64%.

However, regarding the monetary cost for cloud resources (as illustrated in Figure 5), it has been observed that although HEFT provides the best performance, it has the highest cost while the opposite is true for GfC algorithm. In the meantime, our solution conduces to the benefits of balance between schedule length and cloud cost. Compared with HEFT algorithm, our method can save 30.17% of cloud cost while performance reduction is not more than 23%. Also, compared with CCSH algorithm, our method can save 5.06% of cloud cost, which means that our method has a slight economic advantage together with its effectiveness.

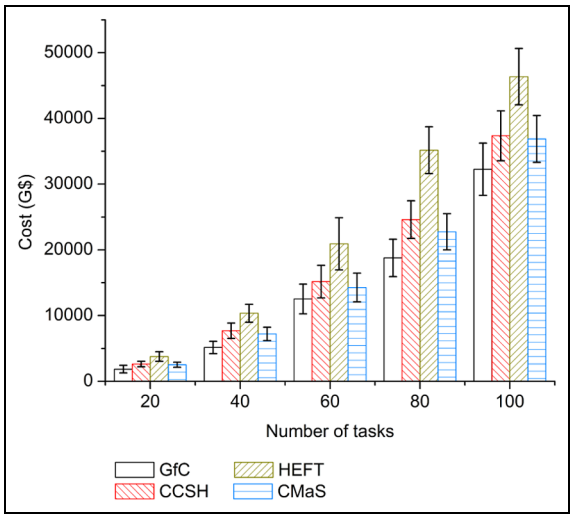

Figure 5. Cost comparison.

The reason behind these results is as follows. The main objective of HEFT algorithm is to minimize the makespan of application execution without taking into account the cost paid for cloud resources. To reduce the makespan, HEFT assigns more tasks to cloud nodes with better processing capacity; hence, it causes higher monetary cost. On the contrary, GfC algorithm aims to achieve cost-savings by assigning tasks to the cheapest processors with low processing capacity, and hence, it takes longer makespan. Meanwhile, in CCSH algorithm, the effect of the monetary cost for the use of Cloud resources is represented by the input cost-conscious factor δ which is used as a weight to calculate the EFT of each task called cost-EFT (CEFT). This method assigns each task to the processor that minimizes CEFT. However, the factor δ takes into account only the processing cost. The costs of other resources (e.g. network bandwidth) are not considered. Compared with CEFT in CCSH, the utility function defined in our algorithm provides a more holistic way to calculate the tradeoff between the makespan and monetary cost of all cloud resources.

Moreover, in order to demonstrate more clearly that our algorithm can achieve better tradeoff between the makespan and the cost of task execution than other methods, we base on a comparison criteria called *cost-makespan tradeoff (CMT)*[33] to determine the best algorithm at each task graph size as follows

$$CMT(a_i) = \frac{\min_{a_k \in AL}[cost(a_k)]}{cost(a_i)} * \frac{\min_{a_l \in AL}[makespan(a_l)]}{makespan(a_i)} \quad (16)$$

where $AL = \{a_1, a_2, \ldots, a_n\}$ is the list of all scheduling algorithms, which we compute the CMT value of each algorithm $a_i \in AL$. The maximum CMT value of an algorithm is 1, which is reachable if the schedule length

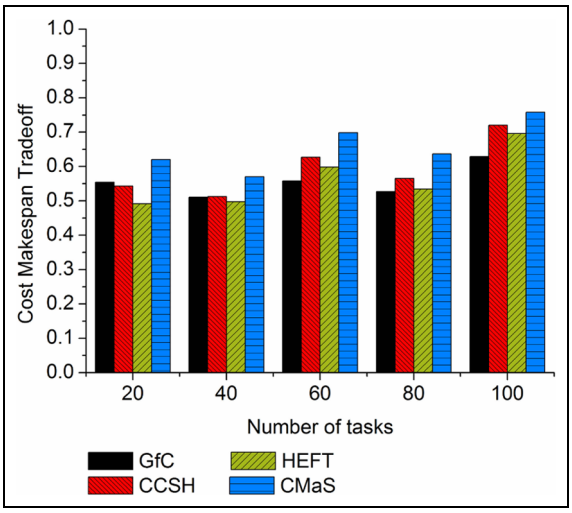

Figure 6. Cost-makespan tradeoff (CMT) comparison.

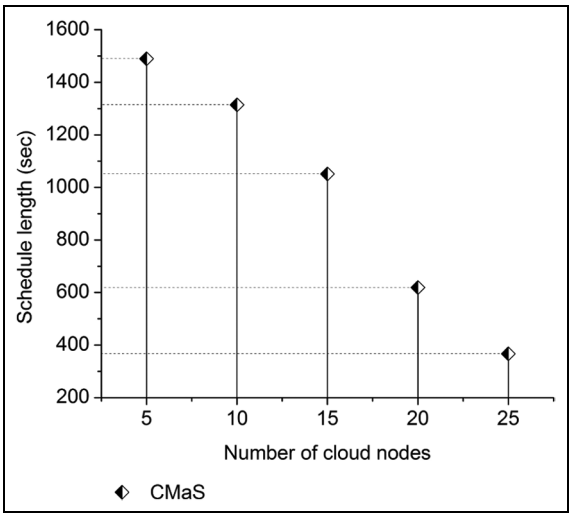

Figure 7. Impact of number of cloud nodes on schedule length.

and the monetary cost of that algorithm are the best (i.e. smallest) compared with the others. Otherwise, the higher value of CMT an algorithm can achieve, the better tradeoff level on monetary cost and schedule length it can provide.

Figure 6 shows the superiority of CMaS algorithm at all task graph sizes. We can see that our method is stable and achieve the highest CMT values compared with the others. Specifically, the average CMT value of our method on all task graph sizes is better than GfC by 16.33%, CCSH by 10.21%, and HEFT by 14.75%. Among other scheduling methods, GfC algorithm gives better result than HEFT and CCSH at small task graph size. However, as the number of tasks increases, the performance of HEFT and CCSH is getting better. CCSH algorithm has a slightly better tradeoff level than HEFT since it takes into account both makespan and cloud cost.

In next experiment, we evaluate the impact of increasing number of cloud nodes on the schedule length and the cloud cost only in CMaS algorithm with a fixed number of tasks. Similar to the previous experiment, the number of processing nodes (i.e. cloud and fog) in the computing infrastructure is fixed at 40. However, we change the number of cloud nodes from 5 to 25 to generate different simulated models of the computing infrastructure. The experimental results shown in Figures 7 and 8 indicate that the present of more cloud nodes, which provide better processing capacity as well as larger amount of available input data for task execution than fog nodes, results in better system performance but higher monetary cost for cloud resources. It is highly noticeable that the schedule length reduces 41.10%, but the cloud cost raises 59.89% as the number of cloud nodes increases from 15 to 20. In general, the results of this experiment can be considered as the

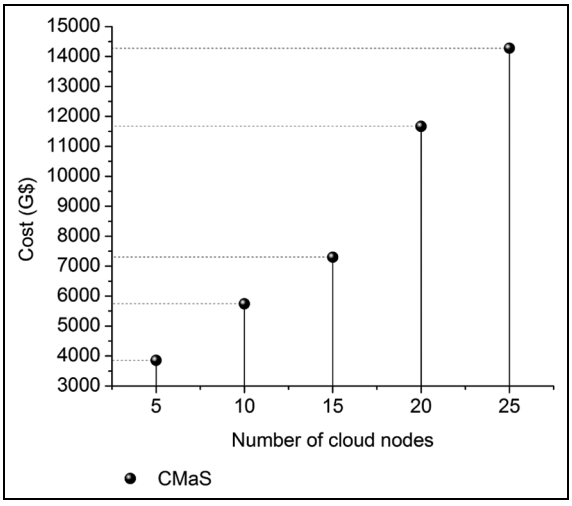

Figure 8. Impact of number of cloud nodes on monetary cost.

basis to determine the appropriate number of cloud nodes and fog nodes in the computing infrastructure so as to satisfy deadline constraint or budget constraint of the application scheduling problem.

Evaluation of deadline-based task reassignment

In this experiment, we evaluate the effectiveness of the deadline-based task reassignment algorithm on controlling the application execution to satisfy the user required QoS. We will specify the deadline constraints on two real-world applications, Gaussian Elimination (GE) program[34] and Epigenomics in biology.[35] The structures of these real application graphs are known. The GE program is characterized by the parameter m that denotes the matrix size. The total number of tasks

in a GE task graph is equal to $(m^2 + m - 2)/2$. The detailed characterization of an Epigenomics application is provided by the workflow generator.[36] In our experiment, the matrix size of the GE application varies from 10 to 30 by the increasing steps of 10. Meanwhile, three sizes of the Epigenomics application that we consider are small with 24 tasks, medium with 46 tasks, and large with 100 tasks. We reuse the previous experimental setup which is composed of 25 cloud nodes and 15 fog nodes in the cloud–fog infrastructure. Each task graph of the two applications will be scheduled by only the CMaS algorithm or the CMaS with the task reassignment phase, which is called deadline-based CMaS algorithm.

In order to evaluate the effect of the task reassignment on deadline constraints, we use HEFT algorithm which can provide the best makespan compared to others as baseline algorithm. Specifically, the application schedule length provided by the HEFT algorithm, denoted by T_{HEFT}, is considered as the optimal schedule length that our system can support. Deadline D is defined by $D = k*T_{HEFT}, k \geq 1$. Obviously, the application deadline specified by users should be not less than the optimal value T_{HEFT}. We categorize the user-defined deadlines into two types: tight deadline and loose deadline. The former is with $k = 1.1$, while the latter is with $k = 1.3$. In other words, the schedule length is required not 10% greater and 30% greater than the optimal value at tight and loose deadline, respectively. Thus, we calculate the ratio of the makespan of the proposed algorithms to T_{HEFT}, which is called OverHEFT Makespan Ratio (OMR), to determine whether the algorithms can meet tight (i.e. $OMR \leq 1.1$) or loose deadline (i.e. $OMR \leq 1.3$)

$$OMR = \frac{makespan_i}{T_{HEFT}} \tag{17}$$

where i is the corresponding algorithm, that is, CMaS or deadline-based CMaS.

In addition, we also evaluate the impact of the task reassignment as user-defined deadline on the monetary cost for the use of cloud resources. As mentioned earlier, compared to the HEFT algorithm, our CMaS algorithm can gain an amount of cost-savings. The OverHEFT Cost Ratio (OCR) is computed by

$$OCR = \frac{cost_i}{cost_{HEFT}} \tag{18}$$

where i is the corresponding algorithm, that is, CMaS or deadline-based CMaS.

Figure 9 clearly shows the advantage of the deadline-based task reassignment algorithm. It can be seen that the CMaS algorithm can satisfy loose deadline in most cases; however, it cannot guarantee the application execution finishes before tight deadline. Meanwhile, tight deadline can be satisfied after the task reassignment phase refines the output schedule from the CMaS scheduling algorithm to a better performance in terms of makespan. Specifically, the deadline-based CMaS can guarantee the finish of the application execution before the predefined tight deadline when it is deployed on the small, medium, and large size of the mentioned applications, except for the GE program with the matrix size of 30 or 464 tasks. Note that the adaptability with tight deadline has a decreasing trend along with the increase on the scale of the applications. In particular, for the GE program, when the size of the matrix increases, the OMR value of deadline-based CMaS gradually rises. To explain, the effect of the critical path on the schedule length of an application may decrease corresponding to the increasing number of tasks of an application. However, in general, by comparing the application makespan of CMaS before and after the reassignment phase, we observe that the deployment of the task reassignment phase is actually efficient to make the completion time of application execution meet or significantly closer to the required deadline. Therefore, the

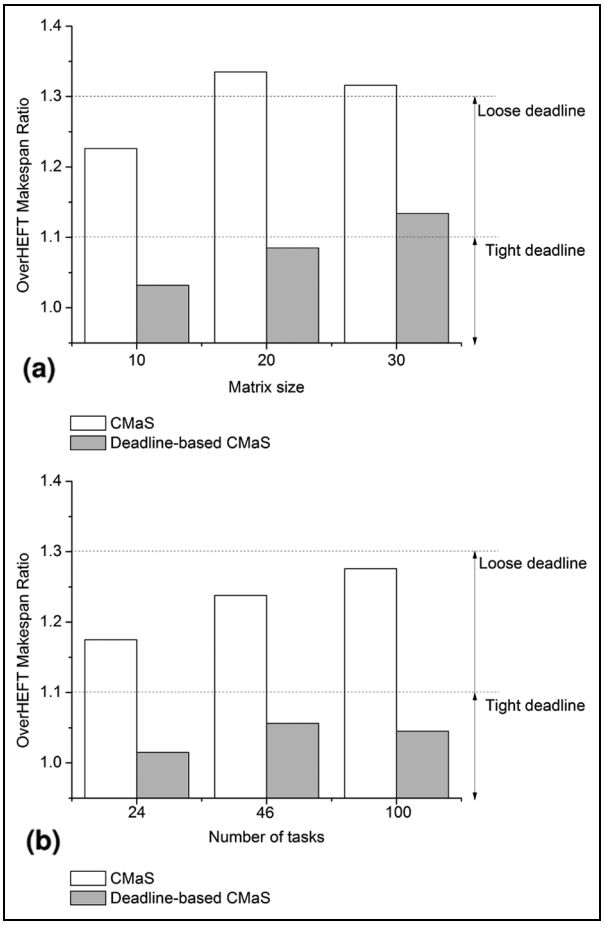

Figure 9. Impact of deadline-based task reassignment on makespan: (a) OverHEFT Makespan Ratio for Gaussian Elimination and (b) OverHEFT Makespan Ratio for Epigenomics.

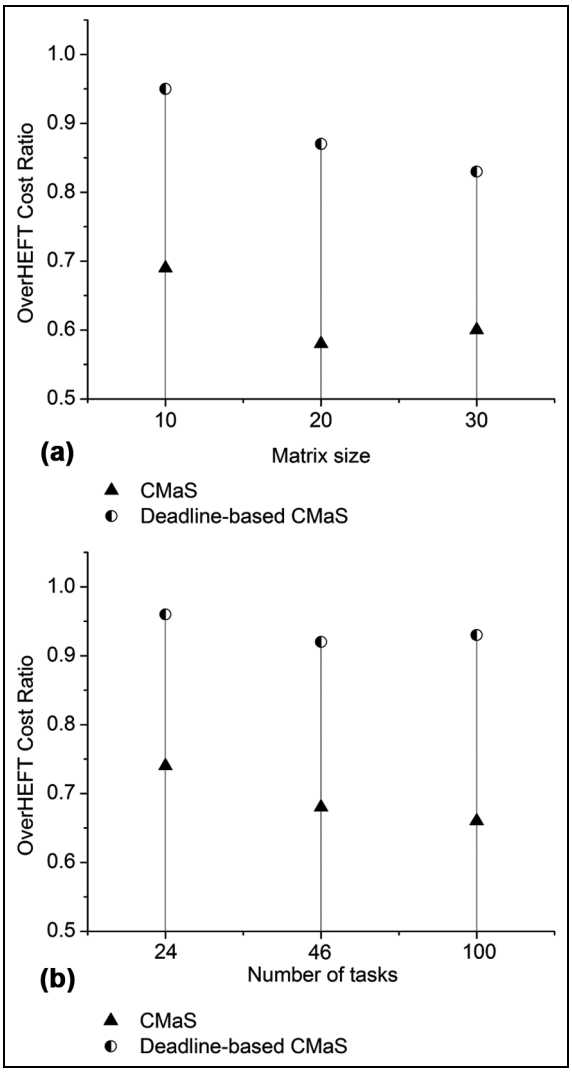

Figure 10. Impact of deadline-based task reassignment on cloud cost: (a) OverHEFT Cost Ratio for Gaussian Elimination and (b) OverHEFT Cost Ratio for Epigenomics.

probability of deadline violation is reduced to a reasonable level.

Figure 10 shows another benefit of the deadline-based CMaS. It can be observed that our CMaS algorithm demands much smaller monetary cost of cloud resources than HEFT algorithm. After the task reassignment phase, the cloud cost becomes higher as more cloud resources are utilized to produce better schedule. However, the cloud cost of the deadline-based CMaS is still handled not to exceed that of the HEFT algorithm (i.e. $OCR \leq 1$). In other words, the deadline-based CMaS algorithm can save an amount of monetary cloud cost while providing schedule that can meet the user-defined deadline or QoS requirement. For example, in the Epigenomics application with 100 tasks, the deadline-based CMaS algorithm can achieve nearly 7% cost reduction compared with the HEFT algorithm.

Conclusion

The IoT devices along with the demands for services and applications are increasing rapidly on both the quantity and the scale. The combined cloud–fog architecture is a promising model that if well exploited can provide efficient data processing various applications or services, especially those which are compute-intensive. This article addresses task scheduling problem in the effort to provide smart devices with a smooth access to the cloud as well as to achieve a better service quality based on the collaboration between cloud and fog computing. For the sake of reaping the most benefit from such a platform, one must allocate computing tasks strategically at each processing node of cloud or fog layer. We propose the CMaS algorithm considering the tradeoff between performance and cost-savings to build the application schedule, which not only guarantees application execution within deadline constraints but also reduces the mandatory cost for the use of cloud resources. A lot of simulations have been conducted and their results prove that our work can provide a better cost-time-effective solution than other existing methods.

In future work, we intend to deploy our proposal into real-world systems. We see CEP for the IoT as a potential application.[10] CEP is an event processing that deals with the task of processing multiple events with the goal of identifying the meaningful events within the event cloud. We consider an IoT deployment scenario with a user-defined analytics workflow composed of CEP queries (tasks) that need to be performed on multiple event streams generated at the edge (e.g. from IoT devices), and several fog nodes at the premise of CCs and public cloud nodes available to perform the queries. With the planned implementation, we can thoroughly observe the real-world operation, performance and work out any shortcomings to improve our proposal.

On the other hand, green computing is now becoming very important. With the huge volume and ever-increasing service requests, the power consumption of both cloud and fog computing platform is soaring. Therefore, we can extend the proposed scheduling for large-scale applications by also considering energy-efficiency while guaranteeing QoS is still a challenge.

Declaration of conflicting interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this

article: This research was supported by the MSIP (Ministry of Science, ICT and Future Planning), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2017-2013-0-00717) supervised by the IITP (Institute for Information & Communications Technology Promotion).

References

1. Lucero S. IoT platforms: enabling the internet of things, https://cdn.ihs.com/www/pdf/enabling-IOT.pdf (2016, accessed 13 February 2017).
2. Huerta-Canepa G and Lee D. A virtual cloud computing provider for mobile devices. In: *Proceedings of the 1st ACM workshop on mobile cloud computing & services: social networks and beyond (MCS '10)*, pp.1–6. New York: ACM, http://doi.acm.org/10.1145/1810931.1810937
3. Bonomi F, Milito R, Zhu J, et al. 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)*, pp.13–16. New York: ACM, http://doi.acm.org/10.1145/ 2342509.2342513
4. Huang CY and Xu K. Reliable realtime streaming in vehicular cloud-fog computing networks. In: *2016 IEEE/CIC international conference on communications in China (ICCC)*, Chengdu, China, 27–29 July 2016, pp.1–6. New York: IEEE.
5. Masip -Bruin X, Marn-Tordera E, Alonso A, et al. Fog-to-cloud Computing (F2C): the key technology enabler for dependable e-health services deployment. In: *2016 Mediterranean ad hoc networking workshop (Med-Hoc-Net)*, Vilanova i la Geltrú, 20–21 June 2015, pp.1–5. New York: IEEE.
6. Lin Y and Shen H. Leveraging fog to extend cloud gaming for thin-client MMOG with high quality of experience. In: *2015 IEEE 35th international conference on distributed computing systems*, Columbus, OH, 29 June–2 July 2015, pp.734–735. New York: IEEE.
7. Deng R, Lu R, Lai C, et al. Optimal workload allocation in fog-cloud computing toward balanced delay and power consumption. *IEEE Internet Things* 2016; 3(6): 1171–1181.
8. Mach P and Becvar Z. Mobile edge computing: a survey on architecture and computation offloading. *IEEE Commun Surv Tut* 2017; 19(3): 1628–1656.
9. Higashino WA, Capretz MA and Bittencourt LF. CEPSim: modelling and simulation of complex event processing systems in cloud environments. *Future Gener Comp Sy* 2016; 65: 122–139, http://www.sciencedirect.com/science/article/pii/S0167739X15003362
10. Choi JH, Park J, Park HD, et al. DART: fast and efficient distributed stream processing framework for internet of things. *ETRI J* 2017; 39(2): 202–212, http://dx.doi.org/10.4218/etrij.17. 2816.0109
11. Ullman J. NP-complete scheduling problems. *J Comput Syst Sci* 1975; 10(3): 384–393, http://www.sciencedirect.com/science/article/pii/S0022000075800080
12. Topcuoglu H, Hariri S and Wu MY. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE T Parall Distr* 2002; 13(3): 260–274, http://dx.doi.org/10.1109/ 71.993206
13. Arabnejad H and Barbosa JG. List scheduling algorithm for heterogeneous systems by an optimistic cost table. *IEEE T Parall Distr* 2014; 25(3): 682–694.
14. Wang G, Wang Y, Liu H, et al. HSIP: a novel task scheduling algorithm for heterogeneous computing. *Sci Programming* 2016; 2016: 3676149, http://dx.doi.org/10.1155/2016/3676149
15. Zeng L, Veeravalli B and Li X. ScaleStar: budget conscious scheduling precedence-constrained many-task workflow applications in cloud. In: *2012 IEEE 26th international conference on advanced information networking and applications*, Fukuoka, Japan, 26–29 March 2012, pp.534–541. New York: IEEE.
16. Li J, Su S, Cheng X, et al. Cost-conscious scheduling for large graph processing in the cloud. In: *2011 IEEE international conference on high performance computing and communications*, Banff, AB, Canada, 2–4 September 2011, pp.808–813. New York: IEEE.
17. Panda SK and Jana PK. A multi-objective task scheduling algorithm for heterogeneous multi-cloud environment. In: *2015 international conference on electronic design, computer networks automated verification (EDCAV)*, Shillong, India, 29–30 January 2015, pp.82–87. New York: IEEE.
18. Li J, Qiu M, Ming Z, et al. Online optimization for scheduling preemptable tasks on IaaS cloud systems. *J Parallel Distr Com* 2012; 72(5): 666–677, http://www.sciencedirect.com/science/article/pii/S0743731512000366
19. Hung PP and Huh EN. An adaptive procedure for task scheduling optimization in mobile cloud computing. *Math Probl Eng* 2015; 2015: 969027.
20. Den Bossche RV, Vanmechelen K and Broeckhove J. Cost-efficient scheduling heuristics for deadline constrained workloads on hybrid clouds. In: *2011 IEEE third international conference on cloud computing technology and science*, Athens, 29 November–1 December, pp.320–327. New York: IEEE.
21. Chopra N and Singh S. Deadline and cost based workflow scheduling in hybrid cloud. In: *2013 international conference on advances in computing, communications and informatics (ICACCI)*, Mysore, India, 22–25 August 2013, pp.840–846. New York: IEEE.
22. Calheiros RN and Buyya R. *Cost-effective provisioning and scheduling of deadline-constrained applications in hybrid clouds*. Berlin, Heidelberg: Springer, pp.171–184, http://dx.doi.org/10.1007/978-3-642-35063-4_13
23. Alsaffar AA, Phuoc HP, Hong C, et al. An architecture of iot service delegation and resource allocation based on collaboration between fog and cloud computing. *Mob Inf Syst* 2016; 2016: 6123234.
24. Souza VBC, Ramrez W, Masip-Bruin X, et al. Handling service allocation in combined fog-cloud scenarios. In: *2016 IEEE international conference on communications (ICC)*, Kuala Lumpur, Malaysia, 23–27 May 2016, pp.1–5. New York: IEEE.
25. Souza VB, Masip-Bruin X, Marin-Tordera E, et al. Towards distributed service allocation in fog-to-cloud (F2C) scenarios. In: *2016 IEEE global communications conference (GLOBECOM)*, Washington, DC, 4–8 December 2016, pp.1–6. New York: IEEE.

26. Yu L, Jiang T, Sun M, et al. Cost-aware resource allocation for fog-cloud computing systems. CoRR 2017, abs/1701.07154, http://arxiv.org/abs/ 1701.07154
27. Yu L, Jiang T and Zou Y. Fog-assisted operational cost reduction for cloud data centers. *IEEE Access* 2017; 5: 13578–13586.
28. Zeng D, Gu L, Guo S, et al. Joint optimization of task scheduling and image placement in fog computing supported software-defined embedded system. *IEEE T Comput* 2016; 65(12): 3702–3712.
29. Nan Y, Li W, Bao W, et al. Cost-effective processing for delay-sensitive applications in cloud of things systems. In: *2016 IEEE 15th international symposium on network computing and applications (NCA)*, Cambridge, MA, 31 October–2 November 2016, pp.162–169. New York: IEEE.
30. Sinnen O and Sousa LA. Communication contention in task scheduling. *IEEE T Parall Distr* 2005; 16(6): 503–515.
31. Gotoda S, Ito M and Shibata N. Task scheduling algorithm for multicore processor system for minimizing recovery time in case of single node fault. In: *2012 12th IEEE/ACM international symposium on cluster, cloud and grid computing (CCGrid 2012)*, Ottawa, ON, Canada, 13–16 May 2012, pp.260–267. New York: IEEE.
32. CloudSim: a framework for modeling and simulation of cloud computing infrastructures and services,http://www.cloudbus.org/cloudsim/ (accessed 10 January 2017).
33. Pham XQ and Huh EN. Towards task scheduling in a cloud-fog computing system. In: *2016 18th Asia-Pacific network operations and management symposium (APNOMS)*, Kanazawa, Japan, 5–7 October 2016, pp.1–4. New York: IEEE.
34. Cosnard M, Marrakchi M, Robert Y, et al. Parallel Gaussian elimination on an MIMD computer. *Parallel Comput* 1988; 6(3): 275–296, http://www.sciencedirect.com/science/article/pii/0167819188900701
35. Bharathi S, Chervenak A, Deelman E, et al. Characterization of scientific workflows. In: *2008 third workshop on workflows in support of large-scale science*, Austin, TX, 17 November 2008, pp.1–10. New York: IEEE.
36. Workflow Generator, https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator (accessed 10 January 2017).