Latency-Aware Task Assignment and Scheduling in Collaborative Cloud Robotic Systems

Shenghui Li*‡, Zhiheng Zheng*‡, Wuhui Chen*‡, Zibin Zheng*‡, Junbo Wang†
*School of Data and Computer Science, Sun Yat-sen University, Guangzhou, China
†Graduate School of Computer Science and Engineering, University of Aizu, Japan
‡National Engineering Research Center of Digital Life, Sun Yat-sen University, Guangzhou, China
lishh53@mail2.sysu.edu.cn, Lotu817@gmail.com, {chenwuh, zhzibin}@mail.sysu.edu.cn, j-wang@u-aizu.ac.jp

***Abstract*—Traditional robotic systems are often incapable of handling complex tasks due to hardware constraints, such as computing ability, storage space, and battery capacity. Cloud robotic systems, characterized by allowing multi-robot systems to access the powerful cloud infrastructures, is a promising solution to fulfill complex tasks, such as disaster management, real-time object recognition, 3D Simultaneous Localization And Mapping (SLAM). However, the destabilizing factors of network could lead to high latency of data transmission in cloud robotic systems, which have made great challenges to the fields that have high real-time requirements. What's more, the existence of heterogeneity of robots further complicates cloud robotics cooperation. In order to minimize the average response time in latency-aware scenarios, we jointly investigate task assignment and scheduling in Collaborative Cloud Robotic Systems (CCRS). We first formulate the problem into a Mixed-Integer Non-Linear Programming (MINLP) and then linearize it into an Integer Linear Programming (ILP) using discrete time structure. To meet the extensibility requirement, we further propose a partitioning-based algorithm to deal with large-scale task graphs. The results show that our two approaches outperform the existing genetic algorithm and greedy algorithm.**

***Index Terms*—cloud robotic system, task assignment, scheduling, latency.**

I. Introduction

There is no doubt that robotic systems have brought significant contributions in different aspects of human life over the past dozen years. Despite the development of robotics, it is infeasible to develop a universal robot with limitless abilities due to various constraints, such as hardware capabilities, power consumption, payload and kinematic constraints. Thus, inspired by the concept of cloud computing, James Kuffner at Google coined the term "Cloud Robotics" [1] to describe a new approach to robotics that takes advantage of the Internet as a resource for massively parallel computation and real-time sharing of vast data resources [2]. By combining the respective advantages of cloud computing and robotic clusters, robotic agents can work cooperatively, not only by sharing their processing resources with each other but also with the supporting of remote cloud servers, making them more intelligent, efficient and knowledgeable.

Although cloud robotic systems have obtained some positive results [2], [3], they are far from widespread use in the latency-aware scenarios, because existing approaches simply offload computational-intensive tasks to the cloud, without consideration of high communication cost between local robot and the remote cloud. On the other hand, traditional Multi-Robot Task Assignment (MRTA) approaches [4], [5] for manufacturing and other applications cannot be applied to cloud robotic applications because the unique set of characteristics of Collaborative Cloud Robotic Systems (CCRS) severely stress the collaboration of robotic agents and the cloud.

First, the overhead of big data delivery [6] is a critical issue that influences the latency. Many applications in CCRS require continuous processing of high data rate sensors such as digital cameras and scanners [7], [8], [9]. Since cloud computing can speed up computation-intensive robotic applications. Many complex tasks that traditional robotic systems are incapable of handling with, will be potentially fulfilled by CCRS. Most of those tasks such as interactive perception applications (for example, object and pose recognition), collective robot learning (for example, collective reinforcement learning) need robots to continuously collect and process large volume environmental data in real-time to maintain accuracy. This hence can lead to long data transmission time. In this case, it would not be beneficial for the local robots to offload computation to the cloud.

Second, the network bandwidth of the cloud robotic network is the bottleneck for data-intensive applications. Since local robots use remote cloud resources to perform tasks, they rely on a persistent connection to a network infrastructure [10]. However, robots usually access the cloud through Internet-based robot-to-cloud (R2C) network, which may cause high latency due to complex environments, network congestion, and limited bandwidth [11]. If too many robots offload the computation to the cloud via Internet access simultaneously, they may generate severe interference to each other, which would reduce the data transmission rate. On the other hand, a team of robots can communicate via robot-to-robot (R2R) networks such as local area networks (LANs), or mobile ad hoc networks (MANETs) [11], which provide faster data transfer rate than R2C networks. Hence, robots can collaboratively work by parallel computing. Therefore, it is significant to investigate the tradeoff between offloading and local processing to avoid possible communication bottleneck and improve response time.

2159-6190/18/$31.00 ©2018 IEEE
DOI 10.1109/CLOUD.2018.00016

Third, heterogeneity always exists in CCRS because it may not be cost-effective to provide every team member with the most expensive equipment. Through the robots' diverse set of capabilities, CCRS can solve some tasks more effectively than their homogeneous counterpart for the same cost. While taking into consideration the heterogeneity of different robots and cloud servers, latency-diversity exits among these compute nodes, which require an appropriate task assignment strategy to bring all those heterogeneous cloud-enable robots into effective teamwork.

Without the guarantee of Quality-of-Service (QoS) for task execution, local robots can hardly take full advantage of the powerful cloud infrastructures. To conquer above weaknesses, we study the latency minimization problem for heterogeneous CCRS via joint optimization of task assignment and scheduling for a given task graph in this paper. The main contributions of this paper are as follows:

- We investigate the task assignment and scheduling for CCRS to cooperatively execute complex applications efficiently. In particular, we study a latency-aware problem in CCRS. To our best knowledge, the latency-aware problem with the objective of latency minimization in cloud robotic cooperation has been little studied in the literature.
- We formulate the latency minimization problem in a form of Mixed-Integer Nonlinear Programming (MINLP) with joint consideration of task assignment and scheduling.
- To deal with the high computational complexity of solving MINLP, we linearize it as an Integer Linear Programming (ILP) problem using discrete time structure. We further propose a scalable algorithm based on task graph partitioning to deal with large-scale task graphs.

The remainder of this paper is organized as follows. Section II reviews some related work in cloud robotics. Section III introduces our system model and section IV gives the problem analysis. In Section V, we formulate the latency optimization problem as an MINLP problem and transform it into an ILP. Next, we propose a scalable algorithm in Section VI. Section VII demonstrates the experimental results and Section VIII concludes our research and points out our future work.

II. RELATED WORK

A. Multi-Robot Task Assignment

Multi-Robot Task Assignment (MRTA) has been widely addressed in the literature from different aspects. MRTA can be classified as centralized assignment and distributed assignment. Centralized techniques comprise of a central planning unit which has the whole environment. M. Gombolay et al. [4] have proposed an algorithm that handles tightly intercoupled temporal and spatial constraints to MRTA problem. Distributed techniques, on the other hand, dispersed tasks amongst the agents without a centralized agent. For example, ALLIANCE is a fully distributed architecture that allows teams of robots perform missions, which has been proposed by Lynne et al. [12]. The existing MRTA techniques can be distributed into four major approaches: Behaviour Based (BB) [12] , Market Based (MB) [13], Combinatorial Optimization Based (COB) [4], and Evolutionary Algorithm Based (EAB) [14]. BB approaches decide if the robot should consider a particular job or not based on a reluctance or willingness like feature; MB approaches are distributed approaches which work on auction-based mechanism; COB approaches model MRTA problems to combinatorial optimization problems and use popular techniques to solve; EAB approaches are population-based optimization schemes that comprise of a population of solutions optimized using evolutionary operators.

B. Cloud Robotics

A number of interesting projects in the domain of cloud-based robotics have appeared over the last years. Researchers in Singapore have proposed DAvinCi [15], a software framework to enable teams of heterogeneous robots to handle environments. W. J. Beksi et al. [16] have presented a cloud-based system architecture for robotic path planning. [17] has introduced a Cloud-based Object Recognition Engine (CORE) to address object recognition using a robotic network with access to remote computing facilities. The European Union project RoboEarth [18], led by Hollands Eindhoven University, aims to build a giant database in the cloud where robots could share information about objects, environments, and tasks. Gouveia et al. proposed two distributed architectures for the SLAM problem, and analyzed their efficiency, localization precision, and map accuracy [19]. Japanese researchers have built the Rapyuta [20], which is a Platform as a service (PaaS) framework for moving computation off from robots and into the Cloud. What's more, the open source software package ROS (Robot Operating System) has been developed with the purpose of helping software developers create robot applications by Willow Garages team. Agostinho et al. [21] proposed a cloud computing environment for networked robotics applications, in which VMs are assigned different roles in the cloud environment, and a layered workflow management system was used for scheduling purposes.

However, these works simply offload computational-intensive tasks to the cloud, without consideration of high communication cost between a robot and the remote cloud. In order to overcome this limitation, W. Chen et al [22] have proposed a QoS-aware RSW allocation algorithm for networked cloud robotic systems with joint optimization of latency, energy consumption, and operating cost. Z. Cheng et al. [23] have proposed a three-layer architecture for wearable device and investigated an offloading strategy to guaranteed delay requirements. L. Wang et al. have presented a framework for near real-time Multisensor data retrieval [24] and a hierarchical auction-based mechanism for real-time resource allocation [11] in typical cloud robotic scenarios.

Different from existing works, we take into consideration both communication cost and computation cost in heterogeneous CCRS and focus on the task completion latency minimization to guarantee the quality of cloud robotics cooperation, which has been little studied in the literature.

TABLE I: Notations

Constants	
S_{ij}	The size of intermediate data from task i to task$j$
R_{ik}	The running time of task i on node k
B_{kl}	The transmission rate from node k to node l
$\mathcal{D}$	The set of time slots $\mathcal{D} = \{1, 2, ..., t\}$
$pred_i$	The predecessors of task i
$succ_i$	The sucessors of task i
Variables	
x_{ik}	A binary variable indicating whether task i is assigned to node k
x_{ik}^l	A binary variable indicating whether task i is scheduled to run on node k starting at time slot l
y_{ij}	A binary variable indicating whether task i is scheduled to run after j
t_i^s	The start time of task i
t_i^e	The finish time of task i

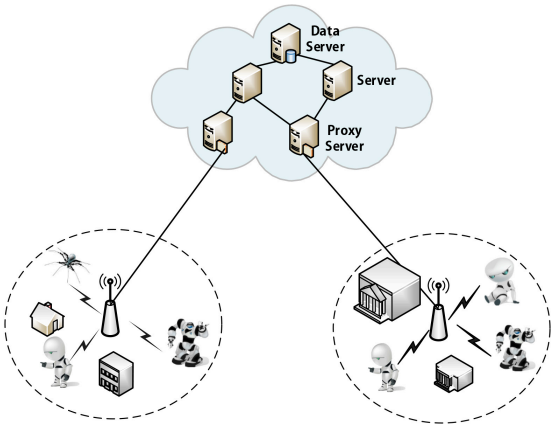


Fig. 1: An overview of CCRS.

III. SYSTEM MODEL

In this section, we introduce the system model. For the sake of the reader's convenience, often-used notations are summarized in Table 1.

A. Collaborative Cloud Robotic System Model

We consider an architecture of CCRS as shown in Fig. 1. We use a graph $G_n = (V_n, E_n)$ to denote the CCRS, which consists of a set of nodes including robots and the cloud, denoted as V_n, and a set of communication links among nodes in V_n, denoted as E_n. In addition, each link (k, l) is associated with a transmission rate B_{kl}. According to the previous discussion, we have $B_{RC} \leq B_{RR}$ in general, i.e., the communication between two nodes in R2R is faster than that in R2C.

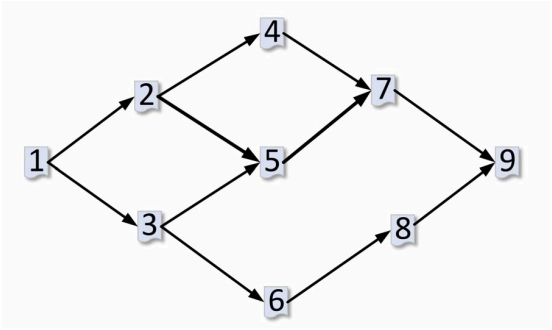


Fig. 2: An example task graph consisting of 9 tasks.

B. Application Model

To structure an application for offloading and parallel computing, we use a Directed Acyclic Graph (DAG) to represent the application model as $G_t = (V_t, E_t)$, $|V_t| = m$, where vertices represent a number of tasks, and edges denote inter-task data and execution dependency. Each task $i \in V_t$ is associated with a weight C_i that represents its computational complexity. The weight of each directed link $(i, j) \in E_t$, representing the corresponding intermediate data size, can be defined as S_{ij}. For each directed edge $(i, j) \in E_t$, We call task i the predecessor of task j, and task j the successor of task i. we use $pred_i$ and $succ_i$ to present the set of predecessors and the set of successors of task i, respectively. A task can execute only when all its predecessors have finished and all its required input data have arrived. Fig.2 shows a task graph containing nine tasks.

Note that there are two special tasks in G_t, source task and destination task, for a task graph with multiple sources or destination tasks, we could convert it to this model by inserting a virtual source node or destination node.

IV. PROBLEM ANALYSIS AND STATEMENT

Due to constraints of power consumption, computing ability, storage space in standalone robots, offloading part of task to a remote server or distributing a complex task to a group of robots potentially reduces response time. For example, a gesture recognition application includes face detection and motion extraction. Computational-intensive tasks in the later such as features extracting can be offloaded to the cloud; other tasks such as sensing, scaler, copy can be accomplished by robots in parallel. Moreover, once an optimal schedule is found, it may be reused when the same application is rerun. To guarantee the required QoS of CCRS, we target on latency minimization, where latency is the time taken to execute all stages of a task graph.

We make two important assumptions:

- Non-overlapping: at most one task runs on a node at any time. Due to the resource constraints on nodes (especially robots), it is beneficial to complete a task as soon as possible so that other tasks that depend on it can start sooner when compared with running multiple tasks on the same node at the same time.

- Non-preemption: a running task will not be interrupted since preemption leads to system overheads and it is difficult to make the scheduling policies.

With the two assumptions above, once a task is scheduled to run on a node starting at time t, it will exclusively occupy the resources in that node until, and constantly run until its completion.

To achieve the target of latency minimization, we need to deal with the challenges of both task assignment and task scheduling. Task assignment determines on which node task should be executed. Compared with the cloud, robots usually have limited processing capability, but they are closer to data source (e.g. camera data, sensor data) and are usually in the same local network, leading to small latency for data transfer. In addition to the tradeoff between processing speed and transmission delay, the existence of heterogeneity of robots further complicates task assignment.

Task scheduling determines the execution sequence of tasks assigned to the same node. For example, if tasks 3, 5, and 6 in Fig. 2 are assigned to the same compute node, 8 and 9 are assigned to another node. we have two possible execution sequences, i.e., {3, 5, 6, 8, 9} and {3, 6, 5, 8, 9} with different performance. When tasks are executed according to {3, 6, 5, 8, 9}, task 8 can quickly get its input data after task 6, and run in parallel with task 5. On the other hand, if tasks are executed according to {3, 5, 6, 8, 9}, they will run in sequence, ultimately resulting in poor performance. Based on the discussion above, we need an effective task assignment and scheduling strategy for CCRS to reduce the latency.

V. PROBLEM FORMULATION

A. Assignment Constraints

First, we define a binary variable x_{ik} for task allocation as follows:

$$x_{ik} = \begin{cases} 1, & \text{if task } i \text{ is assigned to node } k, \\ 0, & \text{otherwise.} \end{cases}$$

Each task should be executed by one and only one node in the CCRS, which can be specified as:

$$\sum_{k \in V_n} x_{ik} = 1, \forall i \in V_t. \tag{1}$$

Note that a node $k \in V_n$ can be allocated to execute task $i \in V_t$ only if there are available resources to satisfy demand. For example, some tasks like sensing cannot be assigned to the cloud, i.e., they should be executed only on robots that equal with corresponding sensors. We use Z_{ik} to indicate whether task i can be executed by node k, In general, this can be expressed as

$$x_{ik} \leq Z_{ik}, \forall i \in V_t, k \in V_n. \tag{2}$$

B. Scheduling Constraints

Any task $i \in V_t$ cannot start before it has received data from all its predecessors in the task graph. Therefore, we have

$$t_i^s \geq \max_{j \in pred_i} \{t_j^e + \sum_{(k,l) \in E_n} \frac{x_{ik} x_{jl} S_{ji}}{B_{kl}}\}, \forall i \in V_t, \tag{3}$$

where t_i^e is the finish time of task i, B_{kl} is the bandwidth between nodes l and k. Note that if task i and one of its predecessor $j \in pred_i$ are assigned the same node, the communication delay is negligible because data delivery within a compute node can be implemented via shared memory without communication delay, which means the second term in the right hand of constraint (3) will be zero. Otherwise, we need to consider the communication delay $\frac{S_{ji}}{B_{kl}}$.

According to the non-overlapping and non-preemption assumptions, once a task is scheduled to run on a node k, it will continue to run on that node and use the node exelusicely until completion. For ease of presentation, we introduce another binary variable as follows:

$$y_{ij} = \begin{cases} 1, & \text{if task } i \in V_t \text{ is scheduled to run after } j \in V_t, \\ 0, & \text{otherwise.} \end{cases}$$

Therefore, the non-overlapping assumption can be formulated as:

$$t_i^s \geq \max_{j \in V_t} \{ \sum_{k \in V_n} x_{ik} x_{jk} y_{ij} t_j^e \}, \forall i \in V_t, \tag{4}$$

where t_j^e denotes the finish time of task j, since there is at most one task runing on a computing node at any time. Any task $i \in V_t$ should start to execute when the assigned node is available, which means no other tasks are currently executing on it. Constraint (4) presents that for task $j \in V_t$ whose priority is higher than i, if j and i are allocated to the same node k, i should wait until the completion of j.

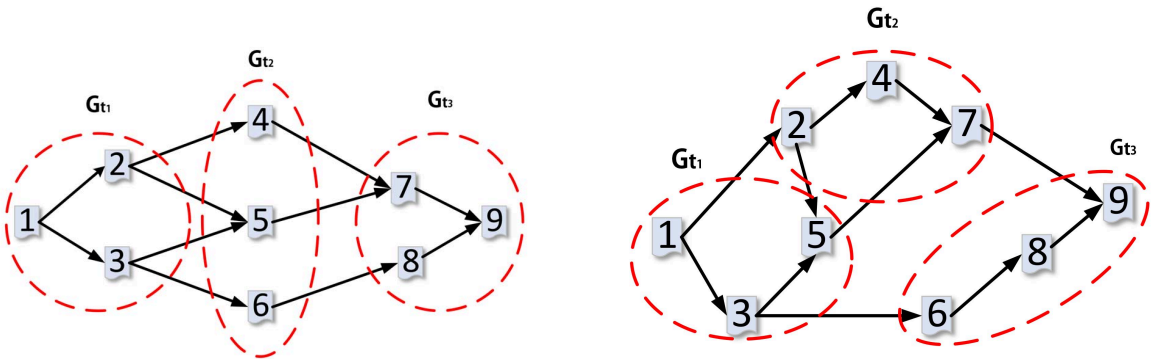
Furthermore, once i starts, it will not be interrupted. Therefor, the realationship between start time t_i^s and finish time t_i^e can be expressed as:

$$t_i^e = t_i^s + \sum_{k \in V_n} x_{ik} R_{ik}, \forall i \in V_t. \tag{5}$$

Without loss of generality, we use R_{ik} to indicate the time-consuming for node k to execute task i, whose precise characterization is challenging in heterogeneous cloud robotic systems. In general, R_{ik} depends on the CPU cycles, workload, and clock frequency. Further more, the CPU cycles needed by a computation task depends on the type of the task itself. For instance, if R_{ik} is inversely proportional to C_k, we have $R_{ik} = \Phi_i / C_k$, where Φ_i and C_k indicate the amount of computation required of task i and the processing rate of node k, respectively.

C. MINLP

Given the system model described above, our goal is to achieve the minimization latency of task graph execution by choosing the best task assignment and scheduling scheme.



(a) legal task graph partitioning.

(b) illegal task graph partitioning.

Fig. 3: Example of task graph partitioning.

By summarizing all constraints discussed above, we can formulate this optimization protblem as a MINLP problem.

$$\begin{aligned} &\textbf{MINLP}: \\ &min: \quad t_m^e, \\ &s.t.: \quad (1),\ (2),\ (3),\ (4) \text{ and } (5), \\ &\qquad x_{ik}, y_{ij} \in \{0,1\}, \end{aligned}$$

where m is the number of tasks, t_m^e represents the finish time of the end task. For the example in Fig. 2, t_m^e means the finish time of task 9.

D. Linearization

It is very difficult to solve the MINLP problem directly due to its high computational complexity. We observe that constraints (3) and (4) are nonlinear due to the products of two and four variables, respectively. To linearize constraint (4), we put forward a method of discretization of continuous time structure, where the furtherest time horizon T for task execution is divided into a sequence of time slots $\mathcal{D} = \{1, 2, ..., t\}$ of identical duration, then we define a new variable x_{jk}^l as follows:

$$x_{ik}^l = \begin{cases} 1, & \text{if task } i \text{ is run on node } k \text{ starting at slot } l, \\ 0, & \text{otherwise.} \end{cases}$$

Each task should start only once, which can be specified as:

$$\sum_{k \in V_n} \sum_{l \in \mathcal{D}} x_{ik}^l = 1, \forall i \in V_t, \tag{6}$$

consider a fixed task i, it will start exactly once on exactly one compute node, constraint (6) is the same as saying that if task i is assigned to node k, it will start exactly once duaring T.

The relationship between x_{ik}^l and x_{ik} can be expressed as:

$$\sum_{l \in \mathcal{D}} x_{ik}^l = x_{ik}, \forall i \in V_t. \tag{7}$$

The start time of task i can be calculated as following formula:

$$t_i^s = \sum_{k \in V_n} \sum_{l \in \mathcal{D}} x_{ik}^l T_l, \forall i \in V_t. \tag{8}$$

The non-overlapping and non-preemption constraint can be expressed as:

$$\sum_{i \in V_t} \sum_{v=max(1, l-R_{ik}+1)}^{l} x_{ik}^v \leq 1, \forall i \in V_t, \tag{9}$$

we will explain why (9) is the right specification for non-overlapping under the assumption of non-preemption. The execution of task i occupies time slot l on node k if and only if it is assigned to node k with the start time in the set $\{max(1, l - R_{ik} + 1), ..., l\}$. The latter holds if and only if $\sum_{v=max(1,l-R_{ik}+1)}^{l} x_{ik}^v = 1$. Thus, (9) means that there is at most one task occupying time v on compute node k.

To linearize constraint (3), we define a new variable f_{ijkl} as:

$$f_{ijkl} = x_{ik} x_{jl}, \quad \forall (i,j) \in E_t, \forall (k,l) \in E_n, \tag{10}$$

which can be equivalently replaced by the following linear constraints:

$$0 \leq f_{ijkl} \leq x_{ik}, \quad \forall (i,j) \in E_t, \forall (k,l) \in E_n, \tag{11}$$

$$x_{ik} + x_{jl} - 1 \leq f_{ijkl} \leq x_{jl}, \quad \forall (i,j) \in E_t, \forall (k,l) \in E_n. \tag{12}$$

Hence, (3) can be rewritten in a linear form as:

$$t_i^s \geq \max_{j \in pred_i} \{t_j^e + \sum_{(k,l) \in E_n} \frac{f_{ijkl} S_{ji}}{B_{kl}}\}, \forall i \in V_t, \tag{13}$$

Now, we can linearize the MINLP problem into an integer linear programming problem.

$$\begin{aligned} &\textbf{ILP}: \\ &min: \quad t_m^e, \\ &s.t.: \quad (1), (2), (5)-(9), (11)-(13), \\ &\qquad x_{ik}, x_{ik}^l \in \{0,1\}. \end{aligned}$$

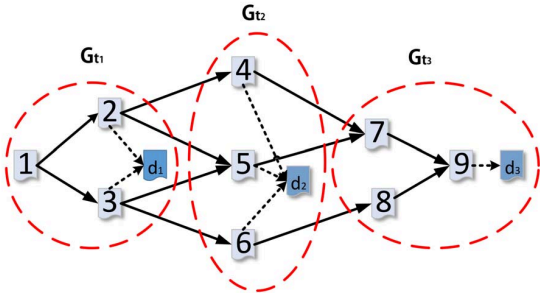


Fig. 4: subgraphs.

VI. Scaling Approach

The ILP described above is only able to solve small-scale instances because the problem is generally NP-hard. In this section, we present a Task Graph Partitioning based Integer Linear Programming approach (TGPLP) that consists of iteratively solving subproblems in which some parts of the problem have been fixed.

Initially, given task graph $G_t = (V_t, E_t)$, we aim to partition it to n legal subgraphs $\mathcal{W} = \{G_{t_1}, G_{t_2}, .., G_{t_n}\}$. A subgraph is said to be legal if a valid schedule can be derived from G_t such that the vertices of each subgraph appear consecutively in the schedule. Fig. 3(a) shows an example of legal partition, in which we can sequentially optimize $\{G_{t_1}, G_{t_2}, G_{t_3}\}$. While Fig. 3(b) shows an illegal partitioning because of the existence of a cycle among two subgraphs (e.g. the execution of task 5 in G_{t_1} needs data from task 2 in G_{t_2}, while task 2 needs data from task 1 in G_{t_1}).

As summarized in Algorithm 1, our algorithm contains two steps. The first step is task graph partitioning, we use topological sorting to equally partition the task graph into n legal subgraphs with the same amount of tasks (Line 1). Then we extend each subgraph G_{t_g} by adding a virtual task node $t^e_{d_i}$ whose execution delay is zero (Line 3). For each task $t_j \in G_{t_g}$ without successors in G_{t_g}, we add an directed link between t_j and $t^e_{d_i}$ as the dependency constraint (Lines 4-6). After the first step, a large-scale task graph can be partitioned into n subgraphs with virtual nodes as shown in Fig. 4. The second step is solving the ILP problems of those subgraphs sequentially (Lines 7-10). In this step, we use a set S to save tasks whose assignment and schedules are already fixed, for each subgraph, we solve the ILP subproblem with the objective of minimizing the finish time of virtual task where some of the variables have been fixed.

VII. Evaluation

We conduct extensive simulations to evaluate our approaches in this section. For comparison, we consider the following approaches:

- *ILP*: our integer linear programming approach.
- *TGPLP*: our ILP approach with task graph partitioning.
- *EA*: an evolutionary algorithm for task allocation in multi-robot teams proposed in related work [14].

Algorithm 1 Task Graph Partitioning based LP Algorithm

Input: $G_t = (V_t, E_t)$, n
Output: t^e_m
1: Topologically sort task graph G_t into n subgraphs $\mathcal{W} = \{G_{t_1}, G_{t_2}, .., G_{t_n}\}$
2: $S \leftarrow \varnothing$
3: **for all** subgraph $G_{t_g}, g = 1, 2, ..., n$ **do**
4: add v_{d_g} into G_{t_g};
5: **for all** task $t_j \in G_{t_g} \wedge succ(t_j, G_{t_g}) = \varnothing$ **do**
6: add e_{j,d_g} into G_{t_g};
7: **end for**
8: Construct an ILP problem by intergrating fixed $t^{e'}_i, x'_{ik}, y'_{ij}, i \in S, j \in V_t, k \in V_n$
9: Solve above ILP, and get the solution of $t^e_j, j \in G_{t_i}$
10: $S \leftarrow S \cup V_{t_i}$
11: **end for**
12: $t^e_m \leftarrow t^e_{d_n}$
return t^e_m

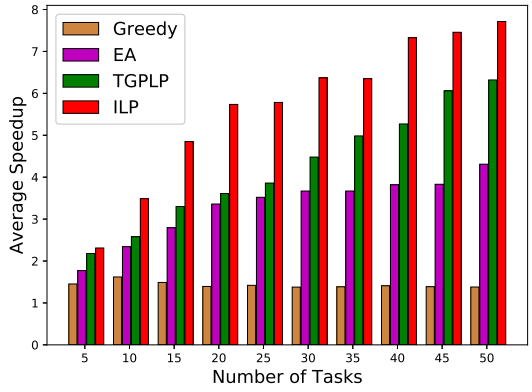


Fig. 5: Impact of number of tasks.

- *Greedy*: starting from the first task in the task graph, greedily assign one by one in the following to the nodes that result in the earliest finish time.

A. Simulation Settings and Comparison Metric

In our experiments, we consider a fully connected CCRS network containing 5 robots and 1 cloud server. We use the commercial solver Gurobi[1] to solve the ILP problems involved. The popular task graph generating program DAGGEN [25] is modified to obtain the random and synthetic DAGs used in the experiments. We control the task graphs and the cloud robotic networks through the following parameters: 1) number of tasks; 2) number of robots; 3) communication-to-computation ratio (CCR), which is defined as the ratio of the average communication cost to the average computation cost.

The default settings in our experiments are as follows: The edge weights of task graphs are randomly set in the range of $[10, 30]$ and $CCR = 0.2$ with $n = 3$ subgraphs in

[1]http://www.gurobi.com

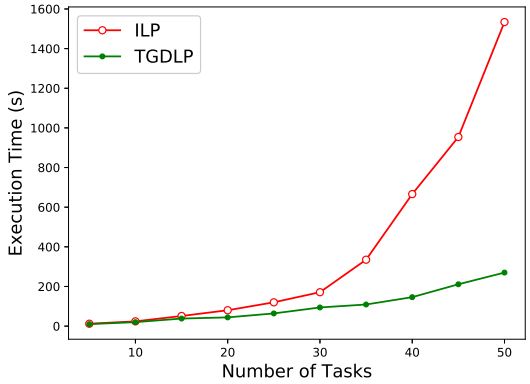


Fig. 6: Comparison on scalability.

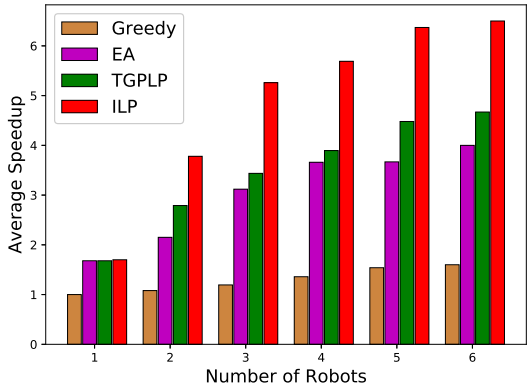


Fig. 7: Impact of number of robots.

TGPLP. The transmission rate in R2C network B_{kl} follows a Gaussian distribution with mean 5 and variance 2. Since R2R networks generally provide higher data transmission rate than R2C networks, we have $B_{RR} = \gamma B_{RC}$, and the default value of γ is 10. By default, we run 30 random task graphs in each simulation with randomly generated tasks.

The performance metric we consider in the evaluation is the $Speedup$, which is defined as the ratio of the sequential execution time to the parallel execution time (i.e., latency):

$$Speedup = \frac{min_{m_i \in M}\{\sum_{v_i \in V} w_{i,t}\}}{latency}, \tag{14}$$

where the numerator is the sequential execution time computed by assigning all tasks to a single robot that minimizes the total computation time. Our simulation was conducted using an Intel(R) Core 6 CPU, 3.4 GHz and 128 Gbytes memory. Python 3.6 was used for algorithm implementation.

B. Evaluation Results

We first check the performance comparison of the four algorithms, i.e., ILP, TGPLP, EA, and Greedy. We randomly generate 10 task graphs by varying the number of tasks from 5 to 50. Fig. 5 shows the comparison results of the four approaches. We first notice that our ILP approach consistently finds optimal solutions whenever the problem is feasible. On average, the ILP approach has the significant improvement by 35% compared to TGPLP, 70% compared to EA, 305% compared to Greedy. Greedy shows poor performance because it assigns tasks one by one in the computing node that results in the earliest finish time, which could not obtain the global optimal solution. Although EA can visibly outperform Greedy approach, it is not as good as our TGPLP approach. As the number of tasks increases, TGPLP shows its advantage in comparison to EA and Greedy. For example, when there are 50 tasks, our TGPLP improves almost 47% speedup in comparison with EA approach, which shows that our TGPLP is more flexible than EA. It is interesting to see that when $25 \leq n \leq 50$, the performance of EA almost remains the same speedup. We also notice that the gap between TGPLP and ILP gradually narrows as the number of tasks increases. The reason is that data correlation among different subgraphs become weakened as the number of tasks increases so that the assignment and scheduling of previous tasks make less effect on the later tasks. In overall, our TGPLP approach achieves approximately the optimal performance, dramatically outperforms Greedy and EA.

As we can see in Fig. 6, it is quite time-consuming to get an optimal solution using ILP. The complexity of ILP approach is exponential and leads to computational intractability as the task number increases. TGPLP approach, on the other hand, provides acceptable result but in a short time. For example, when there are 50 tasks, TGPLP produces speedup within 20% of the optimal solution while saves almost 90% execution time compared with ILP.

The influence of the number of robots is investigated by changing its value from 1 to 5. As shown in Fig. 7, the increase of the robot number is beneficial to raise the speedup. For example, when there is only one robot, the results are 1.05, 1.58, 1.68 and 1.70 under Greedy, EA, TGPLP, ILP, respectively. As the number of robots grows to 5, the speedup increases to 1.54, 3.66, 4.48, 6.37, respectively. There are two reasons for this phenomenon. First, when more robots work together, tasks can be completed faster by running the computations in parallel. Second, offloading computational-intensive tasks to suitable nodes (e.g. the cloud) can also reduce the task execution time.

We finally study the influence of CCR by changing its value from 0.07 to 5 with 50 tasks per task graph. Since ILP is time-consuming in this experiment, we only compare the performance of TGPLP, EA, and Greedy. We can see from Fig. 8 that speedup decreases as CCR increases under all approaches. For example, when $CCR = 0.07$, the speedups are 6.33, 4.53, 1.46, respectively. As CCR grows to 5, the speedups of all the four algorithms drop to below 2, That is because the overhead of data exchange among nodes will overwhelm the benefits of task assignment under lager value of CCR. We also notice that when $CCR = 5$, the performance of EA is even worse than Greedy approach, while our TGPLP still performs better than the other approaches. We

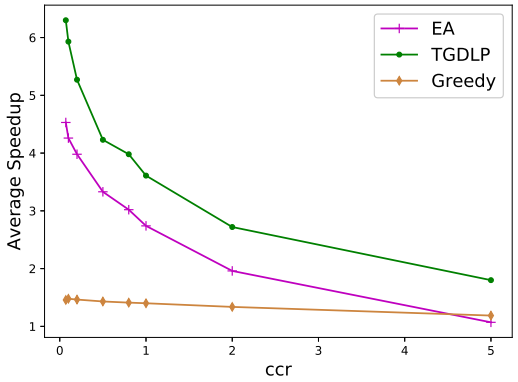


Fig. 8: Impact of CCR.

can conclude that EA approach is unsuitable for data-intensive tasks.

VIII. Conclusion and Future Work

In this paper, we explore the influence of task assignment and scheduling in CCRS to minimize the task completion latency. We first formulate the latency minimization problem as a MINLP and linearize it into an ILP. To tackle the high computational complexity of solving ILP, we further propose a partitioning-based scalable approach accordingly. Extensive simulation results show that our approaches outperform existing EA approach and Greedy approach. Our ILP approach can produce the optimal solution with minimal latency whenever the problem is feasible, whereas our TGPLP approach can find a "good enough" result within an acceptable execution time.

In the future, we intend to improve the scalability of TGPLP with effective heuristic algorithms. Further, we intend to take into consideration some real-world applications under cloud robotic systems; and also develop online algorithms to adapt the scenarios with high dynamicity.

Acknowledgment

The work described in this paper was supported by the National Key R&D Program of China2018YFB1003800), the Guangdong Province Universities and Colleges Pearl River Scholar Funded Scheme 2016, the National Natural Science Foundation of China (No.61722214), and and the Program for Guangdong Introducing Innovative and Enterpreneurial Teams(No.2017ZT07X355). Wuhui Chen is the corresponding author.

References

[1] J. Kuffner, "Cloud-enabled humanoid robots," in *Humanoid Robots (Humanoids), 2010 10th IEEE-RAS International Conference on, Nashville TN, United States, Dec.*, 2010.

[2] K. Goldberg and B. Kehoe, "Cloud Robotics and Automation:A Survey of Related Work," ... *Tech. Rep. UCB/EECS-2013-5*, pp. 1–12, 2013.

[3] M. Bonaccorsi, L. Fiorini, F. Cavallo, A. Saffiotti, and P. Dario, "A Cloud Robotics Solution to Improve Social Assistive Robots for Active and Healthy Aging," *International Journal of Social Robotics*, vol. 8, no. 3, pp. 393–408, 2016.

[4] M. C. Gombolay, R. Wilcox, and J. Shah, "Fast Scheduling of Multi-Robot Teams with Temporospatial Constraints," *Robotics: Science and Systems*, pp. 1–8, 2013.

[5] E. Nunes, M. McIntire, and M. Gini, "Decentralized multi-robot allocation of tasks with temporal and precedence constraints," *Advanced Robotics*, pp. 1–15, 2017.

[6] W. Chen, I. Paik, and Z. Li, "Cost-Aware Streaming Workflow Allocation on Geo-Distributed Data Centers," *IEEE Transactions on Computers*, vol. 66, no. 2, pp. 256–271, 2017.

[7] L. Yi, X. Deng, M. Wang, D. Ding, and Y. Wang, "Localized Confident Information Coverage Hole Detection in Internet of Things for Radioactive Pollution Monitoring," *IEEE Access*, vol. 5, pp. 18665–18674, 2017.

[8] Z. Tang, L. T. Yang, M. Lin, and B. Wang, "Confident Information Coverage Hole Healing in," vol. 14, no. 5, pp. 2220–2229, 2018.

[9] D. Zhang, L. T. Yang, M. Chen, S. Zhao, M. Guo, and Y. Zhang, "Real-Time Locating Systems Using Active RFID for Internet of Things," *IEEE Systems Journal*, vol. 10, no. 3, pp. 1–10, 2014.

[10] B. D. Gouveia, D. Portugal, D. C. Silva, and L. Marques, "Computation Sharing in Distributed Robotic Systems: A Case Study on SLAM," *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 2, pp. 410–422, 2015.

[11] L. Wang, M. Liu, and M. Q. Meng, "A Hierarchical Auction-Based Mechanism for Real-Time Resource Allocation in Cloud Robotic Systems," *IEEE Transactions on Cybernetics*, vol. 47, no. 2, pp. 473–484, 2017.

[12] L. E. Parker, "ALLIANCE: An architecture for fault tolerant multirobot cooperation," *IEEE Transactions on Robotics and Automation*, vol. 14, no. 2, pp. 220–240, 1998.

[13] M. Badreldin, A. Hussein, and A. Khamis, "A comparative study between optimization and market-based approaches to multi-robot task allocation," *Advances in Artificial Intelligence*, vol. 2013, p. 12, 2013.

[14] M. U. Arif, "An evolutionary algorithm based framework for task allocation in multi-robot teams.," in *AAAI*, pp. 5032–5033, 2017.

[15] R. Arumugam, V. Enti, L. Bingbing, W. Xiaojun, K. Baskaran, F. Kong, A. Kumar, K. Meng, and G. Jit, "DAvinCi: A cloud computing framework for service robots," *International Conference on Robotics and Automation*, pp. 3084–3089, 2010.

[16] M. L. Lam and K. Y. Lam, "Path planning as a service PPaaS: Cloud-based robotic path planning," *2014 IEEE International Conference on Robotics and Biomimetics, IEEE ROBIO 2014*, pp. 1839–1844, 2014.

[17] W. J. Beksi, J. Spruth, and N. Papanikolopoulos, "CORE: A Cloud-based Object Recognition Engine for robotics," *IEEE International Conference on Intelligent Robots and Systems*, vol. 2015-Decem, pp. 4512–4517, 2015.

[18] O. Zweigle, R. van de Molengraft, R. D'Andrea, and K. Häussermann, "RoboEarth: connecting robots worldwide," *Proceedings of the 2nd International Conference on Interaction Sciences: Information Technology, Culture and Human*, pp. 184–191, 2009.

[19] B. D. Gouveia, D. Portugal, D. C. Silva, and L. Marques, "Computation Sharing in Distributed Robotic Systems: A Case Study on SLAM," *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 2, pp. 410–422, 2015.

[20] D. Hunziker, M. Gajamohan, M. Waibel, and R. D'Andrea, "Rapyuta: The roboearth cloud engine," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pp. 438–444, IEEE, 2013.

[21] L. Agostinho, L. Olivi, G. Feliciano, F. Paolieri, D. Rodrigues, E. Cardozo, and E. Guimaraes, "A cloud computing environment for supporting networked robotics applications," in *Dependable, Autonomic and Secure Computing (DASC), 2011 IEEE Ninth International Conference on*, pp. 1110–1116, IEEE, 2011.

[22] W. Chen, Y. Yaguchi, K. Naruse, Y. Watanobe, and K. Nakamura, "Qos-aware robotic streaming workflow allocation in cloud robotics systems," *IEEE Transactions on Services Computing*, doi: 10.1109/TSC.2018.2803826, 2018.

[23] Z. Cheng, P. Li, J. Wang, and S. Guo, "Just-in-time code offloading for wearable computing," *IEEE Transactions on Emerging Topics in Computing*, vol. 3, no. 1, pp. 74–83, 2015.

[24] L. Wang, M. Liu, and M. Q. Meng, "Real-Time Multisensor Data Retrieval for Cloud Robotic Systems," *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 2, pp. 507–518, 2015.

[25] F. Suter and S. Hunold, "Daggen: A synthethic task graph generator." accessed on April 11, 2017. [online]. Available: https://github.com/frs69wq/daggen.