Communication-Computation co-Design of Decentralized Task Chain in CPS Applications

Seyyed Ahmad Razavi and Eli Bozorgzadeh
Computer Science Dept.
University of California, Irvine
Email: srazavim,eli@uci.edu

Solmaz S. Kia
Mechanical and Aerospace Eng. Dept.
University of California, Irvine
Email: solmaz@uci.edu

***Abstract*—In this paper, we present a method to find an optimal trade-off between computation and communication of decentralized linear task chain running on a network of mobile agents. Task replication has been deployed to reduce the data links among highly correlated nodes in communication networks. The primary goal is to reduce or remove the data links at the cost of increase in computational load at each node. However, with increase in complexity of applications and computation load on end devices with limited resources, the computational load is not negligible. Our proposed selective task replication enables communication-computation trade-off in decentralized task chains and minimizes the overall local computation overhead while keeping the critical path delay under a threshold delay. We applied our approach to decentralized Unscented Kalman Filter (UKF) for state estimation in cooperative localization of mobile multi-robot systems. We demonstrate and evaluate our proposed method on a network of 15 Raspberry Pi3B connected via WiFi. Our experimental results show that, using the proposed method, the prediction step of decentralized UKF is faster by 15%, and for the same threshold delay, the overall computation overhead is reduced by 2.41 times, compared to task replication without resource constraint.**

I. Introduction

With recent paradigm shift from cloud computing (high centralized computation) to edge and on-device computing, distributed and decentralized architectures have brought computation closer to sensor data on end devices [1][2]. Applications such as deep learning, sensor data fusion, and other compute-intensive algorithms are being decentralized and processed locally on end devices such as mobile robots and drones [3] [4]. Such decentralized algorithms, due to their distributed nature, enhance sensor fusion, system fault tolerance, and data privacy for cyber physical system (CPS) applications. However, with increasing number of compute-intensive applications running on end devices, the computation load can get beyond what low power embedded processing systems can tolerate.

There has been a great effort on reducing the wireless data transfer between the nodes during decentralization. The ideal case is when each allocated task can be fully decoupled from the tasks on other agents. However, in practice, some applications are composed of highly correlated tasks. Once decentralized, the delay/energy overhead of data transfer among the nodes cannot be neglected. This paper addresses the balance between communication and computation load during decentralization of an application running on a network of multi-agent systems.

Task replication has been deployed to reduce the data links among highly correlated nodes in communication networks [5][6]. The primary goal is to reduce or remove the data links at the cost of increase in computational load at each node. In [7] and [8], the proposed scheduling algorithms use duplication to increase the performance of parallel tasks on computer clusters connected via Ethernet while reducing the energy overhead on processors, network card, and routers. The tasks can be executed on any arbitrary processor. In [7], a fast task scheduling based on replication is presented for heterogeneous systems. In their method, they deployed the idle time of processors for task replication without considering any computational resource constraints. The computational overhead has mostly been considered negligible compared to wireless communication delay. However, with increase in complexity of applications and computation load on end devices with limited resources, the computation load may lead to performance degradation. As a result, the communication overhead along with decentralized computation load contributes to total critical path delay of the target application.

Our proposed approach is a systematic CPS framework using *selective* task replication in order to balance communication and computation overhead in a decentralized task chain running on a network of mobile agents. We first generate multiple task replica sets for each agent, which provide trade-off between data communication and on-device computation cost. Among the replica sets for each agent, we select a configuration for each agent such that the total execution time of the decentralized task chain is met while minimizing the average computational overhead. We applied our approach to decentralized Unscented Kalman Filter (UKF) for state estimation in cooperative localization of mobile multi-robot systems [9]. Our results

978-3-9819263-2-3/DATE19/©2019 EDAA

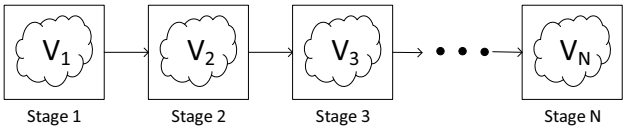

Figure 1: Linear chain model

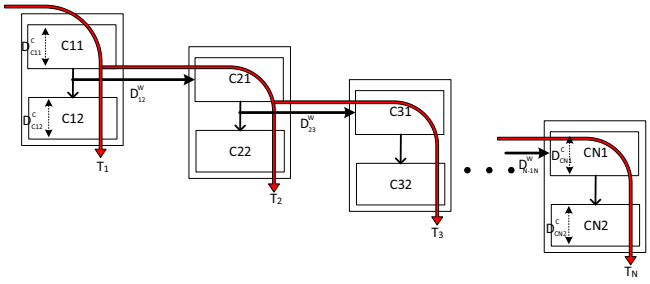

Figure 2: The timing of a linear chain task model

show that with selective task replication, the critical path delay constraint of 150 msec is met with $2.41x$ less CPU load, on average, compared to task replication without any resource constraints. To the best of our knowledge, this is the first effort on fully automated framework that orchestrates the communication and computation loads on decentralized correlated task chains to run efficiently on a network of mobile agents.

II. LINEAR CHAIN

Distributed applications can be modeled as directed acyclic graphs (DAG) where nodes represent the computations and edges represent the data dependency between the nodes. These DAGs may have special forms such as linear chain, fork, and tree [5]. In multi-robot systems and distributed embedded systems, some applications can be distributed as linear chain, for example Deep Neural Network[3], solving system of linear equations ($Ax = B$), and some families of filters such as Kalman filter [10]. In linear chain model, each stage receives data from the previous stage, and after processing, it sends data to the next stage. Figure 1 shows a linear chain with N stages. It can be modeled as a DAG $G = (V, E)$ where $V = V_1, V_2, \cdots, V_N$ are the partitions that are assigned to the agents (e.g. robots). Each partition is a subgraph of G containing the tasks and their local edges. E is a set of edges, representing the data dependency between partitions. The edges are directional and only connect the two consecutive agents in the chain. In this paper, the communication link between mobile agents is wireless (e.g., WiFi). Some inputs of a partition are generated locally (e.g. sensor data), and some have to be received from the previous partition. Each partition may generate local output as well as required input data for the next partition in the chain.

Figure 2 shows the linear chain task model and the application flow. For example, stage 2 computes $C12$ after receiving data from stage 1, and sends data to stage 3. In standard TCP/IP, the data will be copied to OS kernel. Therefore, agent 2 computes $C22$ without waiting to send the data over the air. After receiving data, stage 3 starts computing $C31$. Each stage has a path generating the corresponding local output, and also partially contributes in the delay of the path of its successors. For example, in the same figure, $C21$ is on the critical path of stage $3(T3)$, and all successor stages. The delay of path ending in stage 2 ($T2$), is calculated as the sum of the delay of computation on stage 1 (D^c_{C11}), the delay of transferring data from stage 1 to stage 2 (D^w_{12}), and the delay of computation on agent 2 (D^c_{C21}, D^c_{C22}). T^a_i represents the time by when stage i receives all input data. Hence, considering linear chain model, T_2 is $T^a_2 + D^c_{C21} + D^c_{C22}$. D^{CP}_i refers to computation time of stage i that contributes to critical path of successor stages and D^{NCP}_i refers to computation time that only contributes in the critical path ending in stage i. Hence, T^a_3 will be $T^a_2 + D^{CP}_3 + D^c_{C21}$, and the T_3 will be $T^a_2 + D^{CP}_3 + D^{NCP}_3$ ($D^{CP}_2 = D^c_{C21}$ and $D^{NCP}_2 = D^c_{C22}$). In general,

$$T^a_i = T^a_{i-1} + D^{CP}_{i-1} + D^w_{i-1,i} \tag{1a}$$

$$T_i = T^a_i + D^{CP}_i + D^{NCP}_i \tag{1b}$$

T^a_1 is 0 since the first stage does not need to receive data from any agents. The critical path delay of the task chain is the maximum of delay of all the paths (Equation 2).

$$T_{critical} = \max_{1 \le i \le N} (T_i) \tag{2}$$

Using Equations 1 and 2, it is possible to calculate the delay of critical path incrementally, starting from the first stage. Most CPS applications have to run periodically in less than a threshold delay ($T_{threshold}$). Our goal is to achieve a decentralized task chain such that $T_{critical}$ stays under $T_{threshold}$ with minimum computation overhead due to task replication.

III. COMPUTATION-COMMUNICATION TRADE-OFF FOR TWO PARTITIONS

Replication is one of the well known methods to reduce the communication. It reduces the size of data transfer at the cost of increase in computational load at each agent. In this section, we present our method to generate various replicated graphs for a two-way partitioned graph. Generated task graphs provide a trade off between computation and communication. Let's assume a decentralized application divided into two partitions V_1, V_2. There may exists a subgraph in V_1, $R_1 \in V_1$ such that if it is replicated in V_2, it may reduce the cutsize between the two partitions. Task replication algorithm searches for such a replication set in V_1. After replication of R_1 in V_2, the cut edges between R_1 and V_2 are eliminated from the cut set and the input edges to R_1 are added to cutsize. Figure 3.a shows an example of

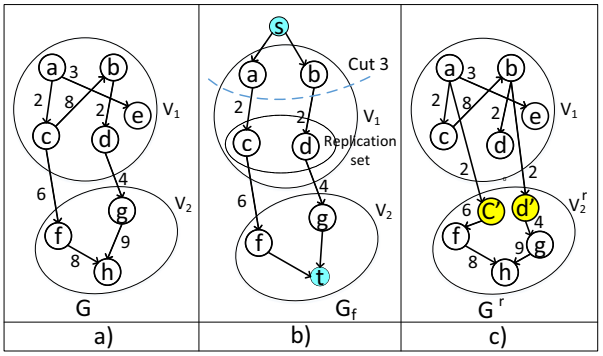

Figure 3: An example of Replication a) Original graph(G), b) G_f and min cut, c) replicated graph G^r

G. Lets assume that I is the set of incoming edges to set V_2 from V_1. To find the minimum replication set, we construct a network graph $G_f = (V_f, L_f)$, where V_f includes nodes in V_1 that are reachable from I (nodes a, b, c, and d) and the nodes in V_2 that are adjacent to I (nodes f, and g) (Figure 3.b). To apply Min-cut max-flow theorem [11], two dummy nodes are added to G_f as a source (node s) and a sink (node t). The Min-cut max-flow theorem on this graph will separate the replication set from the rest of the network. The replication set R_1 is the subset of nodes in V_1 starting from min-cut edges to the nodes adjacent to I. As shown in Figure 3.c, after replicating R_1 (node c and d) in V_2, the cut edges between R_1 and V_2 (edge $c \rightarrow f$, and $d \rightarrow g$) are eliminated from the cut set and the input edges to R_1 are added to cutsize (edge $a \rightarrow c'$ and $b \rightarrow d'$). The replicated partition is called V_2^r. This method gives a replicated graph with minimum cutsize between V_1 and V_2^r.

Task replication comes with the cost of increase in the computation load in V_2. If there is no limitation on the weight of replicated nodes, the added computation load may not be efficiently manageable in the systems with limited computational resources. To control the computation overhead of the replication set, the weight of replication set should be limited. i.e. $W_{R_1} \leq M$. In [12], the authors presented a method called Hyper-MAMC to incrementally find the replication set with a bounded size by starting from the initial result of Min-cut max-flow. We adopt Hyper-MAMC algorithm to replicate the task chains. However, the algorithm works between two partitions only and it does not bound the critical path delay. Using Hyper-MAMC algorithm, we generate multiple replicated sets under various CPU load constraints (M). Among those, the Pareto points will be selected as a set of configurations for each V_1 in the global search for optimum solution on the task linear chain.

The proposed algorithm (Replication-2way) is shown in Alg. 1. The weight associated with each node is the CPU cycle counts of its computation. The M varies between 0 and the sum of the weights of the nodes in V_1 to get various

Algorithm 1: Replication-2way

input : 2-way partition directional Graph $G = V_1, V_2$
output: $CList$:list of Pareto point V_2^r

1 $CList = []$
2 **for** M *in* $(0, max_{inst.}, l)$ **do**
3 $\quad V_2^r = HyperMAMC(G, M)$
4 $\quad CList.push((V_2^r, W(V_2^r), cutsize(V_1, V_2^r)))$
5 **end**
6 **for** *each* $(V_2^r, Comp, cut)$ *in* $Sorted(CList, cutsize)$ **do**
7 $\quad$ **if** $Comp < Min$ **then**
8 $\quad\quad Min = Comp$
9 $\quad$ **end**
10 $\quad$ **else**
11 $\quad\quad CList = CList - (V_2^r, Comp, cut)$
12 $\quad$ **end**
13 **end**
14 Return $(CList)$

replicated graph (Alg. 1,line 2). Each V_2^r in $CList$ indicates a trade-off between computation load and the size of data transfer. Hence, each V_2^r in $CList$ is a point in computation-communication space. By walking through the sorted points based on the cutsize (Alg.1, line 6), and eliminating the points with higher computation load than the minimum observed thus far, we find the Pareto point V_2^r graphs.

For example, lets assume that the weight of all the nodes is 1 in the graph in Figure 3.a. Replication-2way will generate multiple V_2^r, as shown in Figure 4. For $M = 0$, no node will be replicated (Cut 1), and the cutsize will remain intact. For $M = 1$, node c will be replicated (Cut 2) which eliminates

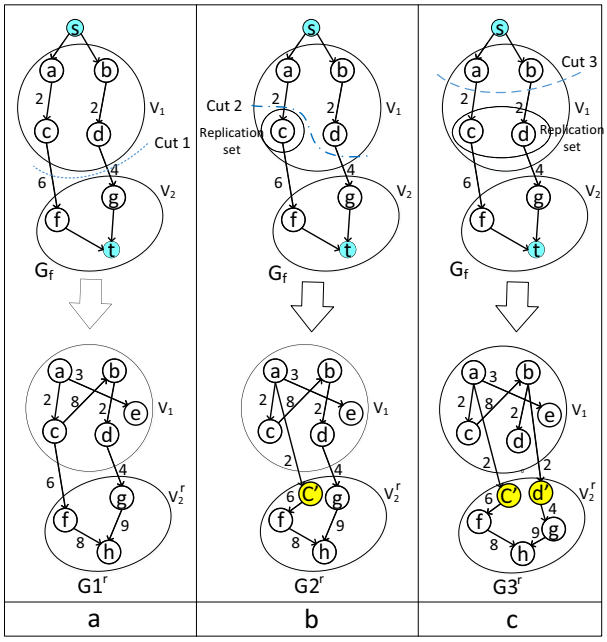

Figure 4: An example of constrained Replication a) $M = 0$, b) $M = 1$, c) $M = 2$

edge $c \rightarrow f$ and adds edge $a \rightarrow c'$, which reduce the cutsize to 6. Finally, for $M = 3$, node a and b will be replicated (Cut 3) which reduce the cutsize to 4. For this example, $CList$ is $[(V1_2^r, 3, 10), (V2_2^r, 4, 6), (V3_2^r, 5, 4)]$.

IV. Selective Replication on linear chain

Using replication, it is possible to minimize the size of transferred data. Although it reduces the communication delay, the replicated computations might increase the critical path delay. As shown earlier, we generate a group of replication sets among which there is a trade-off between their cutsizes (communication cost) and size of their replicated sets (computation cost). Next, we present an algorithm to select the optimum configuration.

Problem Formulation- we are given a N-way partition of $G = (V, E)$ represented by $V = V_1, V_2, \cdots, V_N$, and a threshold delay $T_{threshold}$. The V_i is a set of nodes in V that belongs to partition i, and E represents the data dependency between the nodes. The edges are directional and only occur between each two consecutive partitions, i.e. for all $e(u_i, u_j) \in L$, if $u_i \in v_i$, then $u_j \in v_i | u_j \in v_{i+1}$. The objective is to find a $G^r = (V^r, E^r)$ which $V^r = V_1^r, V_2^r, \cdots, V_N^r$ where V_i^r is the replicated set in V_i, such that the sum of the weight of all nodes in G^r is minimum while the delay of total critical path of G^r is less than $T_{threshold}$.

We use a dynamic programming approach to select one replicated set for each partition among the Pareto points such that the total critical path delay does not exceed a given threshold and the total computation overhead is minimized. Algorithm 2 shows the proposed method. If there are more than two stages in linear chain, applying Replication-2way on each two adjacent partitions will generate local optima. The subgraphs in previous partitions may belong to a reachable set of the current partition and hence, are potential candidates for replication sets. Therefore, to find the list of Pareto point replicated graphs of each V_i ($temp$), we search for R_i in all predecessors of V_i ($\cup_{j=1}^{i-1} V_j$) using Replication-2way (Algo. 2, line 7).

Because the graph is partitioned as a linear chain, any feasible configuration of $V_1, V_2, \cdots, V_{i-1}$ only differs in the weight of the edges, and hence, the T_{i-1}^a and T_i will be computed incrementally based on LV_{i-1}^r and V_i^r in lines 10 and 11. Therefore, for generating V_i^r, it is not required to apply Replication-2way on all feasible configurations (LV_{i-1}^r) generated so far (PS). If the T_i of V_i^r is less than the $T_{threshold}$, it will be added to LV_i^r as a feasible replicated set considering LV_{i-1}^r (line 13). If no replication set is found to keep the critical path delay under the $T_{threshold}$, it will be removed from PS (Line 17). In line 14, the $([LV_i^r], T_i^a, Cost + W(V_i^r))$, which is a tuple of ([the list of valid configurations for agent 1 to i], T_i^a, and the total weight of nodes), will be added to PS to be used in selecting the replication set for stage $i+1$. After this step, the $T_{threshold}$ of all the remaining configurations in PS will be less than $T_{threshold}$. At the end, the configuration with the lowest computation overhead ($Cost$) will be selected.

Algorithm 2: Selective_Replication

input : N-way partition of Linear Task Chain Graph $V = V_1, V_2, V_3, \cdots, V_N$ and $T_{threshold}$
output: PS

```
1  //Initialization;
2  T_1^a = 0
3  LV_1^r = [V_1]
4  Cost = W(V_1)
5  PS=[(LV_1^r, T_1^a, Cost)];
6  for i in [2:N] do
7      temp = Replication_2way(⋃_{j=1}^{i-1} (V_j), V_i)
8      for each (LV_{i-1}^r, T_{i-1}^a, Cost) in PS do
9          for each V_i^r in temp do
10             T_i^a = T_{i-1}^a + D_{i-1}^{CP} + D_{i-1,i}^w
11             T_i = T_i^a + D_i^{CP} + D_i^{NCP}
12             if T_i < T_threshold then
13                 LV_i^r = LV_{i-1}^r + V_i^r
14                 PS.push(([LV_i^r], T_i^a, Cost + W(V_i^r)))
15             end
16         end
17         PS = PS − (LV_{i-1}^r, T_{i-1}^a, Cost)
18     end
19 end
20 Return ((LV_N^r, T_N^a, Cost) in PS with minimum Cost)
```

V. Case Study: Decentralized UKF in Cooperative Localization

The fast and accurate localization of mobile agents (or robots) is a crucial task since a delayed estimation will mislead the robots and might lead to mission failure. The time interval between executions of localization ($T_{threshold}$) should be small enough to be able to capture the motion of all the robots and provide applications with accurate and almost real time location [10]. Cooperative localization is a promising localization method in GPS-denied environment. In Cooperative Localization, robots estimate their location based on local sensor data, such as accelerometer, and correct their estimated location by the measured relative distance, using sensors such as Kinect or WiFi signal strength. Since sensors are noisy, various variants of Kalman filter is used for state estimation so as to improve the accuracy of the localization [13]. Our target application is decentralized Unscented Kalman Filter (UKF) in Cooperative Localization for multi-robot systems [9]. UKF is a recursive filter for estimating the state of a system (here location) referred to as $\mathbf{x}^+$. UKF is composed of prediction and update

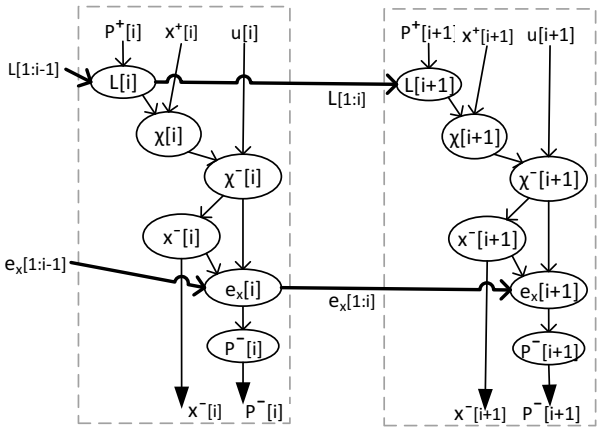

Figure 5: Decentralized UKF without replication

steps. Prediction step runs periodically, and update step runs whenever there is a measurement in the system.

Equation 3 shows the prediction step of UKF. K represents the time. The prediction step for the collective system with n states starts with computing the square root matrix, as a triangular matrix, of matrix $\mathbf{P}^{+}(k)$ using Cholesky Decomposition (CD) method. After that, a set of $2n+1$ sample points, called Sigma Points ($\boldsymbol{\chi}$), is generated by eq. 3b. In the equations, $_{(c)}$ denotes the c^{th} column of the matrix. The system model function will use Sigma points and $\mathbf{u}(k)$ to generate Transformed Sigma Points (eq. 3c). The predicted state is the weighted arithmetic mean of $\boldsymbol{\chi}^{-}$s (eq.3d). Finally, the predicted covariance matrix ($\mathbf{P}^{-}(k+1)$) will be obtained by equation 3f using prediction error $\mathbf{e}_x$ (eq.3e). In the equations, $c \in \{0, \cdots, 2n\}$, $l \in \{1, \cdots, n\}$, and w are system defined constant.

$$\mathbf{L}(k) = CD(\mathbf{P}^{+}(k)) \tag{3a}$$

$$\boldsymbol{\chi}_{(0)} = \mathbf{x}^{+}(k), \boldsymbol{\chi}_{(l,l+n_x)} = \mathbf{x}^{+}(k) \pm \sqrt{(n_x+\kappa)}[\mathbf{L}(k)]_l \tag{3b}$$

$$\boldsymbol{\chi}^{-}_{(c)} = \mathbf{f}(\boldsymbol{\chi}_{(c)}(k), \mathbf{u}(k)), \quad c \in \{0, \cdots, 2n_x\} \tag{3c}$$

$$\mathbf{x}^{-}(k+1) = \sum_{c=0}^{2n_x} w_{(c)}\, \boldsymbol{\chi}^{-}_{(c)} \tag{3d}$$

$$\mathbf{e}_{x,(c)} = \boldsymbol{\chi}^{-}_{(c)} - \mathbf{x}^{-}(k+1) \tag{3e}$$

$$\mathbf{P}^{-}(k+1) = \sum_{c=0}^{2n_x} w_{(c)} \mathbf{e}_{x,(c)} \mathbf{e}^{\top}_{x,(c)} + \mathbf{B}(k)\mathbf{Q}(k)\mathbf{B}(k)^{\top} \tag{3f}$$

$\mathbf{P}^{-}(k+1)$ and $\mathbf{x}^{-}(k+1)$ might be used as $\mathbf{P}^{+}(k+1)$ and $\mathbf{x}^{+}(k+1)$ for the next UKF iteration.

In the decentralized UKF, each partition i is assigned to robot i. Each robot i has to compute its location, i.e. $\mathbf{P}^{-}[i]$, and $\mathbf{x}^{-}[i]$ using its locally generated control signals($\mathbf{u}[i]$), and $\mathbf{L}[1:i-1]$ and $\mathbf{e}_x[1:i-1]$ that receive from the last robot (robot $i-1$) [10][9]. Figure 5 shows the task graph of robot i and $i+1$ for decentralized UKF. Therefore, the task graph of decentralized UKF can be modeled as a *linear chain* (Figure 1), and its critical path delay can be estimated by equation 1. There are two edges, $\mathbf{L}[1:i]$ and

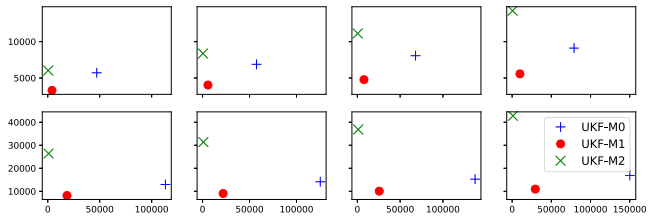

Figure 6: The UKF-M0, UKF-M1, and UKF-M2 computation-communication points, N=15, X-axis: communication, Y-axis: computation

$\mathbf{e}_x[1:i]$, between each robot i and robot $i+1$. By applying *Replication_2way* (Algo. 1), various edges between the partitions can be eliminated and multiple replication sets are generated. We refer to them based on their replication level in ascending order, such as UKF-M1 and UKF-M2. In UKF-M1, $\mathbf{L}[1:i]$, and in UKF-M2, $\mathbf{u}[1:i]$ will be transferred between agent i and agent $i+1$. We call the original task graph of decentralized UKF as UKF-M0, where no task is replicated. The UKF-M0 is inferior to UKF-M1 because it has more communication and computation (Figure 6). Although in UKF-M1, some of the UKF tasks are replicated, the CPU overhead of UKF-M0 can still be higher. Due to high volume of data transfer in UKF-M0, more CPU cycles are consumed for data serialization and TCP/IP stack. The output of *Selective_Replication* for UKF (called SR-UKF) is a list of UKF task graph for all agents, including UKF-M1 and UKF-M2.

VI. Experiments

We implemented out proposed framework on a network of single-board embedded devices, i.e., Raspberry Pi 3 B, with quad-core 64bit CPU, 1 GB main memory, and a built-in WiFi module of 802.11 b/g/n. During our experiments, the CPU frequency is set to 600 MHz, and we used all four cores of the CPU. Each device i runs Linux kernel v4.9 and an application process that implements robot i's task for decentralized UKF in C++ with an open-source linear algebra library EIGEN [14]. To create a realistic network environment, we set up an isolated WiFi network with one access point, Netgear N300 wireless router and N boards. In the following, for all parameters, we used their median values obtained from 500 rounds of UKF. In our experimental studies, we considered robotic team scenarios with robots with 6 local states and with measurement and control signals size of 3. In our implementation, we measured and analyzed the total number of CPU cycles used for the decentralized UKF task, and the critical path delay of each execution cycle of decentralized UKF, which include both the computation and the communication delays on all the robots (or boards). We used Linux Perf to measure the number of CPU cycles associated with decentralized UKF [15].

Table I: The number of CPU cycles for various time intervals for a system of 15 agents

Proposed method (SR-UKF)			
T_{thre}(ms)	Configuration	T_{crit}(ms)	CPU cycles(K)
135	[1,1,1,1,1,1,2,1,1,1,1,1,1,2,1]	134	125,427
140	[1,1,1,1,1,1,1,1,1,1,1,1,2,1,1]	138	114,074
150	[1,1,1,1,1,1,1,1,1,1,1,1,2,1,1]	138	114,074
160	[1,1,1,1,1,1,1,1,1,1,1,2,1,1,1]	151	109,593
170	[1,1,1,1,1,1,1,1,1,2,1,1,1,1,1]	169	101,964
180	[1,1,1,1,1,1,1,1,2,1,1,1,1,1,1]	175	98,751
Reference configurations (not SR-UKF)			
—	FR-UKF	145	275,191
—	NR-UKF	827	136,376

In our proposed method, task replication is selectively applied such that the total CPU utilization is minimized while the total critical path delay ($T_{critical}$) is less than $T_{threshold}$. To show the effect of $T_{threshold}$ on the result, we run the algorithm for various values of $T_{threshold}$ on a team of 15 robots. Table I shows the result for one iteration of decentralized UKF. UKF-M1 and UKF-M2 are represented as 1 and 2, respectively. $T_{threshold}$ and $T_{critical}$ are in msec, and we report the total number of CPU cycles (in K). To show the effect of selective replication, we compared it with configuration where the Min-cut max-flow replication has been applied without any resource constraint and hence, the decentralized UKF task graphs has been Fully Replicated, called FR-UKF, adopted from [10]. To show the effect of replication, we also run the decentralized UKF with No Replication, called NR-UKF. The best $T_{critical}$ that can be achieved by FR-UKF is 145ms, which is higher than the $T_{critical}$ of SR-UKF (134 ms). This is due to the fact that the computation delay of the last node in FR-UKF is higher than the CPU time saved by task replication. The $T_{critical}$ of NR-UKF is 827ms which shows the speedup gained by reducing the size of transferred data using replication. By increasing the $T_{threshold}$, as expected, the total number of CPU cycles is reduced using the proposed method.

To show the effect of our proposed method with the same $T_{threshold}$, we set the $T_{threshold}$ to the minimum value that can be met by the FR-UKF. Table II shows the total number of CPU cycles and the $T_{threshold}$ for both methods for various number of agents. In all cases, our proposed method (SR-UKF) outperforms the FR-UKF. For 15 agents, the total number of CPU cycles associated with decentralized UKF task is improved by $2.41x$ compared to FR-UKF.

VII. Conclusions

we presented a method to find an optimal trade-off between computation and communication of decentralized linear task chain running on a network of mobile agents. Our proposed selective task replication enables communication-computation trade-off in decentralized task chains and minimizes the overall local computation overhead while keeping the critical path delay under a threshold delay. We applied our approach to decentralized UKF, and demonstrated and evaluated our proposed method on a network of 15 Raspberry Pi3 devices. Our experimental results show that, using the proposed method, the overall computation overhead of decentralized UKF is reduced by 2.41x, when compared to task replication method without resource constraint.

Table II: The number of CPU cycles for FR-UKF and SR-UKF with the same $T_{threshold}$

N	T_{thre}(ms)	FR-UKF	SR-UKF	Ratio
5	54	5451	3728	1.46
6	58	10001	6422	1.55
7	89	16909	9428	1.79
8	93	27114	15264	1.77
9	104	41340	23219	1.78
10	109	60725	32153	1.88
11	118	86472	43190	2.00
12	123	119167	56763	2.09
13	125	160540	72933	2.20
14	129	212410	91878	2.31
15	149	275191	114074	2.41

References

[1] I. Azimi *et al.*, "Hich: Hierarchical fog-assisted computing architecture for healthcare iot," *ACM Transactions on Embedded Computing Systems (TECS)*, 2017.

[2] D. Amiri *et al.*, "Edge-assisted sensor control in healthcare iot," in *Globecom SAC EH*, IEEE, 2018.

[3] R. Hadidi *et al.*, "Distributed perception by collaborative robots," *IEEE Robotics and Automation Letters*, 2018.

[4] D. Callegaro and M. Levorato, "Optimal computation offloading in Edge-Assisted UAV systems," in *Globecom SAC TI*, 2018.

[5] A. Benoit *et al.*, "A survey of pipelined workflow scheduling: Models and algorithms," *ACM Computing Surveys*, 2013.

[6] S. Ranaweera and D. P. Agrawal, "A task duplication based scheduling algorithm for heterogeneous systems," in *IPDPS*, IEEE, 2000.

[7] Z. Zong *et al.*, "Ead and pebd: two energy-aware duplication scheduling algorithms for parallel tasks on homogeneous clusters," *IEEE Transactions on Computers*, 2011.

[8] Y. Zhao *et al.*, "Dependency-based energy-efficient scheduling for homogeneous multi-core clusters," in *Trust, Security and Privacy in Computing and Communications, 2013 IEEE International Conference*.

[9] V. Dinh and S. S. Kia, "A server-client based distributed processing for an unscented kalman filter for cooperative localization," in *Multisensor Fusion and Integration for Intelligent Systems, 2015 IEEE International Conference on*.

[10] S. A. Razavi *et al.*, "Resource-aware decentralization of a UKF-based cooperative localization for networked mobile robots," in *Euromicro Conference on Digital System Design*, IEEE, 2018.

[11] E. Gamal *et al.*, "Optimal replication for min-cut partitioning," in *Computer-Aided Design, IEEE/ACM International Conference on*, 1992.

[12] H. H. Yang and D. Wong, "Optimal min-area min-cut replication in partitioned circuits," *IEEE transactions on computer-aided design of integrated circuits and systems*, 1998.

[13] S. S. Kia *et al.*, "Cooperative localization for mobile agents: A recursive decentralized algorithm based on kalman-filter decoupling," *IEEE Control Systems Magazine*, 2016.

[14] G. Guennebaud, B. Jacob, *et al.*, "Eigen v3." http://eigen.tuxfamily.org, 2010.

[15] "perf: Linux profiling with performance counters.https://perf.wiki.kernel.org/,"