LOAM: Low-latency Communication, Caching and Computation in Data-Intensive Computing Networks

Jinkun Zhang
Imperial College London, UK
jinkun.zhang@imperial.ac.uk

Edmund Yeh
Northeastern University, US
eyeh@ece.neu.edu

***Abstract*—Deploying data- and computation-intensive applications such as large-scale AI into heterogeneous dispersed computing networks can significantly enhance application performance by mitigating network resource bottlenecks, including bandwidth, storage, and computing power. However, current resource allocation methods do not provide a comprehensive solution that jointly considers arbitrary topology, elastic resource amount, reuse of computation results, and congestion-dependent optimization objectives. These aspects are vital when modeling state-of-the-art heterogeneous dispersed computing networks with high demand volume. In this paper, we propose LOAM, a low-latency joint communication, caching, and computation placement framework. LOAM incorporates the above aspects with a rigorous analytical foundation. It tackles the formulated NP-hard cost minimization problem with two methods: an offline method with a constant factor approximation of 1/2, and an online adaptive method with a bounded gap from the optimum. Through extensive packet-level simulation, LOAM outperforms multiple baselines in both synthesis and real-world network scenarios.**

I. Introduction

Over the past decade, centralized clouds, as shown in Fig. 1 (a), have been pivotal in IT service delivery. They are favored for their cost-effectiveness and ability to improve energy efficiency and computation speed for devices with limited resources.However, the rise of Internet of Things (IoT) devices and the demand for services requiring ultra-low latency (e.g., online VR/AR gaming, distributed learning) have highlighted the limitations of centralized clouds, prompting a shift towards dispersed computing paradigms like fog and edge computing. These paradigms distribute network resources, including bandwidth, storage, computing power, etc., closer to users, surpassing centralized architectures in meeting the needs of delay-sensitive applications. Dispersed computing outperformed centralized cloud architectures regarding request delay, scalability, error resilience, and AI/ML adaptation [1].

The move towards low-latency dispersed computing comes with the challenge of managing network resources to handle data- and computation-intensive applications, e.g., VR rendering and Large Language Model (LLM) inference. Existing dispersed computing protocols address network resource allocation control with different granularity. For example, in mobile edge computing (MEC), resources are distributed throughout the mobile edge close to users, where task offloading is considered at the application layer, and caching at the network layer [2]. However, current methods do not provide a comprehensive solution that jointly considers arbitrary network topology, elastic resource amount, computation result reuse, and congestion-dependent costs. These aspects can have a significant impact when modeling state-of-the-art heterogeneous computing networks, especially with high demand volume. By jointly considering forwarding (on what paths are requests routed), caching (what to put in network storage), and computation placement (where to offload computation jobs) in a cross-layer and mathematically rigorous manner, this paper is dedicated to providing theoretical insights and further pushing dispersed computing performance to its limits.

First, although existing architectures (e.g., mesh networks [3]) optimize various performance metrics (e.g., network throughput and fairness), most of them are designed for restricted network topologies and fixed resource amounts (e.g., total computing power or storage size). Whereas in emerging collaborative networks, arbitrary topologies and elastic network resource amounts are crucial. For example, device-edge-cloud fusion AI [4] can make trade-offs between resource efficiency and performance. Moreover, with the growth of smart devices, congestion mitigation has become a crucial design objective. Current frameworks can lead to severe suboptimality regarding average request latency, as the queueing effect caused by congestion is not reflected in the model. This paper overcomes these challenges by incorporating arbitrary multi-hop network topologies and directly considering congestion-dependent costs.

Second, although numerous joint optimization frameworks were proposed recently (e.g., forwarding, offloading, and caching [5]), with the increase of data- and computation-intensive applications, existing methods can suffer from sub-optimal interaction between computing and storage utilization. For instance, the throughput-optimal algorithm [6] does not consider practical scenarios where multiple requests for the same computation on the same data object are generated concurrently (e.g., multiple VR users making rendering requests for the same object) In these scenarios, compared to purely moving data and computation [6], directly caching computational results (i.e., reuse of computational results, or *computation reuse*) can significantly reduce request latency [7]. Despite such advantage, computation reuse is often considered separate from other resources.

This paper is the first to achieve latency-minimal resource allocation for cache-enabled dispersed computing with arbitrary topology, elastic resource amounts, computation reuse,

ISBN 978-3-903176-73-7

and congestion-dependent objectives. We propose a conceptually novel and analytically rigorous framework named *LOAM* (low-Latency cOmmunication, cAching, and coMputation). In LOAM, nodes generate computation requests on data objects (e.g., DNN inference with a pre-trained model). The computation requests are offloaded to nodes with adequate computing power, and corresponding data requests are generated to fetch the required data (e.g., pre-trained model). Nodes are equipped with storage to temporarily store data and/or computation results. Nonlinear costs are incurred on the links and nodes due to transmission, computation, and caching. LOAM aims to minimize the network aggregated cost jointly over forwarding, caching, and offloading strategies.

LOAM tackles this NP-hard optimization problem with a solid analytical foundation. It provides an offline algorithm and an online adaptive algorithm, both with performance guarantees. The offline solution archives a $1/2$ approximation to an equivalent problem by exploiting its "submodular + concave" structure, however, requiring prior knowledge of network status and is not adaptive. The online solution guarantees a bounded gap from the optimum using the problem's "convex + geodesic-convex" structure, and can adapt to changes in network status. Proof of lemmas, theorems, and corollaries in this paper can be found in [8]. Our key contributions are:

- We model heterogeneous multi-hop cache-enabled computing networks with elastic resources, computation reuse, and nonlinear congestion-dependent costs.
- We provide performance guarantees for proposed NP-hard cost minimization problem: an offline solution with $\frac{1}{2}$ approximation via Gradient-combining Frank-Wolfe; and an online adaptive solution with a bounded gap from the optimum via a modification to the KKT condition.
- We provide a public-available packet-level network simulator. Extensive simulation shows that proposed solutions significantly outperform baselines in multiple scenarios.

II. Network Model

A. Cache-enabled computing network

Consider a network modeled by a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of nodes and $\mathcal{E}$ is the set of links. We assume $(j,i) \in \mathcal{E}$ if $(i,j) \in \mathcal{E}$. Nodes have heterogeneous capabilities of performing computing tasks, caching data objects and computational results, and communicating with neighbor nodes. For $i \in \mathcal{V}$, let $\mathcal{N}(i) = \{j \in \mathcal{V} | (i,j) \in \mathcal{E}\}$ denote the neighbors of node i. Let $\mathcal{F}$ be the set of all computations can be performed in $\mathcal{G}$, and $\mathcal{C}$ be the set of data objects available in $\mathcal{G}$, where $\mathcal{F}$ and $\mathcal{C}$ are finite. Network $\mathcal{G}$ is *task-driven*. A task is denoted by a 3-tuple (d, m, k), where $d \in \mathcal{V}$ is the task requester (i.e., the destination to which computation results will be delivered), $m \in \mathcal{F}$ is desired the computation and $k \in \mathcal{C}$ is the required data object. We assume the "computation" specified by m includes user-specified computation configurations. e.g., k may refer to a VR 3D model, and m may refer to rendering it with a specific Point-of-View (PoV). Different PoVs are represented by different m. Such assumption guarantees that the computation result can be reused for multiple tasks if they share the same m and k. Let $\mathcal{T}$ be the set of tasks. To guarantee the data feasibility, for each $k \in \mathcal{C}$, assume there is a non-empty *designated server* set $\mathcal{S}_k \subseteq \mathcal{V}$ that keeps k permanently without consuming cache.

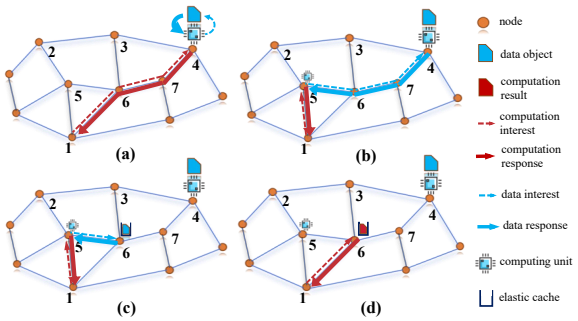

Fig. 1: (a) Cloud computing. (b) Computation offloading without caching [9]. (c) Computation offloading and data caching [6]. (d) Reusing computational results.

For task (d, m, k), we assume d generates *computation interests* (CI) packets for computation m on data k as a stream with rate $r_d(m,k)$ (packet/sec).[1] CI packets are forwarded in $\mathcal{G}$ hop-by-hop. In this paragraph, we first introduce the network behavior without data/result caching. See Fig. 1 (b). Nodes receiving a CI can decide whether to perform the computation locally or to forward it to other nodes. Once a node i decides to perform the computation, it first puts the CI on a pending list, and then generates a corresponding *data interest* (DI) for k. DI packets are also forwarded hop-by-hop until reaching designated servers. Server responds to DI with a *data response* (DR) packet that carries k back to i. Then i can resolve the pending CI by performing computation m on k. After computation, it sends a *computation response* (CR) packet that delivers the computational result to requester d. We assume the CR and DR packets are routed on the same paths as CI and DI but in the reverse direction, respectively.

The above mechanism is boosted by utilizing in-network caches. We assume nodes in $\mathcal{G}$ are equipped with caches of elastic sizes. By paying a size-dependent caching cost, a node can temporarily store data objects and/or computational results. See Fig. 1 (c) for the scenario when data objects are cached, and Fig. 1 (d) for that when computational results are cached. Specifically, upon receiving a CI, a node first checks if the desired computation result is cached. If so, the CR is generated and sent right away without any computation performed. Similarly, upon receiving a DI, the DR is generated and sent if the data object is cached. This paper is the first to model data-centric computing networks with arbitrary topology and reuse of computation results.

[1] We assume the requests of (d, m, k) are identical. For example, node d can represent a wireless access point, receiving identical requests from different devices. The result can be cached at d to resolve these requests.

B. Node-based strategies

We aim to optimize the time-averaged network performance via a node-based formulation. Let $x_i^{\mathrm{d}}(k) \in \{0,1\}$ be the *cache decision* of node i for data k, i.e., $x_i^{\mathrm{d}}(k) = 1$ if i caches k and 0 if not; let $x_i^{\mathrm{c}}(m,k) \in \{0,1\}$ be the cache decision of node i for the computational result of m on data k. Let $\boldsymbol{x} = [x_i^{\mathrm{d}}(k), x_i^{\mathrm{c}}(m,k)]_{i\in\mathcal{V},k\in\mathcal{C},m\in\mathcal{F}}$ be the *global cache decision*.

We assume hop-by-hop multipath routing in $\mathcal{G}$. If $x_i^{\mathrm{c}}(m,k) = 0$, then, of the CI packets received by node i for computation m on data k, a fraction $\psi_{ij}^{\mathrm{c}}(m,k) \in [0,1]$ is forwarded to neighbor node $j \in \mathcal{N}(i)$, and a fraction $\psi_{i0}^{\mathrm{c}}(m,k) \in [0,1]$ is assigned for local computation. If $x_i^{\mathrm{d}}(k) = 0$ and $i \notin \mathcal{S}_k$, then, of the DI packets for data k received by i, a fraction $\psi_{ij}^{\mathrm{d}}(k) \in [0,1]$ is forwarded to neighbor j.[2] We denote by $\boldsymbol{\psi} = [\psi_{ij}^{\mathrm{c}}(m,k), \psi_{ij}^{\mathrm{d}}(k)]_{i,j\in\mathcal{V},k\in\mathcal{C},m\in\mathcal{F}}$ the *global forwarding strategy*. Then, the following flow conservation holds for all $i \in \mathcal{V}$,

$$\begin{aligned} &\sum\nolimits_{j\in\mathcal{V}\cup\{0\}} \psi_{ij}^{\mathrm{c}}(m,k) + x_i^{\mathrm{c}}(m,k) = 1, \quad \forall m \in \mathcal{F}, k \in \mathcal{C}, \\ &\sum\nolimits_{j\in\mathcal{V}} \psi_{ij}^{\mathrm{d}}(k) + x_i^{\mathrm{d}}(k) = \begin{cases} 0, & \text{if } i \in \mathcal{S}_k \\ 1, & \text{if } i \notin \mathcal{S}_k \end{cases} \quad \forall k \in \mathcal{C}. \end{aligned} \tag{1}$$

Conservation (1) implies each CI (DI) packet must be forwarded to one neighbor node (or to local computation for CI packets) if the corresponding result (data) is not stored at the present node. When no loops are formed, it guarantees that all tasks are fulfilled, i.e., the requester will eventually receive a CR for each CI it generates.

Let $t_i^{\mathrm{c}}(m,k)$ be i's time-averaged CI *traffic* (packet/sec) of computation m on data k. It includes both CI generated by i itself and CI forwarded from neighbor nodes to i, namely,

$$t_i^{\mathrm{c}}(m,k) = r_i(m,k) + \sum\nolimits_{j\in\mathcal{N}(i)} \psi_{ji}^{\mathrm{c}}(m,k) t_j^{\mathrm{c}}(m,k). \tag{2a}$$

We denote by $g_i(m,k)$ the average number of computation m performed by node i on data k, then $g_i(m,k) = t_i^{\mathrm{c}}(m,k)\psi_{i0}^{\mathrm{c}}(m,k)$. Let $t_i^{\mathrm{d}}(k)$ be i's traffic of DI for data k. It includes both DI generated by i (due to pending CI), and DI forwarded from other nodes, i.e.,

$$t_i^{d}(k) = \sum\nolimits_{m\in\mathcal{F}} g_i(m,k) + \sum\nolimits_{j\in\mathcal{N}(i)} \psi_{ji}^{\mathrm{d}}(k) t_j^{d}(k). \tag{2b}$$

For a given $\boldsymbol{\psi}$, the uniqueness of $t_i^{c}(m,k)$ and $t_i^{d}(k)$, i.e., the existence of a unique solution to (2), is implied by [10].

C. Problem formulation

We consider nonlinear costs that are incurred on the links due to packet queueing/transmission, and at the nodes due to computation and cache deployment. Let $f_{ij}^{\mathrm{c}}(m,k)$ be the rate (packet/sec) of CI for computation m and data k that travel through (i,j). Let $f_{ij}^{\mathrm{d}}(k)$ be the rate of DI for data k on (i,j). It holds that $f_{ij}^{\mathrm{c}}(m,k) = t_i^{\mathrm{c}}(m,k)\psi_{ij}^{\mathrm{c}}(m,k)$, and $f_{ij}^{\mathrm{d}}(k) = t_i^{\mathrm{d}}(k)\psi_{ij}^{\mathrm{d}}(k)$. In this paper, we assume the size of interest packets (CI, DI) is negligible compared to that of response packets (CR, DR) carrying computational results and data objects, and we only consider the link cost caused by CR and DR. Let L_k^{d} (bit) be the size of data object k, and let L_{mk}^{c} be result size of computation m on k. Since (1) implies every CI/DI on (i,j) must be replied with a corresponding CR/DR on (j,i), the link flow rate (bit/sec) on (i,j) is given by

$$F_{ij} = \sum\nolimits_{m\in\mathcal{F},k\in\mathcal{C}} L_{mk}^{\mathrm{c}} f_{ji}^{\mathrm{c}}(m,k) + \sum\nolimits_{k\in\mathcal{C}} L_k^{\mathrm{d}} f_{ji}^{\mathrm{d}}(k).$$

We denote by $D_{ij}(F_{ij})$ the link cost on (i,j), and assume $D_{ij}(\cdot)$ is continuously differentiable, monotonically increasing and convex, with $D_{ij}(0) = 0$. Such $D_{ij}(\cdot)$ is commonly adopted [10], [11] as it subsumes a variety of existing cost functions, including linear transmission delay and hard link capacity constraints. It also incorporates congestion-dependent metrics. Let μ_{ij} be the service rate of an M/M/1 queue, then $D_{ij}(F_{ij}) = F_{ij}/(\mu_{ij} - F_{ij})$ gives the average number of packets waiting in the queue or being served.

To measure computation costs, let W_{imk} be the workload for node i to perform a single computation request of m on k, the total computational workload at node i is given by

$$G_i = \sum\nolimits_{m\in\mathcal{F},k\in\mathcal{C}} W_{imk} g_i(m,k).$$

The computation cost at i is denoted by $C_i(G_i)$, where $C_i(\cdot)$ is also increasing, continuously differentiable and convex, with $C_i(0) = 0$. Function $C_i(\cdot)$ can represent CPU cycles, computation times, etc. It can also incorporate computation congestion (e.g., average number of packets waiting for available processor or being served at CPU). By Little's Law, when both $D_{ij}(F_{ij})$ and $C_i(G_i)$ represent queue lengths, $\sum_{(i,j)\in\mathcal{E}} D_{ij}(F_{ij}) + \sum_{i\in\mathcal{V}} C_i(G_i)$ is proportional to the expected system latency of computation tasks.

As opposed to the traditional cache capacity limitations [6], we assume the cache has elastic size [12]. A cost is paid for cache deployment. Let X_i be the cache size at i,

$$X_i = \sum\nolimits_{m\in\mathcal{F},k\in\mathcal{C}} L_{mk}^{\mathrm{c}} x_i^{\mathrm{c}}(m,k) + \sum\nolimits_{k\in\mathcal{C}} L_k^{\mathrm{d}} x_i^{\mathrm{d}}(k).$$

We denote by $B_i(X_i)$ the cache deployment cost at node i, where $B_i(\cdot)$ is also continuously differentiable, monotonically increasing and convex. It can represent the expense to buy/rent storage, utility measured by expenses and read/write speed, or approximate hard cache capacity constraints.

We wish to jointly tune $\boldsymbol{x}$ and $\boldsymbol{\psi}$ to minimize the network aggregated cost for transmission, computation, and caching. To construct a continuous relaxation to the mixed-integer NP-hard problem (the NP-hardness for a reduced problem is provided in [13]), we adopt a randomized caching scheme: Suppose $x_i^{\mathrm{c}}(m,k)$ and $x_i^{\mathrm{d}}(k)$ are independent Bernoulli variables. Let ν be the corresponding joint probability distribution over matrices in $\{0,1\}^{|\mathcal{V}|\times|\mathcal{F}|\times|\mathcal{C}|}$ and $\{0,1\}^{|\mathcal{V}|\times|\mathcal{C}|}$, and denote by $\boldsymbol{P}_\nu[\cdot]$, $\mathbb{E}_\nu[\cdot]$ the probability and expectation w.r.t. ν. Let $y_i^{\mathrm{c}}(m,k) \in [0,1]$ be the probability that i caches result for computation m on k, i.e., $y_i^{\mathrm{c}}(m,k) = \boldsymbol{P}_\nu[x_i^{\mathrm{c}}(m,k) = 1] = \mathbb{E}_\nu[x_i^{\mathrm{c}}(m,k)]$. Let $y_i^{\mathrm{d}}(k) = \boldsymbol{P}_\nu[x_i^{\mathrm{d}}(k) = 1] = \mathbb{E}_\nu[x_i^{\mathrm{d}}(k)]$ be the probability that i caches k. Let $\boldsymbol{y} = [y_i^{\mathrm{d}}(k), y_i^{\mathrm{c}}(m,k)]_{i\in\mathcal{V},k\in\mathcal{C},m\in\mathcal{F}}$ be the *global caching strategy*.

When the network adopts randomized caching strategy, without ambiguity, in the rest of the paper, we use F_{ij} to denote $F_{ij}|_{\boldsymbol{y}} = \mathbb{E}_\nu[F_{ij}]$ and let $Y_i = \sum_{k\in\mathcal{C}} y_i(k)$ denote

[2] $\psi_{ij}^{\mathrm{c}}(m,k) = \psi_{ij}^{\mathrm{d}}(k) = 0$ if $(i,j) \notin \mathcal{E}$; $\psi_{ij}^{\mathrm{d}}(k) = 0$ for all j if $i \in \mathcal{S}_k$.

the expected cache size. We also denote the corresponding forwarding strategy by $\boldsymbol{\phi}$, satisfying conservation [3]

$$\sum\nolimits_{j\in\{0\}\cup\mathcal{V}} \phi^{\mathrm{c}}_{ij}(m,k) + y^{\mathrm{c}}_i(m,k) = 1, \quad \forall m\in\mathcal{F}, k\in\mathcal{C},$$
$$\sum\nolimits_{j\in\mathcal{V}} \phi^{\mathrm{d}}_{ij}(k) + y^{\mathrm{d}}_i(k) = \begin{cases} 0, & \text{if } i\in\mathcal{S}_k \\ 1, & \text{if } i\notin\mathcal{S}_k \end{cases} \quad \forall k\in\mathcal{C}. \tag{3}$$

Then, we cast the aggregated cost minimization problem as

$$\min_{\boldsymbol{y},\boldsymbol{\phi}} T = \sum_{(i,j)\in\mathcal{E}} D_{ij}(F_{ij}) + \sum_{i\in\mathcal{V}} B_i(X_i) + \sum_{i\in\mathcal{V}} C_i(G_i) \tag{4}$$
$$\text{subject to} \quad \mathbf{0}\le\boldsymbol{\phi}\le\mathbf{1}, \quad \mathbf{0}\le\boldsymbol{y}\le\mathbf{1}, \text{ and (3) holds.}$$

We do not impose constraints for link, computation, or cache capacities in (4), as they can be incorporated in the convex cost functions. We present two distributed methods to tackle (4): an offline method in Section III and an online adaptive method in Section IV. For notation brevity, in the rest of the paper, we omit the mark for computation type (m,k) (or data k) when the context is without ambiguity, and denote $\phi^{\mathrm{c}}_{ij}(m,k)$, $\phi^{\mathrm{d}}_{ij}(k)$, $y^{\mathrm{c}}_i(m,k)$, $y^{\mathrm{d}}_i(k)$, $t^{\mathrm{c}}_i(m,k)$, $t^{\mathrm{d}}_i(k)$, by ϕ^{c}_{ij}, ϕ^{d}_{ij}, y^{c}_i, y^{d}_i, t^{c}_i, t^{d}_i, respectively.

III. Offline Method: 1/2 Approximation

In this section, we present an offline method for (4). It develops Algorithm 1 of [12] to consider multipath forwarding, in-network computation, and reuse of computational results.

Notice that (3) uniquely determines $\boldsymbol{y}$ if $\boldsymbol{\phi}$ is given. Let $\mathcal{D}_{\boldsymbol{\phi}}$ be the feasible set of $\boldsymbol{\phi}$ satisfies the following,

$$\sum\nolimits_{j\in\{0\}\cup\mathcal{V}} \phi^{\mathrm{c}}_{ij} \le 1,\ \forall(m,k);\ \sum\nolimits_{j\in\mathcal{V}} \phi^{\mathrm{d}}_{ij} \begin{cases} = 0, \text{ if } i\in\mathcal{S}_k \\ \le 1, \text{ if } i\notin\mathcal{S}_k \end{cases}, \forall k.$$

We denote by $\boldsymbol{y}(\boldsymbol{\phi})$ the corresponding $\boldsymbol{y}$ for $\boldsymbol{\phi}\in\mathcal{D}_{\boldsymbol{\phi}}$, and rewrite (4) as the following *gain maximization* problem,

$$\max_{\boldsymbol{\phi}\in\mathcal{D}_{\boldsymbol{\phi}}} \quad G(\boldsymbol{\phi}) = T_0 - T(\boldsymbol{y}(\boldsymbol{\phi}),\boldsymbol{\phi}), \tag{5}$$

where $G(\boldsymbol{\phi})$ denotes the caching-offloading gain, and T_0 is a constant upper bound given by (we assume T_0 is finite)

$$T_0 = \max_{\boldsymbol{\phi}\in\mathcal{D}_{\boldsymbol{\phi}}:\boldsymbol{y}(\boldsymbol{\phi})=\mathbf{0}} T(\mathbf{0},\boldsymbol{\phi}), \tag{6}$$

We next provide a $1/2$ approximation algorithm to (5) by characterizing and exploiting its mathematical structure.

A. "Submodular + concave" reformulation

We say there is a *CI path* $p=[p_1,p_2,\cdots,p_{|p|}]$ from node i to node j for (m,k) if $p_1=i$, $p_{|p|}=j$, and for any $t=1,\cdots,|p|-1$, it holds that $(p_t,p_{t+1})\in\mathcal{E}$ and $\phi^{\mathrm{c}}_{p_tp_{t+1}}(m,k)>0$. Similarly, we say p is a *DI path* for data k if $\phi^{\mathrm{d}}_{p_tp_{t+1}}(k)>0$ for $t=1,\cdots,|p|-1$. Let $\mathcal{P}^{\mathrm{c}}_{ij}(m,k)$ be the set of all CI paths from node i to node j for (m,k), and let $\mathcal{P}^{\mathrm{d}}_{ij}(k)$ be the set of all DI paths from i to j for k.[4] The rate of CI for (m,k) generated by node v and arriving at i is $r_v(m,k)\sum_{p\in\mathcal{P}^{\mathrm{c}}_{vi}(m,k)}\prod_{t=1}^{|p|-1}\phi^{\mathrm{c}}_{p_tp_{t+1}}$, thus

$$f^{\mathrm{c}}_{ij}(m,k) = \phi^{\mathrm{c}}_{ij}\sum_{v\in\mathcal{V}} r_v(m,k)\sum_{p\in\mathcal{P}^{\mathrm{c}}_{vi}(m,k)}\prod_{t=1}^{|p|-1}\phi^{\mathrm{c}}_{p_tp_{t+1}},$$
$$g_i(m,k) = \phi^{\mathrm{c}}_{i0}\sum_{v\in\mathcal{V}} r_v(m,k)\sum_{p\in\mathcal{P}^{\mathrm{c}}_{vi}(m,k)}\prod_{t=1}^{|p|-1}\phi^{\mathrm{c}}_{p_tp_{t+1}}. \tag{7a}$$

The rate of DI packets for k generated by node v and arriving node i is $(\sum_m g_v(m,k))\sum_{p\in\mathcal{P}^{\mathrm{d}}_{vi}(k)}\prod_{t=1}^{|p|-1}\phi^{\mathrm{d}}_{p_tp_{t+1}}$, and

$$f^{\mathrm{d}}_{ij}(k) = \phi^{\mathrm{d}}_{ij}\sum_{v\in\mathcal{V}}\left(\sum_{m\in\mathcal{F}} g_v(m,k)\right)\sum_{p\in\mathcal{P}^{\mathrm{d}}_{vi}(k)}\prod_{t=1}^{|p|-1}\phi^{\mathrm{d}}_{p_tp_{t+1}}. \tag{7b}$$

We rewrite the gain $G(\boldsymbol{\phi})$ as $G(\boldsymbol{\phi})=M(\boldsymbol{\phi})+N(\boldsymbol{\phi})$, where

$$M(\boldsymbol{\phi}) = T_0 - \sum_{(i,j)\in\mathcal{E}} D_{ij}(F_{ij}) - \sum_{i\in\mathcal{V}} C_i(G_i),\ N(\boldsymbol{\phi}) = -\sum_{i\in\mathcal{V}} B_i(Y_i).$$

Lemma 1. *Problem (5) is a "submodular + concave" maximization problem. Specifically, $M(\boldsymbol{\phi})$ is non-negative monotonic DR-submodular*[5]

B. Constant factor approximation

Maximization of "submodular + concave" functions was first systematically studied by [14]. Problem (5) falls into one of the categories provided [14], to which a *Gradient-Combining Frank-Wolfe* (GCFW) algorithm (present in Algorithm 1) guarantees a constant factor approximation of $1/2$.

Algorithm 1: LOAM-GCFW

Input: Integer $N>1$
Let $\varepsilon=N^{-\frac{1}{3}}$, $n=0$, $\boldsymbol{\phi}^{(0)}\in\mathcal{D}_{\boldsymbol{\phi}}$ with $\boldsymbol{y}(\boldsymbol{\phi}^{(0)})=\mathbf{0}$.
do
  Let $\boldsymbol{h}=\arg\max_{\boldsymbol{\phi}\in\mathcal{D}_{\boldsymbol{\phi}}}\left\langle\boldsymbol{\phi},\nabla M(\boldsymbol{\phi}^{(n)})+2\nabla N(\boldsymbol{\phi}^{(n)})\right\rangle$.
  Let $\boldsymbol{\phi}^{(n+1)}=(1-\varepsilon^2)\boldsymbol{\phi}^{(n)}+\varepsilon^2\boldsymbol{h}$.
for $n=0,1,\cdots,N-1$;
Let $\boldsymbol{\phi}^{\text{out}}=\arg\max_{\boldsymbol{\phi}\in\{\boldsymbol{\phi}^{(0)},\cdots,\boldsymbol{\phi}^{(N)}\}} G(\boldsymbol{\phi})$.

Theorem 1 (Theorem 3.10 [14])**.** *Assume $G(\boldsymbol{\phi})$ is L-smooth, i.e., ∇G is Lipschitz continuous. For $N>1$, let $\boldsymbol{\phi}^{out}$ be the result of Algorithm 1, $\boldsymbol{\phi}^*$ be optimal solution to (5), then*

$$G(\boldsymbol{\phi}^{out}) \ge \frac{1-\varepsilon}{2}M(\boldsymbol{\phi}^*) + N(\boldsymbol{\phi}^*) - \varepsilon\cdot O\left(L|\mathcal{V}|^2|\mathcal{F}||\mathcal{C}|\right).$$

where L is the Lipschitz constant of ∇G.

By (3), $\nabla N(\boldsymbol{\phi})$ in Algorithm 1 can be calculated as

$$\frac{\partial N(\boldsymbol{\phi})}{\partial\phi^{\mathrm{c}}_{ij}} = -L^{\mathrm{c}}_{mk}B_i'(Y_i), \quad \frac{\partial N(\boldsymbol{\phi})}{\partial\phi^{\mathrm{d}}_{ij}} = -L^{\mathrm{d}}_{k}B_i'(Y_i).$$

Calculating $\nabla M(\boldsymbol{\phi})$ follows $\frac{\partial M(\boldsymbol{\phi})}{\partial\phi^{\mathrm{c}}_{ij}} = -\frac{\partial T}{\partial\phi^{\mathrm{c}}_{ij}} - \frac{\partial N(\boldsymbol{\phi})}{\partial\phi^{\mathrm{c}}_{ij}}$, and a distributed recursive calculation of $\frac{\partial T}{\partial\phi^{\mathrm{c}}_{ij}}$ is introduced in

[3] For example, suppose the network partitions time into slots and use $(\boldsymbol{x}^t,\boldsymbol{\psi}^t)$ in slot t with $\mathbb{E}_\nu[\boldsymbol{x}^t]=\boldsymbol{y}^t$. Although $(\boldsymbol{x}^t,\boldsymbol{\psi}^t)$ satisfies conservation (1) for every t, when optimizing the time-averaged forwarding strategy $\boldsymbol{\phi}=\lim_{T\to\infty}\frac{1}{T}\sum_{t=1}^{T}\boldsymbol{\psi}^t$, conservation (3) holds for $(\boldsymbol{y},\boldsymbol{\phi})$. It is empirically demonstrated in [12] that such randomized caching and forwarding scheme $(\boldsymbol{y},\boldsymbol{\phi})$ can efficiently approximate real network costs using $(\boldsymbol{x}^t,\boldsymbol{\psi}^t)$.

[4] We assume all CI and DI paths are loop-free.

[5] DR-submodular function is a continuous generalization of submodular functions with diminishing return.

Section IV-A. The linear programming in Algorithm 1 can also be implemented in a distributed manner[6]: for any m and k, node i sets $h^{\mathrm{c}}_{ij}(m,k) = 0$ for all $j \in \{0\} \cup \mathcal{V}$ if $\frac{\partial M(\boldsymbol{\phi})}{\partial \phi^{\mathrm{c}}_{ij}} + 2\frac{\partial N(\boldsymbol{\phi})}{\partial \phi^{\mathrm{c}}_{ij}} < 0$ for all j; otherwise, set $h^{\mathrm{c}}_{ij}(m,k) = 1$ for $j = \arg\max_{v\in\{0\}\cup\mathcal{V}} \frac{\partial M(\boldsymbol{\phi})}{\partial \phi^{\mathrm{c}}_{iv}} + 2\frac{\partial N(\boldsymbol{\phi})}{\partial \phi^{\mathrm{c}}_{iv}}$. For practical simplicity, the initial state $\boldsymbol{\phi}^{(0)}$ can be set to the *shortest-extended-path* scheme with no caching, detailed in Section V. Algorithm 1 provides the first known constant factor approximation to (4).

IV. Online Adaptive Solution

In practical scenarios, user request patterns $r_i(m,k)$ and network status are both not known prior, and can be time-varying, e.g., link cost $D_{ij}(\cdot)$ can be affected by the temporary link quality. The network operator may seek a method with stronger practical feasibility to overcome these limitations. This section presents an online, distributed, and adaptive algorithm to (4), with minimum prior knowledge requirement. We first present a KKT necessary optimality condition for (4), then give a modification to the condition that yields a bounded gap from the global optimum.

A. Closed-form marginals

We start by giving closed-form partial derivatives of $T(\boldsymbol{y},\boldsymbol{\phi})$. Our analysis makes a non-trivial generalization of [12] to cache-enabled computing networks. For $\boldsymbol{y}$,

$$\frac{\partial T}{\partial y^{\mathrm{c}}_i} = L^{\mathrm{c}}_{mk} B'_i(Y_i), \quad \frac{\partial T}{\partial y^{\mathrm{d}}_i} = L^{\mathrm{d}}_{k} B'_i(Y_i). \tag{8}$$

For any (m,k), the marginal cost due to increase of ϕ^{c}_{ij} when $j \neq 0$ consists of two parts: (1) the marginal cost due to increase of F_{ji} since more CR packets are sent through link (j,i), and (2) the marginal cost due to increase of t^{c}_j since node j needs to handle more CI packets. Both parts scale with node i's traffic t^{c}_i. Formally, for all $j \in \mathcal{N}(i)$,

$$\frac{\partial T}{\partial \phi^{\mathrm{c}}_{ij}} = t^{\mathrm{c}}_i \left(L^{\mathrm{c}}_{mk} D'_{ji}(F_{ji}) + \frac{\partial T}{\partial t^{\mathrm{c}}_j} \right). \tag{9a}$$

For $j = 0$, marginal cost $\partial T/\partial \phi^{\mathrm{c}}_{i0}$ consists of the marginal cost due to increased workload G_i and due to more DI packets generation that counts into $t^{\mathrm{d}}_i(k)$, i.e.,

$$\frac{\partial T}{\partial \phi^{\mathrm{c}}_{i0}} = t^{\mathrm{c}}_i \left(W_{imk} C'_i(G_i) + \frac{\partial T}{\partial t^{\mathrm{d}}_i(k)} \right). \tag{9b}$$

In (9), term $\partial T/\partial t^{\mathrm{c}}_i$ is the marginal cost for i to handle unit rate increment of CI packets of (m,k). It is a weighted sum of marginals on out-links and local CPU,

$$\begin{aligned} \frac{\partial T}{\partial t^{\mathrm{c}}_i} = &\sum_{j\in\mathcal{N}(i)} \phi^{\mathrm{c}}_{ij} \left(L^{\mathrm{c}}_{mk} D'_{ji}(F_{ji}) + \frac{\partial T}{\partial t^{\mathrm{c}}_j} \right) \\ &+ \phi^{\mathrm{c}}_{i0} \left(W_{imk} C'_i(G_i) + \frac{\partial T}{\partial t^{\mathrm{d}}_i(k)} \right). \end{aligned} \tag{10}$$

Similar reasoning applies to the marginal costs for DI forwarding strategy $\phi^{\mathrm{d}}_{ij}(k)$. Specifically, it holds that

$$\frac{\partial T}{\partial \phi^{\mathrm{d}}_{ij}} = t^{\mathrm{d}}_i \left(L^{\mathrm{d}}_{k} D'_{ji}(F_{ji}) + \frac{\partial T}{\partial t^{\mathrm{d}}_j} \right), \quad \forall j \in \mathcal{N}(i), \tag{11}$$

[6]Here we only present $h^{\mathrm{c}}_{ij}(m,k)$. Similar calculation applies to $h^{\mathrm{d}}_{ij}(k)$.

where $\partial T/\partial t^{\mathrm{d}}_i$ is the marginal cost for i to handle unit rate increment of DI packets for data k, given by

$$\frac{\partial T}{\partial t^{\mathrm{d}}_i} = \sum_{j\in\mathcal{N}(i)} \phi^{\mathrm{d}}_{ij} \left(L^{\mathrm{d}}_{k} D'_{ji}(F_{ji}) + \frac{\partial T}{\partial t^{\mathrm{d}}_j} \right). \tag{12}$$

One could calculate $\partial T/\partial t^{\mathrm{c}}_i$ and $\partial T/\partial t^{\mathrm{d}}_i$ recursively by (10) and (12), respectively. The recursive calculation starts at designated servers and is guaranteed to terminate in finite steps if no loops are formed. This can be applied to achieve distributed calculation for $\nabla M(\boldsymbol{\phi})$ in Algorithm 1.

B. KKT condition and modification

Based on the closed-form marginals, Lemma 2 provides KKT necessary conditions of (4).

Lemma 2. *Let $(\boldsymbol{y},\boldsymbol{\phi})$ be an optimal solution to problem (4), then for all $i,j \in \mathcal{V}$, $m \in \mathcal{F}$, and $k \in \mathcal{C}$,*

$$\begin{aligned} &t^c_i \left(L^c_{mk} D'_{ji}(F_{ji}) + \frac{\partial T}{\partial t^c_j} \right) \begin{cases} = \lambda^c_{imk}, \text{ if } \phi^c_{ij} > 0 \\ \geq \lambda^c_{imk}, \text{ if } \phi^c_{ij} = 0 \end{cases} \\ &t^c_i \left(W_{imk} C'_i(G_i) + \frac{\partial T}{\partial t^d_i(k)} \right) \begin{cases} = \lambda^c_{imk}, \text{ if } \phi^c_{i0} > 0 \\ \geq \lambda^c_{imk}, \text{ if } \phi^c_{i0} = 0 \end{cases} \\ &t^d_i \left(L^d_k D'_{ji}(F_{ji}) + \frac{\partial T}{\partial t^d_j} \right) \begin{cases} = \lambda^d_{ik}, \text{ if } \phi^d_{ij} > 0 \\ \geq \lambda^d_{ik}, \text{ if } \phi^d_{ij} = 0 \end{cases} \\ &B'_i(Y_i) \begin{cases} = \frac{\lambda^c_{imk}}{L^c_{mk}}, \text{ if } y^c_i > 0 \\ \geq \frac{\lambda^c_{imk}}{L^c_{mk}}, \text{ if } y^c_i = 0 \end{cases} \quad B'_i(Y_i) \begin{cases} = \frac{\lambda^d_{ik}}{L^d_k}, \text{ if } y^d_i > 0 \\ \geq \frac{\lambda^d_{ik}}{L^d_k}, \text{ if } y^d_i = 0 \end{cases} \end{aligned} \tag{13}$$

where λ^c_{imk} and λ^d_{ik} are given by

$$\lambda^c_{imk} = \min\left\{ \frac{\partial T}{\partial y^c_i}, \min_{j\in\{0\}\cup\mathcal{V}} \frac{\partial T}{\partial \phi^c_{ij}} \right\}, \lambda^d_{ik} = \min\left\{ \frac{\partial T}{\partial y^d_i}, \min_{j\in\mathcal{V}} \frac{\partial T}{\partial \phi^d_{ij}} \right\}.$$

Condition (13) is not sufficient for global optimality, a counterexample is provided in [10]. Specifically, consider the degenerate case $t^{\mathrm{c}}_i = t^{\mathrm{d}}_i = 0$, (13) always holds for arbitrary $\boldsymbol{\phi}$ at i, as long as $y^{\mathrm{c}}_i = y^{\mathrm{d}}_i = 0$. Inspired by [10], we propose a modification to (13) that resolves such degenerate cases. Notice that in (13), t^{c}_i and t^{d}_i appear repeatedly in conditions regarding $\boldsymbol{\phi}$ for all j. We divide these conditions by t^{c}_i or t^{d}_i, respectively. Such modification leads to a bounded gap from the global optimum.

Theorem 2. *Let $(\boldsymbol{y},\boldsymbol{\phi})$ be feasible to (4). Assume for all $i \in \mathcal{V}$, $m \in \mathcal{F}$, $k \in \mathcal{C}$, condition (14) holds for $(\boldsymbol{y},\boldsymbol{\phi})$,*

$$\begin{aligned} &\delta^c_{ij}(m,k) \begin{cases} = \delta^c_{imk}, \text{ if } \phi^c_{ij} > 0 \\ \geq \delta^c_{imk}, \text{ if } \phi^c_{ij} = 0 \end{cases}, \forall j \in \{0\} \cup \mathcal{V} \\ &\delta^d_{ij}(k) \begin{cases} = \delta^d_{ik}, \text{ if } \phi^d_{ij} > 0 \\ \geq \delta^d_{ik}, \text{ if } \phi^d_{ij} = 0 \end{cases}, \forall j \in \mathcal{V} \\ &\gamma^c_i(m,k) \begin{cases} = \delta^c_{imk}, \text{ if } y^c_i > 0 \\ \geq \delta^c_{imk}, \text{ if } y^c_i = 0 \end{cases} \quad \gamma^d_i(k) \begin{cases} = \delta^d_{ik}, \text{ if } y^d_i > 0 \\ \geq \delta^d_{ik}, \text{ if } y^d_i = 0 \end{cases} \end{aligned} \tag{14}$$

where $\delta^{\mathrm{c}}_{ij}(m,k)$, $\delta^{d}_{ij}(k)$, $\gamma^{\mathrm{c}}_{i}(m,k)$ and $\gamma^{d}_{i}(k)$ are "modified marginals", i.e., derivatives of T w.r.t. t^{c}_i and t^{d}_i, namely, [7]

$$\delta^{\mathrm{c}}_{ij}(m,k) = \begin{cases} L^{\mathrm{c}}_{mk} D'_{ji}(F_{ji}) + \frac{\partial T}{\partial t^{\mathrm{c}}_j(m,k)}, \text{ if } j \neq 0 \\ W_{imk} C'_i(G_i) + \frac{\partial T}{\partial t^{d}_i(k)}, \text{ if } j = 0 \end{cases} \tag{15a}$$

$$\delta^{d}_{ij}(k) = L^{d}_{k} D'_{ji}(F_{ji}) + \frac{\partial T}{\partial t^{d}_j(k)} \tag{15b}$$

$$\gamma^{\mathrm{c}}_{i}(m,k) = \frac{L^{\mathrm{c}}_{mk} B'_i(Y_i)}{t^{\mathrm{c}}_i(m,k)}, \quad \gamma^{d}_{i}(k) = \frac{L^{d}_{k} B'_i(Y_i)}{t^{d}_i(k)}, \tag{15c}$$

and $\delta^{\mathrm{c}}_{imk}$, δ^{d}_{ik} are the minimum modified marginals given by

$$\delta^{\mathrm{c}}_{imk} = \min\left\{ \gamma^{\mathrm{c}}_{i}(m,k), \min_{j \in \{0\} \cup \mathcal{V}} \delta^{\mathrm{c}}_{ij}(m,k) \right\}, \tag{16a}$$

$$\delta^{d}_{ik} = \min\left\{ \gamma^{d}_{i}(k), \min_{j \in \mathcal{V}} \delta^{d}_{ij}(k) \right\}. \tag{16b}$$

Let $(\boldsymbol{y}^{\dagger}, \boldsymbol{\phi}^{\dagger})$ be any feasible solution to (4). Then,

$$\begin{aligned} T(\boldsymbol{y}^{\dagger}, \boldsymbol{\phi}^{\dagger}) - T(\boldsymbol{y}, \boldsymbol{\phi}) \geq & \sum_{i,k} \delta^{d}_{ik} \left(y^{d}_i - y^{d\dagger}_i \right) \left(t^{d\dagger}_i - t^{d}_i \right) \\ & + \sum_{i,m,k} \delta^{\mathrm{c}}_{imk} \left(y^{\mathrm{c}}_i - y^{\mathrm{c}\dagger}_i \right) \left(t^{\mathrm{c}\dagger}_i - t^{\mathrm{c}}_i \right). \end{aligned} \tag{17}$$

Different from the constant factor approximation in Theorem 1, (17) provides an additive bound for total cost T when the modified condition (14) holds. To provide an intuitive interpretation of (14), recall (9), $t^{\mathrm{c}}_i = \partial f^{\mathrm{c}}_{ij}(m,k)/\partial \phi^{\mathrm{c}}_{ij}$. The modified marginal for $j \in \mathcal{N}(i)$ in (15a) can be written as

$$\delta^{\mathrm{c}}_{ij}(m,k) = \frac{\partial T}{\partial \phi^{\mathrm{c}}_{ij}} \Big/ \frac{\partial f^{\mathrm{c}}_{ij}(m,k)}{\partial \phi^{\mathrm{c}}_{ij}} = \frac{\partial T}{\partial f^{\mathrm{c}}_{ij}(m,k)}$$

i.e., the marginal cost if node i forwards additional CI packets of unit rate to node j. Similarly, $\delta^{\mathrm{c}}_{i0}(m,k)$ and $\delta^{\mathrm{d}}_{ij}(k)$ satisfies

$$\delta^{\mathrm{c}}_{i0}(m,k) = \frac{\partial T}{\partial g_i(m,k)}, \quad \delta^{\mathrm{d}}_{ij}(k) = \frac{\partial T}{\partial f^{\mathrm{d}}_{ij}(k)},$$

namely, the marginal cost with respect to computational input at node i, and the marginal cost due to additional DI packets of unit rate forwarded to node j, respectively.

On the other hand, we define virtual *cached flows* as $h^{\mathrm{c}}_i(m,k) = t^{\mathrm{c}}_i y^{\mathrm{c}}_i$ and $h^{\mathrm{d}}_i(k) = t^{\mathrm{d}}_i y^{\mathrm{d}}_i$, i.e., the rate of CI/DI that terminate at i due to i's caching strategy. Then

$$\gamma^{\mathrm{c}}_{i}(m,k) = \frac{\partial T}{\partial h^{\mathrm{c}}_i(m,k)}, \quad \gamma^{\mathrm{d}}_{i}(k) = \frac{\partial T}{\partial h^{\mathrm{d}}_i(k)} \tag{18}$$

gives the marginal cache deployment cost for i to resolve unit rate of CI/DI packets by expanding its cache, respectively.

Therefore, (14) implies that each node independently handles incremental arrival CI/DI packets in the way that achieves the minimum modified marginal – either by forwarding to neighbors/local computation, or by expanding its own cache. Based on Theorem 2, we developed corollaries to better illustrate the practical implication of bound (17). Due to space limitation, we present them in [8].

[7] In (15a) (15b), we assume $\delta^{\mathrm{c}}_{ij}(m,k) = \delta^{\mathrm{d}}_{ij}(k) = 0$ if $j \notin \mathcal{N}(i)$. In (15c), we assume $\gamma^{\mathrm{c}}_{i}(m,k) = \infty$ if $t^{\mathrm{c}}_i(m,k) = 0$, and $\gamma^{\mathrm{d}}_{i}(k) = \infty$ if $t^{\mathrm{d}}_i(k) = 0$.

Algorithm 2: Gradient Projection (LOAM-GP)

Input: Initial loop-free $(\boldsymbol{y}^{(0)}, \boldsymbol{\phi}^{(0)})$, stepsize α

do

- Update $\partial T/\partial t^{\mathrm{c}}_i$ and $\partial T/\partial t^{\mathrm{d}}_i$ by Recursive Marginal Calculation.
- Updates $\boldsymbol{y}^{(t)}$, $\boldsymbol{\phi}^{(t)}$ by (19) and (20).
- Round continuous $\boldsymbol{y}^{(t+1)}$ to discrete $\boldsymbol{x}^{(t+1)}$.

at *end of t-th slot*;

C. Online adaptive algorithm

Algorithm overview. The proposed algorithm (summarized in Algorithm 2) is a gradient projection variant inspired by [12]. It converges to condition (14), does not require prior knowledge of $r_i(m,k)$ and $\mathcal{S}_k$, and is adaptive to moderate changes in $r_i(m,k)$ and cost functions.

We partition time into *slots* of duration T_{slot}, and assume the network starts at a feasible $(\boldsymbol{y}^{(0)}, \boldsymbol{\phi}^{(0)})$ with $T^{(0)} < \infty$. At the end of each slot, strategies are updated as

$$\boldsymbol{\phi}^{(t+1)} = \boldsymbol{\phi}^{(t)} + \Delta\boldsymbol{\phi}^{(t)}, \quad \boldsymbol{y}^{(t+1)} = \boldsymbol{y}^{(t)} + \Delta\boldsymbol{y}^{(t)}, \tag{19}$$

with distributed calculation for $\Delta\boldsymbol{\phi}^{(t)}$ and $\Delta\boldsymbol{y}^{(t)}$,

$$\Delta\phi^{\mathrm{c}(t)}_{ij} = \begin{cases} -\phi^{\mathrm{c}(t)}_{ij}, & \text{if } j \in \mathcal{B}^{\mathrm{c}}_i(m,k)^{(t)} \\ -\min\left\{ \phi^{\mathrm{c}(t)}_{ij}, \alpha e^{\mathrm{c}}_{ij}(m,k)^{(t)} \right\}, & \text{if } e^{\mathrm{c}(t)}_{ij} > 0 \\ S^{\mathrm{c}}_i(m,k)^{(t)}, & \text{for one } j \text{ such that } e^{\mathrm{c}(t)}_{ij} = 0 \end{cases}$$

$$\Delta y^{\mathrm{c}(t)}_i = \begin{cases} -\min\left\{ y^{\mathrm{c}(t)}_i, \alpha e^{\mathrm{c}(t)}_{i,\mathrm{y}} \right\}, & \text{if } e^{\mathrm{c}(t)}_{i,\mathrm{y}} > 0 \\ S^{\mathrm{c}}_i(m,k)^{(t)}, & \text{if } e^{\mathrm{c}(t)}_{i,\mathrm{y}} = 0, \text{ all } \Delta\phi^{\mathrm{c}(t)}_{ij} \leq 0 \end{cases} \tag{20}$$

where $\mathcal{B}^{\mathrm{c}}_i(m,k)$ is the set of *blocked nodes* to prevent routing loops, $\alpha > 0$ is the stepsize, and

$$e^{\mathrm{c}}_{ij}(m,k) = \delta^{\mathrm{c}}_{ij}(m,k) - \delta^{\mathrm{c}}_{imk}, \; e^{\mathrm{c}}_{i,\mathrm{y}}(m,k) = \gamma^{\mathrm{c}}_{i}(m,k) - \delta^{\mathrm{c}}_{imk},$$
$$S^{\mathrm{c}}_i(m,k) = -\sum\nolimits_{j:\Delta\phi^{\mathrm{c}}_{ij} \leq 0} \Delta\phi^{\mathrm{c}}_{ij} + \Delta y^{\mathrm{c}}_i.$$

The DI strategies, i.e., $\Delta\phi^{\mathrm{d}(t)}_{ij}$ and $\Delta y^{\mathrm{d}(t)}_i$, are updated with a same manner and omitted due to space limitation. The intuitive ideas of (20) are: (i) do not forward any request to blocked nodes, (ii) if a modified marginal ($\delta^{\mathrm{c}}_{ij}(m,k)$ or $\gamma^{\mathrm{c}}_{i}(m,k)$) is not the minimum, shrink the corresponding forwarding/caching fractions, and (iii) add the shrunk fractions to the minimum marginal direction.

The blocked node sets $\mathcal{B}^{\mathrm{c}}_i(m,k)$ and $\mathcal{B}^{\mathrm{d}}_i(k)$ ensures no loops (in both CI paths and DI paths) are formed throughout the algorithm, provided the initial state $(\boldsymbol{y}^{(0)}, \boldsymbol{\phi}^{(0)})$ is loop-free. Discussion and algorithms to identify blocked node sets are provided in [12] and omitted here for brevity. After updating (20), We use *Distributed Randomized Rounding* presented in [12] to round continuous $\boldsymbol{y}^{(t+1)}$ to binary $\boldsymbol{x}^{(t+1)}$, guaranteeing that expected flow rates, computation workloads and cache sizes equal the relaxed value.

Recursive Marginal Calculation. Recall (15), besides the status of local link/CPU/storage captured by $D'_{ij}(F_{ij})$, $C'_i(G_i)$ and $B'_i(Y_i)$, the calculation in (20) requires the knowledge of

$\partial T/\partial t_i^{\mathrm{c}}$ and $\partial T/\partial t_i^{\mathrm{d}}$, which have a global dependence and can be recursively calculated by (10)(12). We utilize the 2-stage message-broadcasting procedure described in [9] to carry out the recursive calculation. The procedure can be used both for (20) in the online adaptive Algorithm 2, and for $\nabla M(\boldsymbol{\phi})$ in the offline Algorithm 1. The underlying idea is that: recursive calculation (12) start from i with $i \in \mathcal{S}_k$ or $y_i^{\mathrm{d}} = 1$ such that $\partial T/\partial t_i^{\mathrm{d}} = 0$, and a node j calculates $\partial T/\partial t_j^{\mathrm{d}}$ after obtaining all $\partial T/\partial t_i^{\mathrm{d}}$ from all downstream neighbors i with $\phi_{ji}^{\mathrm{d}} > 0$; calculation (10) starts after all $\partial T/\partial t_i^{\mathrm{d}}$ are obtained, and starts at nodes with $y_i^{\mathrm{c}} = 1$ or $\phi_{i0}^{\mathrm{c}} = 1$. If no loops are formed, this broadcast is guaranteed to traverse throughout the network and terminate within a finite number of steps.

Convergence and complexity. Algorithm 2 can be implemented online as it does not require prior knowledge of $r_i(m,k)$, and adapts to changes in $r_i(m,k)$ since during the recursive calculation, $D'_{ij}(F_{ij})$ and $C'_i(G_i)$ can be directly estimated by the packet number on links and CPU. It can also adapt network topology change: whenever a link (i,j) is removed from $\mathcal{E}$, node i only needs to add j to the blocked node set; when link (i,j) is added to $\mathcal{E}$, node i removes j from the blocked node set.[8]

Theorem 3. *Assume $(\boldsymbol{y}^{(0)}, \boldsymbol{\phi}^{(0)})$ is loop-free with $T^0 < \infty$, and $(\boldsymbol{y}^{(t)}, \boldsymbol{\phi}^{(t)})$ is updated by Algorithm 2 with sufficiently small α. Then, sequence $\left\{(\boldsymbol{y}^{(t)}, \boldsymbol{\phi}^{(t)})\right\}_{t=0}^{\infty}$ converges to a limit point $(\boldsymbol{y}^*, \boldsymbol{\phi}^*)$ satisfying (14) with loop-free guaranteed.*

Theorem 3 is a straightforward extension of [10] Theorem 5. The convergence property can be further improved by adopting second-order methods [11]. The Recursive Marginal Calculation introduces $(|\mathcal{C}| + |\mathcal{T}|)\,|\mathcal{E}|$ broadcast messages per slot, with on average amount $(|\mathcal{C}| + |\mathcal{T}|)\,/T_{\text{slot}}$ per link/second. We assume the broadcast messages are sent in an out-of-band channel. Let t_c be the broadcast message transmission time and $\bar{h}$ be the maximum length of extended paths, the broadcast completion time is at most $\bar{h}t_c$. Algorithm 2 has space complexity $O\left(|\mathcal{C}| + |\mathcal{T}|\right)$ at each node. The update may fail if broadcast completion time exceeds T, or if $(|\mathcal{C}| + |\mathcal{T}|)\,/T_{\text{slot}}$ exceeds the broadcast channel capacity. If so, we can use longer slots, or allow some nodes to update every multiple slots. If $\mathcal{C}$ or $\mathcal{F}$ is too large or infinite, the algorithm should only be applied to the most popular data/computations. The rest requests can be handled through other simple methods.

V. Simulation

Simulator. We build a publicly available packet-level network simulator. For tasks in $\mathcal{T}$, d is uniformly chosen in $\mathcal{V}$, m and k are chosen in the $\mathcal{F}$, $\mathcal{C}$ with a Zipf-distribution of parameter 1.0, respectively. Rates $r_d(m,k)$ are uniformly random in $[1.0, 5.0]$, and CI packets are generated with Poisson processes. For each k, we assume $|\mathcal{S}_k| = 1$ and choose the designated server uniformly in $\mathcal{V}$. We monitor the network status with interval $T_{\text{monitor}} = 10$. We use congestion-dependent cost $D_{ij}(F_{ij}) = F_{ij}/(1/d_{ij} - F_{ij})$ and $C_i(G_i) = G_i/(1/c_i - G_i)$. For methods only considering linear link costs (e.g., when calculating the shortest path), we use the marginal cost $D'_{ij}(0) = 1/d_{ij}$ and $C'_i(0) = 1/c_i$ for the weights on links and CPUs, representing cost with no congestion. Let $B_i(Y_i) = b_i Y_i$, where b_i is the unit cache price. Parameters d_{ij}, c_i, and b_i are uniformly selected from the interval $[0.5\bar{d}, 1.5\bar{d}]$, $[0.5\bar{c}, 1.5\bar{c}]$, and $[0.5\bar{b}, 1.5\bar{b}]$, respectively. Mean value $\bar{d}$, $\bar{c}$ and $\bar{b}$ are given in Table I. We use the data size $L_k^{\mathrm{d}} = 0.2$ for all k, and result size $L_{mk}^{\mathrm{c}} = 0.1$ for all m, k.

Simulated network scenarios are summarized in Table I. **ER** is a connectivity-guaranteed Erdős-Rényi graph, where bi-directional links exist for each pair of nodes with probability $p = 0.07$. **grid** is a 2-dimensional 10×10 grid network. **Tree** is a full binary tree of depth 6. **Fog** is a full 3-ary tree of depth 4, where children of the same parent is concatenated linearly [6]. **GEANT** is a pan-European data network for the research and education community [15]. **LHC** (Large Hadron Collider) is a prominent data-intensive computing network for high-energy physics applications. **DTelekom** (or abbreviated as **DTel**) is a sample topology of Deutsche Telekom company [15]. **SW** (Watts-Strogatz small world) is a ring-graph with additional short-range and long-range edges.

We implement proposed Algorithm 1 (**LOAM-GCFW**) and Algorithm 2 (**LOAM-GP**), and multiple baseline methods:**CloudEC** refers to could computing and elastic caching. Computation requests are sent to nearest computing server (i.e., nodes with top 5% computation capacity), and Elastic Caching (Algorithm 2 in [12]) is used for computation reuse. **EdgeEC** refers to edge computing and elastic caching. Computations are performed at requester nodes, and Elastic Caching is used to cache data objects. **SEPLFU** refers to Shortest Extended Path (SEP) and LFU. SEP is a generalization of the shortest path from requester to data server, where links are weighted by $1/d_{ij}$, and computation is considered a link with weight $1/c_i$. Least Frequently Used (LFU) is used for cache eviction, and cache size is determined by MinCost[9]. **SEPACN** refers to SEP and Adaptive Caching with Network-wide capacity constraint (ACN)[10]. We add the total cache budget by 1 and re-run ACN in each slot.

We set $T_{\text{slot}=10}$. Both `LOAM-GCFW` and `LOAM-GP` start with the shortest extended path and $\boldsymbol{y}^0 = \mathbf{0}$. For `SEPLFU` and `SEPACN`, we run simulation for enough long time and record the lowest slot total cost. For other methods, we measure steady-state total costs after convergence. For `LOAM-GCFW`, $N = 100$. For `LOAM-GP`, stepsize $\alpha = 0.01$.

Results and analysis. We summarize the network aggregated costs T across different scenarios in Fig. 2. The costs for each scenario are normalized w.r.t. the worst method. It implies that the proposed offline method `LOAM-GCFW` and online method `LOAM-GP` outperform other methods in all scenarios, with a total cost reduction of up to 35%. Although `LOAM-GCFW` has a

[8] A new node v can randomly initiate $\boldsymbol{\phi}_v$ satisfying (3) and loop-freeness.

[9] MinCost is a cache deployment algorithm. It adds the cache capacity by 1 at the node with the highest cache miss cost in every time slot [12].

[10] ACN is a joint cache deployment and content placement method [16].

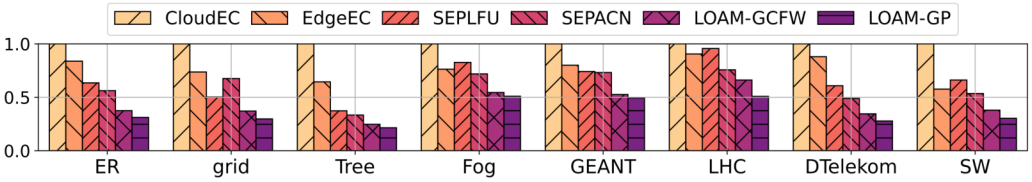

Fig. 2: Normalized total cost T of different methods in multiple network scenarios

Name	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{E}\rvert$	$\lvert\mathcal{C}\rvert$	$\lvert\mathcal{F}\rvert$	$\lvert\mathcal{T}\rvert$	$\bar{d}$	$\bar{c}$	$\bar{b}$
ER	50	240	100	20	200	5	10	20
grid	100	358	100	20	400	5	15	30
Tree	63	124	100	20	100	5	10	20
Fog	40	130	100	20	150	3	10	30
GEANT	22	66	50	10	100	3	5	10
LHC	16	62	50	10	100	3	10	15
DTel	68	546	200	30	400	5	15	20
SW	120	687	200	30	400	5	15	20

TABLE I: Network Scenarios

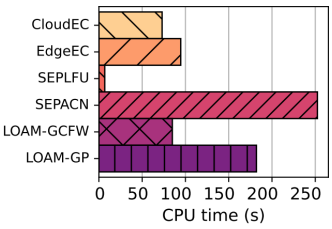

Fig. 3: Run time

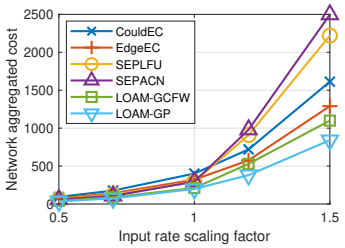

Fig. 4: T versus β

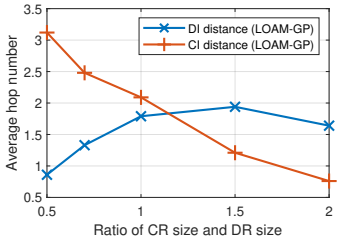

Fig. 5: T versus $L^{\mathrm{c}}_{mk}/L^{\mathrm{d}}_{k}$

stronger analytical guarantee of $1/2$ approximation, LOAM-GP outperforms LOAM-GCFW with up to 15% improvement.

Fig. 3 compares the convergent CPU time of different methods in GEANT. Although LOAM-GP outperforms LOAM-GCFW in steady state performance, by tuning N, we can directly adjust the completion time of LOAM-GCFW.

In GEANT, we scale all CI input rates $r_i(m,k)$ by a global factor β. Fig. 4 shows the change of total cost T over different β, with other parameters fixed. Our performance advantage grows as the network becomes more congested, especially against non-congestion-aware methods, e.g. SEPLFU.

In Fig. 5, we scale the result packet sizes L^{c}_{mk} in GEANT with a global factor, and compare the average travel hop number of CI and DI packets (i.e., the distance for computation offloading and data retrieval) by LOAM-GP over the ratio $L^{\mathrm{c}}_{mk}/L^{\mathrm{d}}_{k}$. The trajectories imply as the result size grows, LOAM-GP offloads tasks to nearer computation sites while leaving a longer distance for data retrieval.

VI. Conclusion

We introduced LOAM, a framework to optimize resource allocation in heterogeneous dispersed computing networks. By integrating communication, caching, and computation placement, LOAM offers robust solutions to the NP-hard aggregated cost minimization problem. Our findings reveal LOAM's potential to significantly enhance data- and computation-intensive applications, setting a new benchmark for future research in dispersed computing optimization. The adaptability of LOAM underscores its relevance in evolving networks, promising to drive advancements in the field.

Acknowledgment

Jinkun Zhang is supported by Grant EP/Y037243/1 and EP/X04047X/1.

References

[1] G. Sriram, "Edge computing vs. cloud computing: an overview of big data challenges and opportunities for large enterprises," *International Research Journal of Modernization in Engineering Technology and Science*, vol. 4, no. 1, pp. 1331–1337, 2022.

[2] A. Islam, A. Debnath, M. Ghose, and S. Chakraborty, "A survey on task offloading in multi-access edge computing," *Journal of Systems Architecture*, vol. 118, p. 102225, 2021.

[3] A. Apostolaras, G. Iosifidis, K. Chounos, T. Korakis, and L. Tassiulas, "A mechanism for mobile data offloading to wireless mesh networks," *IEEE Transactions on Wireless Communications*, 2016.

[4] K. Li, "Design and analysis of heuristic algorithms for energy-constrained task scheduling with device-edge-cloud fusion," *IEEE Transactions on Sustainable Computing*, 2022.

[5] W. Wen, Y. Cui, T. Q. Quek, F.-C. Zheng, and S. Jin, "Joint optimal software caching, computation offloading and communications resource allocation for mobile edge computing," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 7, pp. 7879–7894, 2020.

[6] K. Kamran, E. Yeh, and Q. Ma, "Deco: Joint computation scheduling, caching, and communication in data-intensive computing networks," *IEEE/ACM Transactions on Networking*, 2021.

[7] C. Barrios and M. Kumar, "Service caching and computation reuse strategies at the edge: A survey," *ACM Computing Surveys*, 2023.

[8] J. Zhang and E. Yeh. (2024) Loam: Low-latency communication, caching, and computation placement in data-intensive computing networks. [Online]. Available: https://drive.google.com/file/d/1IQ6WJ5BbAk13YQxSmbZIjMmkn4TeawTS/view?usp=sharing

[9] J. Zhang, Y. Liu, and E. Yeh, "Optimal congestion-aware routing and offloading in collaborative edge computing," in *2022 20th International Symposium on Modeling and Optimization in Mobile, Ad hoc, and Wireless Networks (WiOpt)*. IEEE, 2022, pp. 121–128.

[10] R. Gallager, "A minimum delay routing algorithm using distributed computation," *IEEE transactions on communications*, 1977.

[11] Y. Xi and E. M. Yeh, "Node-based optimal power control, routing, and congestion control in wireless networks," *IEEE Transactions on Information Theory*, vol. 54, no. 9, pp. 4081–4106, 2008.

[12] J. Zhang and E. Yeh, "Congestion-aware routing and content placement in elastic cache networks," in *IEEE INFOCOM 2024-IEEE Conference on Computer Communications*. IEEE, 2024, pp. 1–10.

[13] S. Ioannidis and E. Yeh, "Adaptive caching networks with optimality guarantees," *ACM SIGMETRICS Performance Evaluation Review*, 2016.

[14] S. Mitra, M. Feldman, and A. Karbasi, "Submodular+ concave," *Advances in Neural Information Processing Systems*, 2021.

[15] D. Rossi and G. Rossini, "Caching performance of content centric networks under multi-path routing (and more)," *Relatório técnico, Telecom ParisTech*, vol. 2011, pp. 1–6, 2011.

[16] V. S. Mai, S. Ioannidis, D. Pesavento, and L. Benmohamed, "Optimal cache allocation under network-wide capacity constraint," in *International Conference on Computing, Networking and Communications (ICNC)*. IEEE, 2019.