Dynamic Control of Data-Intensive Services over Edge Computing Networks

Yang Cai*, Jaime Llorca†, Antonia M. Tulino†‡, Andreas F. Molisch*
*University of Southern California, CA 90089, USA. Email: {yangcai, molisch}@usc.edu
†New York University, NY 10012, USA. Email: {jllorca, atulino}@nyu.edu
‡Universityà degli Studi di Napoli Federico II, Naples 80138, Italy. Email: antoniamaria.tulino@unina.it

***Abstract*—Next-generation distributed computing networks (e.g., edge and fog computing) enable the efficient delivery of *delay-sensitive*, *compute-intensive* applications by facilitating access to computation resources in close proximity to end users. Many of these applications (e.g., augmented/virtual reality) are also *data-intensive*: in addition to user-specific (*live*) data streams, they require access to shared (*static*) digital objects (e.g., image database) to complete the required processing tasks. When required objects are not available at the servers hosting the associated service functions, they must be fetched from other edge locations, incurring additional communication cost and latency. In such settings, overall service delivery performance shall benefit from jointly optimized decisions around (i) routing paths and processing locations for live data streams, together with (ii) cache selection and distribution paths for associated digital objects. In this paper, we address the problem of dynamic control of data-intensive services over edge cloud networks. We characterize the network stability region and design the first throughput-optimal control policy that coordinates processing and routing decisions for both live and static data-streams. Numerical results demonstrate the superior performance (e.g., throughput, delay, and resource consumption) obtained via the novel multi-pipeline flow control mechanism of the proposed policy, compared with state-of-the-art algorithms that lack integrated stream processing and data distribution control.**

I. Introduction

The class of augmented information (AgI) services encompasses to a wide range of services and applications designed to deliver information of real-time relevance that results from the online aggregation, processing, and distribution of multiple data streams. AgI services such as system automation (e.g., smart homes/factories/cities, self-driving cars) and Metaverse experiences (e.g., multiplayer gaming, immersive video, virtual/augmented reality) are driving unprecedented requirements for communication, computation, and storage resources [1]. To address this need, distributed cloud network architectures such as multi-access edge computing (MEC) are becoming a promising paradigm, providing end users with efficient access to nearby computation resources. Together with continued advances in network virtualization and programmability [2], distributed cloud networks allow flexible and elastic deployment of disaggregated services composed of multiple software functions that can be dynamically instantiated at distributed network locations.

The associated problem of service function chain (SFC) orchestration has received significant attention in the recent literature. One main line of work studies this problem in a static setting, where the goal is to allocate multi-dimensional (i.e., communication, computation, storage) network resources for function/data placement and flow routing, in order to optimize a network-wide objective, e.g., maximizing accepted traffic [3] or minimizing operational cost [4]. While useful for long timescale end-to-end service optimization, the corresponding formulations typically take the form of (NP-Hard) mixed integer programs that lead to algorithms with either high complexity or sub-optimal performance, and overlook the dynamic nature of service demands and network conditions.

To address the problem in a dynamic scenario, [5] introduced the *dynamic cloud network control* problem, where one needs to make online packet processing, routing, and scheduling decisions in response to stochastic system states (e.g., service demands and resource capacities). Among existing techniques, Lyapunov-drift control [6] is a widely-used approach to design throughput-optimal algorithms, e.g., DCNC [5] and UCNC [7], by *dynamically* exploring routing and processing diversity. In general, centralized routing policies, e.g., UCNC, that exploit global knowledge to guarantee that packets follow *acyclic* paths, can attain better delay performance than their fully distributed counterparts, e.g., DCNC. We refer the reader to a longer version of this paper [8] for a more comprehensive literature review.

An increasingly relevant feature of next-generation AgI services is their *intensive data requirements*: in addition to *live data* streams, task (or service function) processing also requires access to pre-stored *static data* objects. For example, in an augmented reality (AR) application, access to pre-stored scene objects is required to generate augmented/enhanced images or video streams (see Fig. 1). While the aforementioned studies on dynamic cloud network control optimize the live data stream (routing and processing) pipeline without considering the access to static data, other works in data-centric computing networks [9] have addressed the static data distribution and processing problem without considering the live data pipeline.

To close the gap, this paper addresses the problem of "online delivery of *data-intensive AgI services* over edge computing networks", which require real-time aggregation, processing, and distribution of live and static data streams. We design a control policy that *jointly* decides (i) routing paths and processing locations for live streams, and (ii) cache selection and distribution paths for associated static objects, to optimize end-to-end

978-1-6654-3540-6/22/$31.00 © 2022 IEEE

service delivery performance. The main challenges come from the coupling of live and static data routing with the associated function processing decisions. To the best of our knowledge, this is the first paper that studies joint processing and routing control of multiple pipelines for the online delivery of stream-processing services.

The contributions of this work are summarized as follows:

- We characterize the stability region of distributed cloud networks supporting data-intensive AgI service delivery.
- We design the first throughput-optimal control policy for online data-intensive service delivery, termed DI-DCNC, which coordinates processing and routing decisions for live and static data streams.
- We conduct illustrative numerical experiments to demonstrate the superior performance of DI-DCNC.

II. System Model

A. Cache-Enabled MEC Network Model

Consider a distributed cloud network, modeled by a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, with $\mathcal{V}$ and $\mathcal{E}$ denoting the node and edge sets, respectively. Each vertex $i \in \mathcal{V}$ represents a node equipped with computation resources (e.g., edge server) for service function processing. Each edge $(i,j) \in \mathcal{E}$ represents a point-to-point communication link, which can support data transmission from node i to j. Let $\delta^-(i)$ and $\delta^+(i)$ denote the incoming and outgoing neighbor sets of node i, respectively.

Time is slotted, and the network processing and transmission capacities are quantified as follows. (i) C_i: the maximum number of processing instructions (e.g., floating point operations) that can be executed in one time slot at node i. (ii) C_{ij}: the maximum number of data units (e.g., packets) that can be transmitted in one time slot over link (i,j).

We assume that the network nodes are also equipped with storage resources to cache databases composed of digital objects whose access may be required for service function processing. In this paper, we focus on cache selection and routing decisions with fixed database placement and refer the reader to [8] for extensions that include database placement/replacement policies. Let $\mathcal{V}(k) \subset \mathcal{V}$ denote the *static sources* of database k, i.e., the set of nodes that cache database k.

B. Data-Intensive Service Model

In this paper, we illustrate the proposed algorithm in the context of a single-function data-intensive service. The generalization to multiple service functions is briefly described in Section V and presented in detail in [8].

Each service ϕ includes one task (or service function) that must process **coupled** user-specific data and associated digital objects – referred to as *live packets* and *static packets*, respectively – for the generation of consumable streams. As shown in Fig. 1, an example AR application is composed of an AR processing function that must process the user live data (source video stream) together with associated static objects (scene objects) to create the output augmented data (augmented video stream).

The function of service ϕ is described by four parameters $(r_\phi, \zeta_\phi, k_\phi, \xi_\phi)$, defined as: (i) *workload* r_ϕ: the amount of computation resource (e.g., instructions per time slot) required to process one input live packet, (ii) *merging ratio* ζ_ϕ: the number of static packets required to process one input live packet, (iii) *database ID* k_ϕ: the ID of the database containing the required static object, and (iv) *scaling factor* ξ_ϕ: the number of output packets per input live packet.

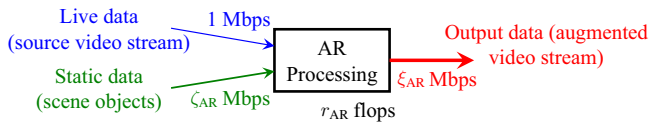

Fig. 1. Example AR application with one processing function that takes two inputs (live and static data) to create augmented data.

C. Client Model

We define a client c by a 3-tuple (s, d, ϕ), denoting the source node s (where the live packets arrive to the network), the destination node d (where the output packets are requested for consumption), and the requested service ϕ, respectively.

1) Live Packet Arrival: Let $a^{(c)}(t)$ be the number of live data packets of client c arriving to the network at time t. For each client c, we assume the arrival process $\{a^{(c)}(t) : t \geq 0\}$ is i.i.d. over time, with mean arrival rate of $\lambda^{(c)}$, and bounded maximum arrival number.

2) Static Packet Provisioning: Upon a live packet arrival, one static source $v \in \mathcal{V}(k_\phi)$ must be selected to produce a copy of the required static packet.

We refer to a live packet and its associated static packet required for its processing as belonging to the same *packet-level request*.

D. Queuing System

Each packet (live or static) arriving to the network gets associated with a route for its delivery, and we establish actual queues to accommodate packets waiting for processing or routing. For each link $(i,j) \in \mathcal{E}$, we create one *transmission* queue collecting all packets waiting to cross the link. In addition, a novel **paired-packet queuing system** is constructed at each node $i \in \mathcal{V}$, composed of: (i) the *processing* queue collecting the *paired* live and static packets concurrently present at node i, which are ready for joint processing, and (ii) the *waiting* queue collecting the *unpaired* live or static packets waiting for their in-transit associates, which are not qualified for processing until joining the processing queue upon their associates' arrivals.

III. Policy Space and Network Stability Region

A. Transformation to an Augmented Layered Graph

We propose an augmented layered graph (ALG) model extending the layered graph [7], to analyze and optimize the data-intensive AgI service delivery problem.

1) Topology: The ALG is composed of three pipelines, referred to as *live*, *static* and *output* pipelines, and each pipeline has the same topology as the actual network $\mathcal{G}$, except for the static pipeline that includes an additional node o'_1 and its outgoing edges to all static sources. We note that: (i) The live, static, and output pipelines accommodate exogenously arriving

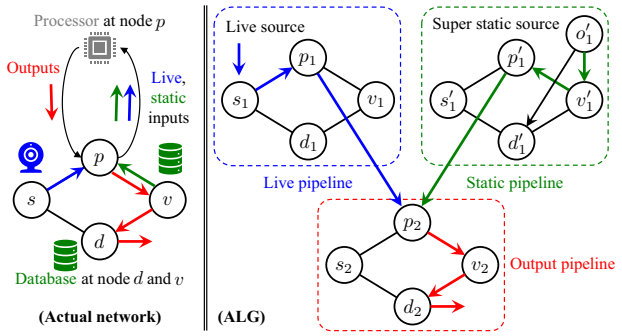

Fig. 2. Illustration of the ALG model for the delivery of an AR service.

live packets, in-network replicated static packets, and processed output packets, respectively, and represent their associated routing over the network. (ii) In the static pipeline, we create a *super static source* node o_1' connected to all static sources $v \in \mathcal{V}(k_\phi)$, which is equivalent to assuming that o_1' *is the only static source of database* k_ϕ.[1] (iii) The three pipelines are placed in two layers, describing packets *before* (layer 1, henceforth indicated by subscript 1) and *after* processing (layer 2), respectively. There are inter-layer edges connecting corresponding nodes, which represent processing operations, i.e., the live and static packets pushed through these edges are processed into the output packet in the actual network.

In the example in Fig. 2, the live packet and static packet (which is generated via replication at the static source v) are routed following the blue and green paths to node p, respectively. After getting processed at node p, the output packet is delivered along the red path to destination d.

Mathematically, given the actual network $\mathcal{G}$, the ALG of service ϕ is given by $\mathcal{G}^\phi = (\mathcal{V}^{(\phi)}, \mathcal{E}^{(\phi)})$, with $\mathcal{V}^{(\phi)} = \{i_1, i_1', i_2 : i \in \mathcal{V}\} \cup \{o_1'\}$ and $\mathcal{E}^{(\phi)} = \mathcal{E}_{\text{tr}}^{(\phi)} \cup \mathcal{E}_{\text{pr}}^{(\phi)} \cup \mathcal{E}_{\text{st}}^{(\phi)}$, in which $\mathcal{E}_{\text{tr}}^{(\phi)} = \{(i_1, j_1), (i_1', j_1'), (i_2, j_2) : (i, j) \in \mathcal{E}\}$, $\mathcal{E}_{\text{pr}}^{(\phi)} = \{(i_1, i_2), (i_1', i_2) : i \in \mathcal{V}\}$, and $\mathcal{E}_{\text{st}}^{(\phi)} = \{(o_1', v_1') : v \in \mathcal{V}(k_\phi)\}$.

2) Flow Constraints: Denote by $f_{ij} \geq 0$ the flow rate associated with edge $(i, j) \in \mathcal{E}^{(\phi)}$, defined as the average rate of packets (in *packets per slot*) traversing edge $(i, j) \in \mathcal{E}^{(\phi)}$. In particular, for $\forall (i, j) \in \mathcal{E}$, $f_{i_1 j_1}$, $f_{i_1' j_1'}$, $f_{i_2 j_2}$ represent the transmission rates of the live, static, and output packets over link (i, j); for $\forall v \in \mathcal{V}(k_\phi)$, $f_{o_1' v_1'}$ is the local replication rate of the static packets at static source v; for $\forall i \in \mathcal{V}$, $f_{i_1 i_2}$, $f_{i_1' i_2}$ denote the processing rates of live and static packets at node i, respectively, while $\xi_\phi f_{i_1 i_2}$ denotes the rate of the resulting output packets at node i.

The flow rates must satisfy the following constraints.

(i) Live and output flow conservation: for $\forall i \in \mathcal{V}$,

$$\sum\nolimits_{j \in \delta^+(i)} f_{i_1 j_1} + f_{i_1 i_2} = \sum\nolimits_{j \in \delta^-(i)} f_{j_1 i_1}, \tag{1a}$$

$$\sum\nolimits_{j \in \delta^+(i)} f_{i_2 j_2} = \sum\nolimits_{j \in \delta^-(i)} f_{j_2 i_2} + \xi_\phi f_{i_1 i_2}, \tag{1b}$$

i.e., the total outgoing rate of packets that are transmitted and processed equals the total incoming rate of packets that are received and generated (via service processing), for both live (1a) and output (1b) data streams.

(ii) Static flow conservation: for $\forall i \in \mathcal{V}$,

$$\sum_{j \in \delta^+(i)} f_{i_1' j_1'} + f_{i_1' i_2} = \sum_{j \in \delta^-(i)} f_{j_1' i_1'} + f_{o_1' i_1'} \mathbf{1}_{\{i \in \mathcal{V}(k_\phi)\}}, \tag{2}$$

which can be interpreted analogous to (i), except for that static packets are generated via replication rather than processing.

(iii) Data merging: for $\forall i \in \mathcal{V}$,

$$f_{i_1' i_2} = \zeta_\phi f_{i_1 i_2}, \tag{3}$$

i.e., the processing rates of static and live packets are associated via the "merging ratio" ζ_ϕ defined in Section II-B.

B. Policy Space

Next, we define the space of policies for data-intensive service delivery, encompassing joint packet processing, routing, and replication decisions. In line with [7] and the increasingly adopted software defined networking (SDN) paradigm, we focus on centralized routing and distributed scheduling policies. Upon a packet-level request and associated live packet arrival at its source, the policy selects: (i) routing path and processing location for the live packet, (ii) cache selection (i.e., chosen static source to replicate the static packet) and distribution path for the associated static packet, and (iii) routing path for the resulting output packet. In addition, each node/link needs to schedule packets for processing/transmission at each time slot.

1) Decision Variables: A policy thus includes two actions.

Route Selection: For each packet-level request, choose a set of edges in the ALG and associated flow rates satisfying (1) – (3), based on which: (i) the cache selection decision is specified by the replication rate $f_{o_1' v_1'}$, (ii) the routing path for each (live or static) packet is given by the edges with non-zero rates in the corresponding pipeline, and (iii) the processing location selection is indicated by the processing rates $f_{i_1 i_2}$ and $f_{i_1' i_2}$, which guarantee that the live and static packets meet at the same node i due to (3).

Packet Scheduling: At each time slot, for each node $i \in \mathcal{V}$ and link $(i, j) \in \mathcal{E}$, schedule packets from the local processing queues (which hold *paired* packets) and transmission queues, for operation, such that the incurred resource consumption does not exceed the corresponding capacities C_i and C_{ij}.

2) Efficient Policy Space: We define the efficient policy space as that in which policies only select *acyclic* routing paths, which can be shown to not compromise the achievable performance (e.g., throughput, delay, and resource consumption).

As illustrated in Fig. 2, for each request, we choose a star route (STAR) σ in the ALG, which includes an internal node p_2 and three leaf nodes s_1, o_1' and d_2 connecting to it via paths $\sigma_1 \cup (p_1, p_2)$, $\sigma_1' \cup (p_1', p_2)$, and σ_2, respectively, where p_2 denotes the *processing location*, and σ_1, σ_1', and σ_2 the acyclic *routing paths* of the live (from s_1 to p_1), static (from o_1' to p_1'), and output (from p_2 to d_2) packets, respectively.

For each client c, the set of all STARs, denoted by $\mathcal{F}_c$, is *finite*, and the route selection decision can be expressed by

$$\mathcal{A}(t) = \left\{ a^{(c,\sigma)}(t) : \sum\nolimits_{\sigma \in \mathcal{F}_c} a^{(c,\sigma)}(t) = a^{(c)}(t),\ \forall c, \sigma \right\} \tag{4}$$

[1] If node o_1' can provide static packets to i_1' along a path $(o_1', v_1', \cdots, i_1')$ in the ALG; then, in the actual network, we can select v to produce the packets and send them to node i along the rest of the path. And vice versa.

where $a^{(c,\sigma)}(t)$ denotes the number of requests of client c that arrive at time t and get associated with STAR σ for delivery.

C. Network Stability Region

We define the *network stability region* Λ as the set of arrival vectors $\boldsymbol{\lambda} = \{\lambda^{(c)} : \forall\, c\}$ under which there exists a policy that stabilizes the actual queues. The stability region describes the network capability to support service requests, and is characterized by the following theorem.

Theorem 1: An arrival vector $\boldsymbol{\lambda}$ is interior to the stability region Λ if and only if, for each client c, there exist probability values $\big\{\mathbb{P}_c(\sigma) : \sum_{\sigma\in\mathcal{F}_c}\mathbb{P}_c(\sigma) = 1\big\}$, such that: for $\forall\, i, (i,j)$,

$$\sum_{c,\sigma}\lambda^{(c)}\rho_i^{(c,\sigma)}\mathbb{P}_c(\sigma) \le C_i,\ \sum_{c,\sigma}\lambda^{(c)}\rho_{ij}^{(c,\sigma)}\mathbb{P}_c(\sigma) \le C_{ij} \tag{5}$$

where $\rho_i^{(c,\sigma)}$ and $\rho_{ij}^{(c,\sigma)}$ denote the processing and transmission resource loads imposed on node i and link (i,j) if a packet of client c is delivered via STAR σ, given by:

$$\rho_i^{(c,\sigma)} = r_\phi \mathbf{1}_{\{(i_1,\,i_2)\in\sigma\}} \tag{6a}$$

$$\rho_{ij}^{(c,\sigma)} = \mathbf{1}_{\{(i_1,\,j_1)\in\sigma\}} + \zeta_\phi \mathbf{1}_{\{(i_1',\,j_1')\in\sigma\}} + \xi_\phi \mathbf{1}_{\{(i_2,\,j_2)\in\sigma\}}. \tag{6b}$$

Proof: See [8, Theorem 1]. □

In (6b) , the three terms denote the resource loads imposed on link (i,j) if the live, static, and output packets traverse (i_1,j_1), (i_1',j_1'), and (i_2,j_2) in STAR σ, respectively.

IV. Proposed Algorithm

Next, we present the proposed *data-intensive dynamic cloud network control* (DI-DCNC) policy. We first introduce a single-hop virtual system (in Section IV-A) to derive packet routing decisions (in Section IV-B). Then, we present the packet scheduling policy, and summarize the actions taken in the actual network (in Section IV-C).

A. Virtual System

1) Precedence Constraint: In line with [7], [10], we create a virtual network, where the *precedence constraint* that imposes that packets must be transmitted hop-by-hop along its route, is relaxed by allowing a packet upon route selection to be immediately inserted into the virtual queues associated with every link in the route.

We emphasize that the virtual system is only used for route selection and is NOT relevant to packet scheduling.

2) Virtual Queues: We create virtual queues $\tilde{Q}_i(t)$ for $\forall\, i \in \mathcal{V}$ and $\tilde{Q}_{ij}(t)$ for $\forall\,(i,j)\in\mathcal{E}$, to represent the resource loads of the nodes and links in the virtual system, which are interpreted as the *anticipated* resource loads in the actual network.

The queuing dynamics are given by:

$$\tilde{Q}_i(t+1) = \big[\tilde{Q}_i(t) - C_i + \tilde{a}_i(t)\big]^+ \tag{7a}$$

$$\tilde{Q}_{ij}(t+1) = \big[\tilde{Q}_{ij}(t) - C_{ij} + \tilde{a}_{ij}(t)\big]^+ \tag{7b}$$

where C_i and C_{ij} are the amount of processing/transmission resource available at each time slot, and $\tilde{a}_i(t)$ and $\tilde{a}_{ij}(t)$ denote the "additional" resource loads imposed on the nodes and links upon newly arriving requests at time t. Recall that each request gets associated with a route for delivery upon arrival, which immediately impacts the virtual queuing states of all links in the route. Hence,

$$\tilde{a}_i(t) = \sum_{c,\sigma}\rho_i^{(c,\sigma)}a^{(c,\sigma)}(t),\ \tilde{a}_{ij}(t) = \sum_{c,\sigma}\rho_{ij}^{(c,\sigma)}a^{(c,\sigma)}(t) \tag{8}$$

with $\rho_i^{(c,\sigma)}$ and $\rho_{ij}^{(c,\sigma)}$ given by (6).

B. Optimal Virtual Network Decisions

1) Lyapunov Drift Control: Next, we leverage Lyapunov drift control theory to derive a policy that stabilizes the *normalized* virtual queues $\boldsymbol{Q}(t) = \big\{Q_i(t) \triangleq \tilde{Q}_i(t)/C_i : \forall\, i \in \mathcal{V}\big\} \cup \big\{Q_{ij}(t) = \tilde{Q}_{ij}(t)/C_{ij} : \forall\,(i,j)\in\mathcal{E}\big\}$, which have equivalent stability properties as the virtual queues and can be interpreted as *queuing delay* in the virtual system.

Consider the Lyapunov function $L(t) \triangleq \|\boldsymbol{Q}(t)\|^2/2$ and its drift $\Delta(\boldsymbol{Q}(t)) \triangleq L(t+1) - L(t)$. We can derive the following upper bound of the drift $\Delta(\boldsymbol{Q}(t))$ (see [8, Appendix B.1]):

$$\Delta(\boldsymbol{Q}(t)) \le B - \|\boldsymbol{Q}(t)\|_1 + \sum\nolimits_{c,\sigma} O^{(c,\sigma)}(t)\, a^{(c,\sigma)}(t) \tag{9}$$

where B is a constant, and $O^{(c,\sigma)}(t)$ is referred to as the weight of STAR σ, given by:

$$O^{(c,\sigma)}(t) = \sum_{(i,j)}\big[w_{i_1j_1}^{(c)}(t)\,\mathbf{1}_{\{(i_1,j_1)\in\sigma\}} + w_{i_1'j_1'}^{(c)}(t)\,\mathbf{1}_{\{(i_1',j_1')\in\sigma\}} + w_{i_2j_2}^{(c)}(t)\,\mathbf{1}_{\{(i_2,j_2)\in\sigma\}}\big] + \sum_{i\in\mathcal{V}} w_{i_1i_2}^{(c)}(t)\mathbf{1}_{\{(i_1,i_2)\in\sigma\}} \tag{10}$$

in which $w_{i_1j_1}^{(c)}(t) = Q_{ij}(t)/C_{ij}$, $w_{i_1'j_1'}^{(c)}(t) = \zeta_\phi(Q_{ij}(t)/C_{ij})$, $w_{i_2j_2}^{(c)}(t) = \xi_\phi(Q_{ij}(t)/C_{ij})$, $w_{i_1i_2}^{(c)}(t) = r_\phi(Q_i(t)/C_i)$, and we define $w_{i_1'i_2}^{(c)}(t) = w_{o_1'v_1'}^{(c)}(t) = 0$ for $\forall\, i\in\mathcal{V}, v\in\mathcal{V}(k_\phi)$.

The proposed algorithm is designed to minimize (9) over the route selection decision $\mathcal{A}(t)$ given by (4), or equivalently,

$$\min_{\mathcal{A}(t)}\ \sum_{c,\sigma} O^{(c,\sigma)}(t)\, a^{(c,\sigma)}(t),\ \text{s.t. (4).} \tag{11}$$

2) Route Selection: Given the linear structure of (11), the optimal route selection decision is given by:

$$a^{\star\,(c,\sigma)}(t) = a^{(c)}(t)\mathbf{1}_{\{\sigma=\sigma^\star\}},\ \sigma^\star \triangleq \arg\min_{\sigma\in\mathcal{F}_c} O^{(c,\sigma)}(t), \tag{12}$$

i.e., all requests of client c arriving at time t are delivered via the *min-STAR* $\sigma^\star$, i.e., the STAR with the minimum weight.

The remaining problem is to find the min-STAR $\sigma^\star$. To this end, we create a *weighted ALG* in which we assign the weight $w_{ij}^{(c)}(t)$ defined in (10) to each edge $(\imath,\jmath)$ in the ALG, under which the weight of STAR σ, $O^{(c,\sigma)}(t)$, equals the sum of individual edge weights, given by:

$$O^{(c,\sigma)}(t) = |\sigma_1| + |\sigma_1'| + w_{p_1p_2}^{(c)}(t) + |\sigma_2| \tag{13}$$

where $|\sigma_1|$, $|\sigma_1'|$, and $|\sigma_2|$ denote the weights of the routing paths of the live, static, and output packets in the weighted ALG, respectively, and p is the processing location.

Based on (13), we propose to find the min-STAR in two steps. First, we fix the processing location p, so we can treat the optimization of path weights for the live, static, and output

packets as *separate* problems. For example, minimizing $|\sigma_1|$ is equivalent to finding the shortest path from s_1 to p_1 in the weighted ALG, and similarly for $|\sigma_1'|$ (from o_1' to p_1') and $|\sigma_2|$ (from p_2 to d_2). Therefore, the minimum weight of all STARs *with node $p \in \mathcal{V}$ as their processing location* is

$$W_p = \mathrm{SPW}(s_1, p_1) + \mathrm{SPW}(o_1', p_1') + w^{(c)}_{p_1 p_2}(t) + \mathrm{SPW}(p_2, d_2) \tag{14}$$

where $\mathrm{SPW}(\imath, \jmath)$ denotes the weight of the shortest path $\mathrm{SP}(\imath, \jmath)$ from node $\imath$ to $\jmath$ in the weighted ALG (the path of the static packet is given by $\mathrm{SP}(o_1', p_1') = (o_1', v_1'(p)) \cup \mathrm{SP}(v_1'(p), p_1')$, with $v(p)$ denoting the selected static source). Then, in the second step, we select the processing location $p^\star \in \mathcal{V}$ to minimize the weight (14). The procedure is summarized as follows:

Route Selection: At each time slot t, all requests of client c get associated with the STAR specified as:

$$\text{Processing: } p^\star = \arg\min_{p \in \mathcal{V}} W_p \text{ (given by (14))}, \tag{15a}$$

$$\text{Cache selection: } v^\star = v(p^\star), \tag{15b}$$

$$\text{Transmission: } \underbrace{\mathrm{SP}(s_1, p_1^\star)}_{\text{live packet}}, \underbrace{\mathrm{SP}(v_1^{\star\prime}, p_1^{\star\prime})}_{\text{static packet}}, \underbrace{\mathrm{SP}(p_2^\star, d_2)}_{\text{output packet}}. \tag{15c}$$

3) Complexity Analysis: The above procedure requires running Dijkstra's algorithm to compute the shortest path from s_1, o_1', and d_2 to the other nodes in the ALG, for each possible choice of p, with overall complexity $\mathcal{O}(|\mathcal{V}| \cdot |\mathcal{E}| \log |\mathcal{V}|)$ [11].

C. Optimal Actual Network Decisions

Next, we present the control decisions in the actual network. We adopt the route selection decisions made in the virtual network (15), and the extend the nearest-to-origin (ENTO) policy [10] for packet scheduling, described as follows.

Packet Scheduling: *At each time slot, for each node/link, give priority to the packets in the corresponding processing/transmission queues that have crossed the smallest number of edges in the ALG.*

To sum up, the proposed **DI-DCNC** algorithm operates as follows. At each time slot: (i) assign the STAR (15) to all requests of client c for delivery, (ii) at each node/link, schedule packets for processing/transmission by ENTO, and (iii) update the virtual queues by (7).

D. Performance Analysis

Theorem 2: For any arrival vector $\boldsymbol{\lambda}$ within the stability region Λ, the actual queues are rate stable under DI-DCNC.

Proof: See [8, Theorem 2]. □

V. Extensions to General Service Model

In general, a service can perform (i) multi-step processing (i.e., more than one service function) on (ii) multiple source streams (i.e., more than one live and static input).

To handle such services, we can incorporate more (i) layers and (ii) pipelines, into the ALG. The (extended) DI-DCNC algorithm follows the same rule for route selection, i.e., finding the min-weight route to deliver each packet-level request, and a key step is to select a *sequence* of processing locations, which transforms the problem into multiple unicast problems. We refer the reader to [8] for detailed descriptions and derivations.

VI. Numerical Results

In this section, we evaluate DI-DCNC in the 16-node grid network of Fig. 3. Each cylinder node caches a database, with the associated digit representing the database ID. The network resources are summarized in Table I. We consider four clients, and each service includes 2 functions, specified in Table II. To recall, client = (source, destination, service), function = (workload, merging ratio, database ID, scaling factor). The arrival processes are modeled by independent Poisson processes with λ Mbps.

TABLE I
Available Processing / Transmission Resources

Processing	$C_i = 10$ GHz for $i = A, B, C, D$; 5 GHz elsewhere
Transmission	$C_{ij} = 20$ Mbps for $\forall (i, j)$

TABLE II
Clients (s, d, ϕ) and Function Spec $(r\,[\text{GHz/Mbps}], \zeta, k, \xi)$

Client	(E, H, ϕ_1)	(F, G, ϕ_2)	(G, F, ϕ_3)	(H, E, ϕ_4)
Func 1	$(.2, 1, 1, 1)$	$(.5, 2, 3, 1)$	$(.1, 1, 5, 1)$	$(1, 5, 7, 1/2)$
Func 2	$(.2, 1, 2, 2)$	$(.5, 3, 4, 1/2)$	$(.1, 1, 6, 3)$	$(1, 10, 8, 1/3)$

We employ two benchmark algorithms: (i) Static-to-live (S2L), which makes individual routing decisions for live data, and then brings the static packets to the selected processing nodes. (ii) Live-to-static (L2S), which brings live data to the "nearest" (in the weighted ALG) static source for processing.

A. Network Stability Region

First, we study the network stability regions of the algorithm. Fig. 4 depicts the attained average delay under different arrival rates (which is set to ∞ if the network is not stable).

As we can observe, the DI-DCNC algorithm attains good (average) delay performance over a wide range of arrival rates; when λ crosses a *critical* point ($\lambda_1 \approx 12.9$ Mbps), the network is no longer stable, which is the boundary of its stability region. Similar behaviors are observed from the other two algorithms, and we find that DI-DCNC outperforms them in terms of the achieved throughput: 12.9 Mbps (DI-DCNC) $>$ 9.9 Mbps (S2L) $>$ 5.0 Mbps (L2S). Therefore, compared to S2L and L2S, DI-DCNC can better exploit network resources to improve total throughput. We also notice that the delay attained by DI-DCNC is very similar and not lower than both benchmarks in the low-congestion regimes. This can be explained by the throughput-oriented design of DI-DCNC, which tries to reduce the *aggregate* queuing delay of both live and static data pipelines. Such objective, while closely related (especially in high congestion regimes), is not exactly equivalent to the actual service delay, which depends on the *maximum* delay between the two *concurrent* pipelines.

B. Resource Occupation

Next, we compare the resource occupation of the algorithms. Assume that the available processing/transmission capacities at each node/link are given by α_1 and α_2 (in *percentage*) of corresponding maximum budgets, respectively. We then define

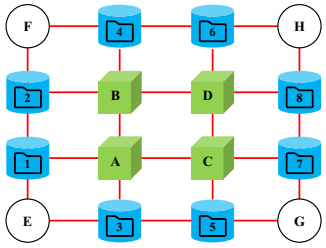

Fig. 3. Network and stored databases.

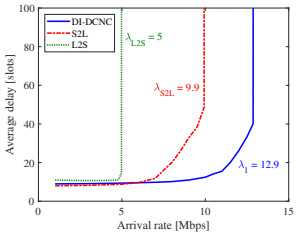

Fig. 4. Network stability region.

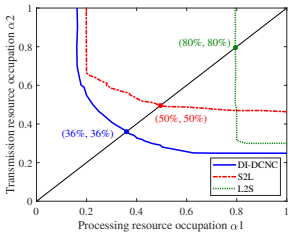

Fig. 5. Resource occupation.

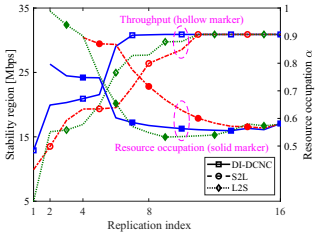

Fig. 6. Impact of database replication.

the *feasible region* as the collection of (α_1, α_2) pairs under which the delay requirement is fulfilled. Assume $\lambda = 4$ Mbps and average delay constraint $= 20$ time slots.

The border lines of the feasible regions are shown in Fig. 5. Since a lower latency can be attained with more resources, i.e., $(\alpha_1, \alpha_2) \to (1, 1)$, the feasible regions are to the upper-right of the border lines. As we can observe, DI-DCNC can save the most network resources. In particular, if $\alpha_1 = \alpha_2 = \alpha$, the resource saving ratios, i.e., $1 - \alpha$, of the algorithms are: 64% (DI-DCNC) $> 50\%$ (S2L) $> 20\%$ (L2S). Another observation is: S2L is transmission-constrained (compared to its sensitivity to processing resource consumption). To wit: in the horizontal direction, it can achieve a maximal saving ratio of 80% (when $\alpha_2 = 1$), which is comparable to DI-DCNC; however, the maximal ratio is 50% for transmission resources (when $\alpha_1 = 1$), and there is a large gap to DI-DCNC in that dimension. The reason is that S2L ignores the transmission load of the static packets, leading to additional transmission resource consumption. While L2S is processing-constrained, because it is forced to use the processing resources at the static sources.

C. Impact of Database Replication

Finally, we study the impact of database replication. We introduce the notion of *replication index* to denote the number of copies of each database cached in the network. We assume equal replication index for all databases.[2] We evaluate the same metrics of throughput and resource occupation, assuming $\lambda = 15$ Mbps, delay requirement $= 20$ slots, $\alpha_1 = \alpha_2 = \alpha$.

The results are shown in Fig. 6. For each algorithm, as the replication index grows, we observe increasing throughput and decreasing resource occupation,[3] because with a higher replication index: for DI-DCNC and S2L, the distance to a static source reduces, while for L2S, processing resources at more network locations can be used. When comparing the three algorithms, we find that: (i) when the replication index hits 16, the three algorithms become identical, and their performance converge. (ii) DI-DCNC achieves the best performance in most of the interval; in addition, it converges to the saturation point (the point after which the gain from increasing replication index becomes marginal) faster than other algorithms.

[2]For example, in Fig. 3, the replication index is 1, as each database is only cached at one node. If all 16 nodes cache all databases, the replication index would be 16.

[3]In general, resource occupation reduces with replication index, but there are exceptions (see L2S curve in [7, 13]). A possible reason is that the developed algorithms do not explicitly minimize hop-distance, which can become more dominant than queuing delay in low congestion regimes (with low arrival rate or high replication index), making the end-to-end delay non-monotonic.

VII. Conclusions

In this paper, we designed a cloud network control policy for online delivery of data-intensive AgI services. We considered a more general service model, requiring both live and static data-streams as inputs to a service function. Based on this model, we characterized the network stability region and proposed an efficient, throughput-optimal algorithm, DI-DCNC, which jointly decides (i) routing paths and processing locations for live data streams, and (ii) cache selection and distribution paths for associated data objects. Via numerical experiments, we demonstrated the superior performance of multiple pipeline coordination for the delivery of next-generation data-intensive real-time stream-processing services.

Acknowledgments: This work was supported by National Science Foundation (NSF) CNS-1816699, RINGS-2148315.

References

[1] Y. Cai, J. Llorca, A. M. Tulino, and A. F. Molisch, "Compute- and data-intensive networks: The key to the Metaverse," arXiv:2204.02001. [Online]. Available: https://arxiv.org/abs/2204.02001, Apr. 2022.

[2] "The programmable cloud network – A primer on SDN and NFV," Alcatel-Lucent, Paris, France, White Paper, Jun, 2013. [Online]. Available: https://whitepapers.us.com/.

[3] M. Huang, W. Liang, Y. Ma, and S. Guo, "Maximizing throughput of delay-sensitive NFV-enabled request admissions via virtualized network function placement," *IEEE Trans. Cloud Comput.*, vol. 9, no. 4, pp. 1535–1548, Oct.-Dec. 2021.

[4] Y. Yue, B. Cheng, X. Liu, M. Wang, B. Li, and J. Chen, "Resource optimization and delay guarantee virtual network function placement for mapping SFC requests in cloud networks," *IEEE Trans. Netw. Service Manag.*, vol. 18, no. 2, pp. 1508–1523, Jun. 2021.

[5] H. Feng, J. Llorca, A. M. Tulino, and A. F. Molisch, "Optimal dynamic cloud network control," *IEEE/ACM Trans. Netw.*, vol. 26, no. 5, pp. 2118–2131, Oct. 2018.

[6] M. J. Neely, *Stochastic network optimization with application to communication and queueing systems*. San Rafael, CA, USA: Morgan & Claypool, 2010.

[7] J. Zhang, A. Sinha, J. Llorca, A. M. Tulino, and E. Modiano, "Optimal control of distributed computing networks with mixed-cast traffic flows," *IEEE/ACM Trans. Netw.*, vol. 29, no. 4, pp. 1760–1773, Aug. 2021.

[8] Y. Cai, J. Llorca, A. M. Tulino, and A. F. Molisch, "Joint compute-caching-communication control for online data-intensive service delivery," arXiv:2205.01944, May 2022.

[9] K. Kamran, E. Yeh, and Q. Ma, "DECO: Joint computation scheduling, caching, and communication in data-intensive computing networks," *IEEE/ACM Trans. Netw.*, vol. 1, no. 99, pp. 1–15, 2021.

[10] A. Sinha and E. Modiano, "Optimal control for generalized network flow problems," *IEEE/ACM Trans. Netw.*, vol. 26, no. 1, pp. 506–519, 2018.

[11] J. Kleinberg and Éva Tardos, Algorithm design. Delhi, India: Pearson Education India, 2006.