Function Load Balancing Over Networks

Derya Malak, *Member, IEEE*, and Muriel Médard, *Fellow, IEEE*

***Abstract*—Using networks as a means of computing can reduce the communication flow over networks. We propose to distribute the computation load in stationary networks and formulate a flow-based delay minimization problem that jointly captures the costs of communications and computation. We exploit the distributed compression scheme of Slepian-Wolf that is applicable under any protocol information. We introduce the notion of entropic surjectivity as a measure of function's sparsity and to understand the limits of functional compression for computation. We leverage Little's law for stationary systems to provide a connection between surjectivity and the computation processing factor that reflects the proportion of flow that requires communications. This connection gives us an understanding of how much a node (in isolation) should compute to communicate the desired function within the network. Our results suggest that to effectively compute different function classes with different surjectivities, the networks can be restructured with the transition probabilities being tailored for functions, i.e., task-based link reservations, which can enable mixing versus separately processing of a diverse function class. We numerically evaluate our technique for search, MapReduce, and classification functions, and infer how sensitive the processing factor to the surjectivity of each computation task is.**

***Index Terms*—Entropic surjectivity, Little's law, computation processing factor, communications, sparse representation.**

I. Introduction

CHALLENGES in cloud computing include effectively distributing computation to handle the large volume of data with growing computational demand, and the limited air interface resources. Furthermore, various tasks such as computation, storage, communications are inseparable. In network computation is required for reasons of dimensioning, scaling and security, where data is geographically dispersed. We need to exploit the sparsity of data within and across sources, as well as the sparsity inherent to labeling, to provide approximately minimal representations.

An equivalent notion to that sparsity is that of data redundancy. Data is redundant in the sense that there exists, a possibly latent and ill understood, sparse representation of it which is parsimonious and minimal, and that allows for data reconstruction, possibly in an approximate manner. Redundancy can occur in a single source of data or across multiple sources.

Manuscript received March 1, 2021; revised June 28, 2021; accepted July 29, 2021. Date of publication August 2, 2021; date of current version September 20, 2021. This work was supported in part by the National Science Foundation under Grant CNS 2008639 and Grant CNS 2008624. A preliminary version appeared in Proc., IEEE Infocom 2020 [1]. *(Corresponding author: Derya Malak.)*

Derya Malak is with the Communication Systems Department, EURECOM, 06904 Sophia Antipolis, France (e-mail: deryamalak@gmail.com).

Muriel Médard is with the EECS Department, MIT, Cambridge, MA 02139 USA (e-mail: medard@mit.edu).

Digital Object Identifier 10.1109/JSAIT.2021.3101762

A. Motivation and Related Work

As computation becomes increasingly reliant on numerous, possibly geo-dispersed, sources of data, making use of redundancy across multiple sources without the need for onerous coordination across sources becomes increasingly important. The fact that a minimal representation of data can occur across sources without coordination is the topic of distributed compression. The core result is that of Slepian and Wolf [2], who showed that distributed compression without coordination is as efficient as compression with cooperation, in terms of asymptotic minimality of representation.

The use of the redundancy in both functions and data to provide sparse representations of functions is the topic of the rather nascent field of functional compression. A centralized scheme requires all data to be transmitted to some central unit to perform certain computations. However, in many cases such computations can be performed in a distributed manner at different nodes in the network avoiding transmission of unnecessary information, hence, significantly reducing the resource usage. This can help improve the trade-off between communications and computation.

Compressed sensing and information theoretic limits of representation provide a solid basis for function computation in distributed environments. We next review them briefly.

Distributed compression: Compressed sensing [3]–[5] is an alternate approach to coding for making use of sparsity to provide parsimonious representation and reconstruction of data within an allowed distortion level in a low-complexity and robust manner. It can replace joint source-channel-network coding for wireless networks [6], [7], and distributed sources [8]. Distributed compression has also been considered from an information theoretic perspective, e.g., for source coding, using syndromes in [9], and source-splitting techniques in [10]. There also exist some information theoretic limits, such as side information problem [11], [12], Slepian-Wolf coding for multi-depth trees [2], and general networks via multicast and linear network coding [13].

Functional compression: To capture underlying redundancy both in data and functions, and recover a sparse representation, or labeling, at the destination, the rate-region for functional compression via graph entropy has been derived in [14] and [15], and for the side information scenario in [16] and [17]. In [18] graph coloring approaches to distributed encoding in tree networks with zero distortion, and in [19] polynomial time sparse graph compression techniques have been devised. The encoding rate has been analyzed for

broadcast networks in [20], and random geometric graphs in [21]. Function computing has also been studied using multi-commodity flow techniques in [22], [23]. However, there do not exist a family of coding approaches for functional compression that are simple and robust and that approximate the information theoretic limits unlike the case for compression, where coding and compressed sensing techniques exist.

Computing capacity in networks: Computing capacity of a network code is the maximum number of times the target function can be computed per use of the network [24]. The maximum rate at which functions of sensor measurements can be computed and communicated to the sink node has been examined in [25]. A line of work considered function computing in networks for a specific set of functions, e.g., in [26] for symmetric Boolean functions in tree networks. The capacity for special cases such as trees, identity function [27], linear network codes to achieve the multicast capacity have been studied [28]. For scalar linear functions, the computing capacity has been fully characterized by min cut in [29]. For vector linear functions over a finite field, necessary and sufficient conditions have been obtained in [30] to compute the function. The rate (and distortion) region has been explored for communicating functions of correlated sources and Slepian-Wolf for partially invertible functions in [31], source-network codes for function computation in [32], and lossy linear function computation in [33]. For general functions and network topologies, upper cut-set bounds on the computing capacity based on cut sets have been studied in [26], [34]. In [24], authors have generalized the equivalence relation for the computing capacity. However, in these works, characterization based on the equivalence relation associated with the target function is only valid for special topologies, e.g., the multi-edge tree. For general networks, this equivalence relation is not sufficient to explore general computation problems.

Cost-performance tradeoffs and coded computing: Minimizing the communication load is essential in various frameworks, such as sensor networks [25], content replication and delivery [35], or a MapReduce algorithm [36]. Coded computing aims to tradeoff communications bottlenecks by injecting computations. This tradeoff in distributed computing has been examined in the framework of MapReduce [37], [38]. Distributed coded computing of linear reduce functions in multi-stage networks by exploiting multicast opportunities has made the communication load inversely proportional to the computation load in [37]. Nodes with heterogeneous processing capabilities have been studied in [39]. While fully distributed algorithms cause a high communication load, fully centralized systems suffer from high computation load. Ideally, we aim to find a middle ground to optimize the total cost of distributed computing in networks.

Queueing theoretic techniques: There exist approaches, such as [40], where authors have devised throughput-optimal algorithms for computing the class of fully-multiplexible functions, and [41], [42], where authors have designed decentralized routing techniques for a class of functions represented by a binary tree. Other works also incorporated queueing aspects for fusing sensor readings [43], or computing the maximum of them [44]. Existing adaptive techniques, such as the coding approaches developed in [45] for in-network computing, and [46] for data gathering in broadcast networks, may not be tailored to computing general nonlinear functions or more general topologies.

B. Contributions

In this paper, we provide a fresh look at the distributed function computation problem in a networked context. To the best of our knowledge, there are no constructive approaches to this problem except for special cases as outlined in Section I-A. As a first step to ease this problem, we will provide a utility-based approach for general cost functions. As special cases, we continue with simple examples of point search with running time $O(\log N)$ where N is the input size in bits, then MapReduce with $O(N)$, then the binary classification model with $O(\exp(N))$. Our main contribution is to provide the link between the computation problem and Little's law.

We devise a delay optimization framework for stationary Jackson network topologies for distributed function computing using a flow-based technique and by jointly capturing the costs of communications and computation of a general class of functions. Function outcomes can be viewed as colors on the characteristic graph of the function on the data, which is central to functional compression [18]. We characterize the complexity of functions via the joint entropy of the characteristic graphs and then leverage the joint graph entropy to define our key metric, namely the entropic surjectivity – a notion of sparsity inherent to functions. This model serves as a simple means of exploiting the function's entropic surjectivity by employing the concepts of graph entropy and Little's law to provide approximately minimal representations for computing. Our primary insight is that the core characteristics required for operating the distributed computation scheme are those associated with the entropic surjectivity of the functions.

The novelty of our approach comes from two aspects. First, we do not limit ourselves to a class of topology provided that the network is product-form. Jackson networks allow for the treatment of each node in isolation, independent of the topology. Hence, we do not constrain ourselves to cascade operations as in [6] due to the restriction of topology to linear operations. Second, we do not restrict the model to a specific class of functions. We use the notion of entropic surjectivity to generalize Little's law to computation. The enabler of our approach is the connection between Little's law and the proportion of flow that requires communications determined by entropic surjectivity. While the Jackson network is analyzed by applying Little's law and Markov routing policy, and the communication time is devised readily, to the best of our knowledge, the computing perspective has never been integrated into this problem before.

Our proposed approach does not put any assumptions on the topology and characterizes the functions only via their entropic surjectivity. The probabilistic reservation of the bandwidth (i.e., the Markov routing policy) is on the tasks that have different graph entropies. The distributed data compression scheme of Slepian-Wolf focuses on source compression

and is applicable under any protocol information [47]. Our model provides insights into distributing computation and allocating resources, where the flows of different functions can be processed separately or mixed.

We next detail the sketch of the steps and the organization for the rest of the paper.

C. Steps Towards Building the Computation Framework and Organization of the Paper

In Section II, we describe the networked multi-class stationary computation system, where each class represents a different function category. Each node has a computation queue to perform different computation tasks and a communication queue to serve different function classes. The main objective is to minimize the average aggregate delay cost of these function classes via balancing the computing and communication load of the network. To that end, in Section II-A, we start by reviewing the key elements of queueing theory, and in Section II-B, we present the layout of the multi-class network model. In Section II-C, we generalize Little's law in queueing theory, which relates the long-term average number of customers in a stationary system to the long-term average effective arrival rate and the average time that a customer spends in the system [48], to the computation framework. In our framework, the waiting time for a class of functions is equal to the average time a packet spends in computation and communication, which we evaluate using flow conservation principles. In Section II-D, we use the fundamental compression limits for functions given by the measure of graph entropy, which is an information-theoretic term for identifying the compression rate achievable in which pairs of source values potentially may be confused [14]. By linking the notion of graph entropy for computing functions to the generalized Little's law for computation, we derive lower bounds on the rate of generated flows, namely processing factors γ_f. We coin the term entropic surjectivity and denote it by Γ_c for function class c, to capture how well we can compress the function versus its domain, where the higher Γ_c is, the higher the cost of functional compression is. In Section II-E, using queuing and information-theoretic concepts, we provide the delay cost models for task completion.

In Section III, we detail the routing model for computation and distribution of multiple function classes while ensuring stability. More specifically, in Section III-A, we establish connections between processing factors γ_f, effective arrival rates, and routing matrix. In Section III-B, we elaborate on how the entropy rate for the Markov routing policy and mixing different function classes affect the average computation time. We infer that in a stationary network with transition probabilities being tailored for tasks, i.e., via task-based link reservations that ensure mixing different function classes instead of fixed routings, it is possible to compute functions more effectively.

In Section IV, we analyze the flow, derive load thresholds for computing, and devise cost optimization problems. More specifically, in Section IV-A, we formulate two optimization problems to determine the average aggregate delay cost of the network without intermediate computing versus that with intermediate computing via taking into account the stability, processing, and compression constraints. In Section IV-B, we explore the fundamental limits of the traffic loads associated with different computation tasks. We compute load thresholds for each node to determine whether a node should conduct computation or not. In Section IV-C, we consider a single class flow and investigate two examples with low and high task computation complexities, respectively, and how to effectively distribute these tasks leveraging the communication and computing resources and the computation complexity of the tasks. We show that low-complexity tasks, such as Search and MapReduce, may favor distributed processing. However, high-complexity tasks, such as Classification, may require centralized processing provided that the compute resources are sufficient. In Section IV-D, we explore how to allocate the compute tasks corresponding to multiple function classes by exploiting their processing factors.

In Section V, we present numerical results and evaluations to demonstrate the utility of the proposed function load distribution framework, and understand the load threshold for different function classes. Our main observations are that we should reserve most of the processing power for running simple tasks, such as MapReduce and Search, and less for high complexity tasks, such as Classification, to ensure cost-effective distributed computing. Furthermore, mixing flows can provide a more effective allocation of the processing resources and increase the cost savings over separating flows. In addition, as the processing capability increases, higher Γ_c is allowed, and the regime for which computation is allowed expands. When the communication delay is negligible, intermediate computing can further improve the overall cost.

In Section VI, we conclude the manuscript by discussing possible future directions.

II. A Networked Computation Model

In this section, we detail our network model that incorporates communications and computation. We consider a general stationary network topology. Sources can be correlated, and computations are allowed at intermediate nodes, and we are interested in computing a set of deterministic functions. While doing so our goal is to effectively distribute computation. To that end, intermediate nodes need to decide whether to compute or relay.

A. Main Elements From Queueing Theory

A collection of nodes V serves multiple classes of computation flows denoted by C. Multiple nodes can serve each class. External arrivals into classes occur according to independent Poisson processes with a rate β_v^c for $c \in C$ and $v \in V$. The total effective arrival rate at node v is λ_v. Node v first performs a compute service and a computation service for the function class c at rates χ_v^c and μ_v^c, respectively. After completing service at node v, a class c job will instantaneously become a class c' job and routed to node w with probability $p_{v,w}^{rou}(c, c')$. In Section III, we will determine the effective arrival rates at each node using the routing probabilities of the queueing

network, where the routing is Markovian. In Section III, we will also detail the flow conservation principles to guide us the principles in Section IV for determining the load thresholds for computing to optimize the average task completion time. In the sequel, we provide comprehensive descriptions of the building blocks of the multi-class networked computation framework.

B. A Product-Form Multi-Class Network Model

We assume a multiple-class open Jackson network of $|V|$ nodes with computing capabilities, in which packets can enter the system from an external/virtual source s, get processed and leave the system, i.e., are sent to a virtual destination 0. Let $G = (V, E)$ be a graph modeling the communication network where $V = \{v_0, v_1, \ldots, v_{|V|-2}, v_{|V|-1}\}$ represents the collection of all $|V|$ nodes where $v_0 = s$ and $v_{|V|-1} = 0$, and there is a directed link from $v \in V$ to $w \in V$ if $(v, w) \in E$. We also let $V' = V \backslash \{s, 0\}$. Multi-class networks of queues, quasi-reversible networks, networks in product-form, or networks with symmetric queues have been well explored in the literature, see for example [49, Ch. 3] and [50]. These classes of networks of queues can capture a broad array of queue service disciplines, including FIFO, LIFO, and processor sharing. A Jackson network exhibits a product-form equilibrium distribution and we can consider each node in isolation and investigate the steady-state properties of the network. A benefit of this is each node needs to know how much it needs to manage, which is less complicated than when nodes need the topological information to determine how to manage individual computational flows. Similar behavior is observed when a network of queues, such that each individual queue in isolation is quasi-reversible, always has a product-form stationary distribution [49].

Set of computation flows: For a set of functions $f \in \mathcal{F}$, we denote the set of computational flows by $C = \{c = (f, X)\}$ which represents the class of functions and $X \in \mathcal{X}$ defined on the probability space $(\mathcal{X}, \mathcal{P})$ where $\mathcal{X}$ is the set of symbols and $\mathcal{P}$ is the data (or source) distribution. Multiple classes represent different types of functions, computation for each class is done in parallel, and the network can do computations and conversions between different classes (which we detail in Section II-E). Let $C = |C|$ be the cardinality of all classes, and any function of the same class $c \in C$ has the same complexity. Hence, we denote a function f by f_c if it belongs to class c. The number of classes of functions that the network can handle will be a proxy for the resolution of network's computation capability, and $|C|$ being sufficiently large means that a wide class of functions can be computed/approximated.

Source data in bits: The incoming traffic intensity associated with class $c \in C$ packet at node v is λ_v^c follows a Poisson process with arrival rate λ_v^c in bits/sec. The discrete-time stochastic process modeling the source at node v is $\mathbf{X} = (X^{(1)}, \ldots, X^{(T)})$, where $X^{(t)}$ is a random variable representing a value observed at time t. The total arrival rate of the source $\mathbf{X}$ at v is $\lambda_v = \sum_{c \in C} \lambda_v^c = H(X)$ under the assumption that arrivals from different classes occur independently, where $H(X) = \lim_{T \to \infty} \frac{H(\mathbf{X})}{T}$ is the entropy rate or source information rate when the limit exists. Hence, the source at each v is characterized by the aggregate arrival process to the node, and is a mixture model with multiple classes. Because the function class c has an effective rate λ_v^c, its contribution to $H(X)$ depends on its proportion, i.e., $\frac{\lambda_v^c}{\sum_{c \in C} \lambda_v^c}$. However, we do not require that an observed value of the source $\mathbf{X}$ should identify the class to which it belongs.

An arrival to v associated with class c requires a compute service proceeded by a communication service at rate $\mu_v^c > \lambda_v^c$, where $\rho_v^c = \frac{\lambda_v^c}{\mu_v^c}$. Each processed packet departing from v has a Poisson distribution. We call the rate of flow generated by v by computing f_c, $c \in C$ the surjection factor of v and denote it by $\gamma_f(\lambda_v^c)$. The surjection factor $\gamma_f(\lambda_v^c)$ is monotone increasing in λ_v^c and $\gamma_f(\lambda_v^c) \leq \lambda_v^c$. We denote the long-term average of the aggregate number of packets in the compute and communication service at v due to processing of f_c by L_v^c.

We consider a network of quasi-reversible communication and computation queues. In this setting each node $v \in V$ maintains a computation queue and a communications queue.

To model the states of the computation queues, let $\mathcal{M}$ be the set of $v \in V$ quasi-reversible computation queues, m_v^c and $m_v = \sum_{c \in C} m_v^c$ be the number of packets of class $c \in C$ and the total number of packets in the compute queue of v, respectively. To model the computation queue state of v, let $\mathbf{m}_v = (c_1, \ldots, c_{m_v})$ where c_i is the class of packet $i = 1, \ldots, m_v$. We let the computation traffic intensity associated with class $c \in C$ at node $v \in V$ be σ_v^c. In the special case when an arriving packet requires a compute service which is drawn from an exponential distribution[1] with mean $\frac{1}{\chi_v^c}$ independently, its traffic intensity is $\sigma_v^c = \frac{\lambda_v^c - \gamma_f(\lambda_v^c)}{\chi_v^c}$. Hence, the total intensity at node v equals $\sigma_v = \sum_{c \in C} \sigma_v^c$. Utilization satisfies $\sigma_v^c < 1$ to ensure stability of computation. The global state of $\mathcal{M}$ is $\mathbf{m} = (\mathbf{m}_0, \mathbf{m}_2, \ldots, \mathbf{m}_{|V|-1})$. To ensure quasi-reversibility, we assume a M/M/1 network with $\chi_v^c = \chi_v$ for all c. Then the steady-state distribution of the compute queue of v has a product-form [51, Ch. 9] given by $\phi_v(\mathbf{m}_v) = (1 - \sigma_v) \prod_{c \in C} (\sigma_v^c)^{m_v^c}$. Hence, the stationary distribution of $\mathcal{M}$ is $\phi(\mathbf{m}) = \prod_{v \in V} \phi_v(\mathbf{m}_v)$. Once a packet's compute service at v is complete, it is then forwarded by the communication queue of v to the nodes $v \in V \backslash s$.

Let $\mathcal{N}$ be the set of $v \in V$ quasi-reversible communication queues, n_v^c and $n_v = \sum_{c \in C} n_v^c$ be the number of packets of class $c \in C$ and the total number of packets in the communication queue of v, respectively. The state of this queue is $\mathbf{n}_v = (c_1, \ldots, c_{n_v})$, where c_i is the class of packet $i = 1, \ldots, n_v$. A class c packet has a departure from the communication queue of v that follows a Poisson process with rate $\gamma_f(\lambda_v^c)$ and requires an exponential distributed communication

[1]Later in Section II-E we consider different computation models where utilization σ_v^c for computation can have different forms.

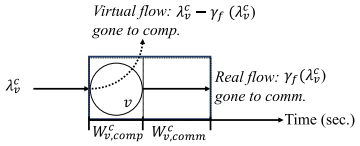

Fig. 1. Little's law in a computation scenario where $L_v^c = \gamma_f(\lambda_v^c) \cdot W_v^c$.

service with mean $\frac{1}{\mu_v^c}$ independently. Hence, the outgoing traffic intensity is $\rho_v^c(out) = \frac{\gamma_f(\lambda_v^c)}{\mu_v^c}$. Hence, the total intensity at v equals $\rho_v(out) = \sum_{c \in C} \rho_v^c(out)$. Utilization satisfies $\rho_v^c(out) < 1$, which suffices for stability. The stationary distribution of communication queue v is analyzed in isolation similarly as in compute queue and is $\pi_v(\mathbf{n}_v) = (1 - \rho_v(out)) \prod_{c \in C} (\rho_v^c(out))^{n_v^c}$. The steady-state distribution of the global state of $\mathcal{N}$ is $\pi(\mathbf{n}) = \prod_{v \in V} \pi_v(\mathbf{n}_v)$, where $\mathbf{n} = (\mathbf{n}_0, \mathbf{n}_2, \ldots, \mathbf{n}_{|V|-1})$.

Global state of the network is $[\mathbf{n}; \mathbf{m}] = [\mathbf{n}_v; \mathbf{m}_v]_{v \in V} \in \mathbb{R}^{n+m}$ where $n = \sum_{v \in V} n_v$, and $m = \sum_{v \in V} m_v$. We further let $\rho = [\rho_v^c]_{c \in C,\, v \in V}$, $\lambda = [\lambda_v^c]_{c \in C,\, v \in V}$, $\boldsymbol{\lambda}^c = [\lambda_v^c]_{v \in V \setminus \{s\}} \in \mathbb{R}^{(|V|-1) \times 1}$, $\mu = [\mu_v^c]_{c \in C,\, v \in V}$, $\boldsymbol{\mu}^c = [\mu_v^c]_{v \in V \setminus \{s\}} \in \mathbb{R}^{(|V|-1) \times 1}$, $\rho(out) = [\rho_v^c(out)]_{c \in C,\, v \in V}$, and $\sigma = [\sigma_v^c]_{c \in C,\, v \in V}$.

We next introduce a novel relation for computation by building a connection between the limits of functional compression rates and Little's law in queueing theory.

C. Computing With Little's Law

Little's law states that the long-term average number L of packets in a stationary system is equal to the long-term average effective arrival rate λ multiplied by the average time W that a packet spends in the system. More formally, it can be expressed as $L = \lambda W$. The result applies to any system that is stable and non-preemptive, and the relationship does not depend on the distribution of the arrival process, the service distribution, and the service order [48].

In our framework, each node is equipped with a compute queue and communications queue where the processing is sequential. Specifically, the incoming flow of rate λ_v^c first goes into the compute queue for processing to provide a compressed representation (for each class of function) which is then forwarded to the network at a rate $\gamma_f(\lambda_v^c)$. Because computation on a flow yields a reduction in the amount of flow to be transmitted, $\lambda_v^c > \gamma_f(\lambda_v^c)$. This is different from a standard communication where the incoming data is forwarded to the network without any computation on it and $\lambda_v^c = \gamma_f(\lambda_v^c)$. Since we employ a flow-based approach, from flow conservation, the difference $\lambda_v^c - \gamma_f(\lambda_v^c)$ models the virtual flow rate that is lost as a result of computation exerted by node v. This loss can be modeled as a function of a node's total self-loop flows [22], [23]. Let m_v^c and n_v^c be the long-term average number of packets of class c at v waiting for compute and communication service, respectively. Let $W_{v,comp}^c(m_v^c)$ and $W_{v,comm}^c(n_v^c)$ be the average time needed for the compute and communication operations at node v for c, respectively, that are positive and non-decreasing in their respective flows. Let W_v^c be the delay cost function that models the average time a data packet of class c spends at node v, which is given by[2]

$$W_v^c = W_{v,comp}^c(m_v^c) + W_{v,comm}^c(n_v^c). \tag{1}$$

The long-term average number of packets in node v for class c functions equals $L_v^c = m_v^c + n_v^c$. Given rates $\lambda_v^c - \gamma_f(\lambda_v^c)$ and $\gamma_f(\lambda_v^c)$, respectively for computation and communications, we can determine the relations among L_v^c, m_v^c, and n_v^c as function of $\gamma_f(\lambda_v^c)$:

$$m_v^c = L_v^c \cdot \left(1 - \frac{\gamma_f(\lambda_v^c)}{\lambda_v^c}\right), \quad n_v^c = L_v^c \cdot \frac{\gamma_f(\lambda_v^c)}{\lambda_v^c}. \tag{2}$$

We aim to infer the outgoing rate $\gamma_f(\lambda_v^c)$ generated from an incoming flow rate λ_v^c. By Little's law, the long-term average number L_v^c of packets at node v for class c functions satisfies

$$L_v^c = \gamma_f(\lambda_v^c) \cdot W_v^c. \tag{3}$$

In Section II-D we introduce a novel flow-based notion for computing that gives a lower bound on the surjection factor. We detail the delay cost function W_v^c in Section II-E.

D. Characterizing the Flow of Computation

The fundamental limits of compressing a function f_c in distributed networks are given by the graph entropy of f_c [14], [15], [16], [18]. Graph entropy generalizes the Shannon's entropy $H(\mathbf{X})$ by employing the notion of a characteristic graph. Each vertex of the characteristic graph represents a sample value of the same random variable, and two vertices are connected if they should be distinguished because the function f_c can get different values in these two points. Total incoming flow rate needed is approximated as $H(\mathbf{X})$. More precisely, let $f_c(\mathbf{X}^{(t)})$ be a function of a collection of source variables represented by the random vector through time $\mathbf{X}^{(t)} = \{X_1^{(t)}, \ldots, X_M^{(t)}\}$, where $\mathbf{X}^{(t)}$ can be a set of possibly correlated source samples of a multivariate random process, and sample $m \in \{1, 2, \ldots, M\}$ can take values from the alphabet $\mathcal{X}_m$. Dropping the superscript (t) in the source vector, to construct the characteristic graph of f_c on X_1, i.e., G_{X_1}, we draw an edge between vertices $u_1 \in \mathcal{X}_1$ and $u_2 \in \mathcal{X}_1$, if $f_c(u_1, x_2, \ldots, x_M) \neq f_c(u_2, x_2, \ldots, x_M)$ for any realization $x_2, \ldots, x_M$ whose joint instance has non-zero measure. Surjectivity of f_c determines the connectivity and hence coloring of the graph. The graph entropy $\boldsymbol{H}(f_c(\mathbf{X}))$ is the entropy rate of the coloring of G_{X_1} for the function f_c on $\mathbf{X}$ which characterizes the minimal representation needed to reconstruct with fidelity the desired function $f_c(\mathbf{X})$ [14]. In general, functional compression reduces the amount of rate needed to recover f_c on data $\mathbf{X}$, and the savings increase along $H(\mathbf{X}) \to \boldsymbol{H}(f_c(\mathbf{X}))$. We note that source 1 can achieve an encoding rate close to the graph entropy of X_1 such that $\boldsymbol{H}(f_c(\mathbf{X})) = H_{G_{X_1}}(X_1) = \lim_{n \to \infty} \frac{1}{n} \min_{c_{G_{X_1}^n}} H(c_{G_{X_1}^n}) \leq H(c_{G_{X_1}})$ [14], where $c_{G_{X_1}^n}$ is a valid coloring of $G_{X_1}^n$ which is the n^{th} power of a graph G_{X_1}. For further details on graph entropies we

[2]It is possible to generalize the sequential processing to pipelining where computation and communications progress together, and the parallelism ensures $W_v^c = \max[W_{v,comp}^c(m_v^c),\ W_{v,comm}^c(n_v^c)]$. We left the non-sequential model as a future direction.

refer the reader to [2], [14]–[16], [18]. The degenerate case of the identity function corresponds to having a complete characteristic graph. The fundamental limits of asymptotic compression in this case are given by the Slepian-Wolf theorem such that the sum rate of the sources should exceed $H(\mathbf{X})$, and X_1 can be asymptotically compressed up to the rate $H(X_1|\mathbf{X}\backslash X_1)$ when $\mathbf{X}\backslash X_1 = \{X_2, \ldots, X_M\}$ is available at the receiver [2]. In the sequel, instead of restricting to X_1, we consider the set of sources $\mathbf{X}$ and denote the joint graph entropy of $[G_{X_m}]_{m\in\{1,2,\ldots,M\}}$ by $\boldsymbol{H}(f_c(\mathbf{X}))$. To compute $f_c(\mathbf{X})$ sources collectively need to transmit at a minimum of $\boldsymbol{H}(f_c(\mathbf{X}))$.

In general, finding minimum entropy colorings $c_{G^n_{X_1}}$ of characteristic graphs G_{X_1} is NP-hard, and the optimal rate region of functional compression remains an open problem [52]. However, in some instances, it is possible to efficiently compute these colorings [18]. In [18], the sources compute colorings of high probability subgraphs of their G_{X_m}, $m \in \{1, 2, \ldots, M\}$ and perform Slepian-Wolf compression on the colorings and send them. Similarly, intermediate nodes compute the colorings for their parents', and use a look-up table to compute the corresponding functions.

The minimum functional compression rate attainable over a network can be obtained via extending the notion of graph entropy $\boldsymbol{H}(f_c(\mathbf{X}))$ and its rate $\boldsymbol{H}(f_c(X)) = \lim_{T\to\infty} \frac{\boldsymbol{H}(f_c(\mathbf{X}))}{T}$ to capture the topology as well as function's surjectivity. We next introduce a novel notion to measure the fundamental limits of compression in a network for a function the destination wants to compute.

Definition 1 (Entropic Surjectivity, Γ_c): Entropic surjectivity of $f_c : \mathbf{X} \to Y$, $c \in C$ is how well it can be compressed with respect to the compression rate of source symbols $\mathbf{X}$. We denote the entropic surjectivity of function f_c with respect to $\mathbf{X}$ by

$$\Gamma_c = \frac{\boldsymbol{H}(f_c(\mathbf{X}))}{H(\mathbf{X})}. \tag{4}$$

Note that Γ_c is maximized if the function f_c with domain $\mathcal{X}$ and codomain $\mathcal{Y}$ is surjective, i.e., for every $y \in \mathcal{Y}$ there exists at least one $\mathbf{x} \in \mathcal{X}$ with $f_c(\mathbf{x}) = y$. It is lower bounded by zero when the function maps all elements of $\mathcal{X}$ to the same element of $\mathcal{Y}$. Hence, Γ_c can be used as a measure of how surjective the function f_c is. A surjective function f_c has high entropy via compression through a network, and yields a high Γ_c, i.e., is harder to compress.

The proportion of flow that requires communications – the proportion of generated flow as a result of computing – needs to satisfy the following condition

$$\Gamma_c \leq \frac{\gamma_f(\lambda_v^c)}{\lambda_v^c}. \tag{5}$$

From flow conservation, the maximum amount of flow that vanishes due to computation is when (5) holds with equality, i.e., $\gamma_f(\lambda_v^c) = \lambda_v^c\Gamma_c$. In the regime of low compression, the network is communication intensive, i.e., $\gamma_f(\lambda_v^c)$ is high versus the computation intensive regime with high compression, i.e., low $\gamma_f(\lambda_v^c)$. In any regime, $\gamma_f(\lambda_v^c)$ has to be sufficiently large to ensure that the compute task is performed. More formally, employing the limits of functional compression

$$\Gamma_c\lambda_v^c \leq \gamma_f(\lambda_v^c) \leq \lambda_v^c < \mu_v^c. \tag{6}$$

To the best of our knowledge, there do not exist algorithms that exploit the theoretical limits provided by coloring characteristic graphs of functions to optimize the surjection factor $\gamma_f(\lambda_v^c)$. To that end, we aim to approximate $\gamma_f(\lambda_v^c)$ using the connection between Little's law that relates the number of packets L_v^c to function's entropic surjectivity Γ_c. To that end, we next characterize the average computing time of different function classes for the proposed network setting.

E. Cost Models for Task Completion Time

For the setup of a multi-class open Jackson network with node set V, all packets enter the system from a virtual source s. These packets are processed in the network and then leave the network via virtual destination 0. We next explore the breakdown for the compute and communications time of a packet and the average time it spends in the system.

Functional compression cost: We assume an average-case complexity model for computation where the algorithm does the computation in a sorted array (e.g., binary search). Let $d_{f_c}(m_v^c)$ denote the time complexity for computing functions of class $c \in C$ packet at $v \in V$ in the units of packets. While $d_{f_c}(m_v^c)$ increases in the function's complexity, its behavior is determined by the function class c, and as function of the input size m_v^c, i.e., the number of packets needed to represent the input. Hence, m_v^c denotes the number of packets of class c at v waiting for computation service. The compute cost at v of processing a flow of class c is the time complexity which describes the average running time for performing computation on a packet, in sec

$$W^c_{v,comp}(m_v^c) = \begin{cases} \frac{1}{\lambda_v^c} \cdot d_{f_c}(m_v^c), v \in V\backslash s,\ c \in C, \\ 0,\ \ v = s,\ c \in C. \end{cases} \tag{7}$$

This implies that the virtual source s does not perform computation services.

A taxonomy of functions: We consider three function categories and with different time complexities. For *Search* which tries to locate an element in a sorted array, an algorithm runs in logarithmic time, hence has low complexity. For *MapReduce*, the reduce functions of interest are linear, and the algorithm runs in linear time, which is of medium complexity. For *Classification*, we consider the set of all decision problems that have exponential runtime. The time complexity, i.e., the order of the count of operations, of these functions satisfies:

$$d_{f_c}(m_v^c) = \begin{cases} O(\log(m_v^c)), & \text{Search}, \\ O(m_v^c), & \text{MapReduce}, \\ O(\exp(m_v^c)), & \text{Classification}. \end{cases} \tag{8}$$

Using the order of the count of operations in (8) and $W^c_{v,comp}(m_v^c)$ in (7), we can model the delay cost functions for computations of different classes of functions for $v \in V'$. In Section V, we will numerically investigate the behavior of computing cost.

Communication cost: Each intermediate computation provides a compressed representation. The cost of communications equals the average waiting time, i.e., the sum of the

average queueing and service times of a packet in sec. It is given as a function of the departing flow:

$$W^c_{v,comm}(n^c_v) = \begin{cases} \frac{1}{\mu^c_v - \gamma_f(\lambda^c_v)}, & v \in V', \ c \in C, \\ \frac{1}{\mu^c_v - \beta^c}, & v = s, \ c \in C, \\ 0, & v = 0, \ c \in C, \end{cases} \tag{9}$$

which is convex in $\gamma_f(\lambda^c_v)$, hence increasing in $n^c_v = \frac{\gamma_f(\lambda^c_v)}{\mu^c_v - \gamma_f(\lambda^c_v)}$ and upper bounded by $\frac{1}{\mu^c_v(1-\rho^c_v)}$ because $\rho^c_v(out) \leq \rho^c_v$. When $\gamma_f(\lambda^c_v) = \lambda^c_v$ since no packets experience compute service we have $m^c_v = 0$, and $n^c_v = \frac{\rho^c_v}{1-\rho^c_v}$. The computation time of the identity function is null $W^c_{v,comp}(0) = 0$. As a result $\rho^c_v(out) = \rho^c_v$ and $W^c_{v,comm}(n^c_v) = \frac{1}{\mu^c_v - \lambda^c_v}$. It follows from (9) that $W^c_{v,comm} = 0$ when $v = 0$, $c \in C$. From (2) m^c_v increases in λ^c_v because increases always at a smaller rate than the incoming flow rate does. This in turn increases $d_{f_c}(m^c_v)$ and $W^c_{v,comp}(m^c_v)$. Furthermore, the long-term average L^c_v in (3) increases in λ^c_v, keeping μ^c_v fixed. Hence, (3) determines the behavior of $W^c_{v,comm}(n^c_v)$. As a result n^c_v increases if the scaling of $\gamma_f(\lambda^c_v)$ in λ^c_v is linear, versus n^c_v is less sensitive when the scaling is sublinear. We also note that the communication cost could instead be chosen to model the queue size, utilization, or the queueing probability. Our model does not account for the physical data transmission time, excluding the link layer aspects.

In the case when the computation cost is similar as the communication cost model, a compute service for each packet $c \in C$ arriving to v has an exponential distribution with mean $\frac{1}{\chi^c_v}$ independently. Hence, the utilization of class c packets at node v is $\sigma^c_v = \frac{\lambda^c_v - \gamma_f(\lambda^c_v)}{\chi^c_v}$ and $m^c_v = \frac{\sigma^c_v}{1-\sigma^c_v}$. In this special case, the compute cost for $v \in V'$ is given by the following convex model:

$$W^c_{v,comp}(m^c_v) = \frac{1}{\chi^c_v(1-\sigma^c_v)} = \frac{m^c_v}{\chi^c_v \sigma^c_v} = \frac{m^c_v}{\lambda^c_v - \gamma_f(\lambda^c_v)} \text{ sec}, \tag{10}$$

which generalizes the cost of MapReduce in (8) due to the dependence of σ^c_v on m^c_v. This model is yet to account for the routing information. In Section III we will focus on balancing different function classes, their conversions, and routings, to generalize (10) to a departure-based model.

III. Task Load Balancing

In this section, we detail the routing model for different function classes and their conversions. At each node, computation is followed by communication, as modeled in (1), while satisfying the stability conditions. In general packets can change their classes when routed from one node to another [50]. Hence, at each node, conversion between classes is allowed. Let $C^{in}_v \subseteq C$ and C^{out}_v be the set of incoming and outgoing classes at node $v \in V$. We assume that $C^{out}_v \subseteq C^{in}_v \subseteq C$ as computations on C^{in}_v can only reduce the number of function classes in the outgoing link. Each computation transforms a class $c \in C$ of flow into another class $c' \in C$ of flow where the rates of transitions among the classes are known [50, Ch. 10.3.8]. Because $C^{out}_v \subseteq C^{in}_v$, a computation transforms a flow class c_i into a flow class c_j if $j \geq i$. However, vice versa, conversion from c_j to c_i, violates the purpose of computation because c_i has a higher complexity and processing of c_j data may not increase its entropy. In other words it holds that

$$\boldsymbol{H}(f_{c_j}(\mathbf{X})) \leq \boldsymbol{H}(f_{c_i}(\mathbf{X})), \quad j \geq i. \tag{11}$$

We characterize the class transformations via a Markov routing policy which we detail next.

A. Routing and Flow Conservation

Let the original arrival rate of class c packets to the network be Poisson with rate β^c. Let $p^{arr}_v(c) = p^{rou}_{s,v}(c)$ be the probability that an arriving class c packet is routed to queue v, where s represents a virtual (shared) source node that denotes the origin of a task. Assuming that all arriving packets are assigned to a queue, we have that $\sum_{v \in V} p^{rou}_{s,v}(c) = 1$.

We allow packets to depart from the network after service completion. There is a virtual (shared) destination node $0 \in V$ that indicates the completion of a task. Let $p^{dep}_v(c) = p^{rou}_{v,0}(c)$ denote the probability that a function class c packet departs upon completing its service at v. The total departure rate of class c packets from v in the forward process is $\gamma_f(\lambda^c_v) \cdot p^{rou}_{v,0}(c)$. With these definitions, the routing probabilities between nodes have the following structure:

$$\begin{aligned} p^{rou}_s(c) &= \left[p^{rou}_{s,v}(c)\right]_{v \in V \setminus \{s\}} \\ &= \left(\left[p^{rou}_{s,v_1}(c) p^{rou}_{s,v_2}(c) \ldots p^{rou}_{s,v_{|V|-2}}(c) p^{rou}_{s,0}(c)\right]\right)^{\mathsf{T}} \\ &\in [0,1]^{(|V|-1)\times 1}. \end{aligned}$$

For $v \in V'$ we have the following upper-triangular matrix which is a valid stochastic matrix:

$$p^{rou}_v(c) = [p^{rou}_{v,w}(c, c')]_{w \in V \setminus \{s\},\, c' \in C} \in [0,1]^{(|V|-1)\times|C|}, \tag{12}$$

where (11) justifies the upper-triangular structure except for the sink node.

The network accommodates different types of state transitions, i.e., arrival, departure, routing, and internal state transitions [50, Ch. 10.6.3]. As a result of function computation, a packet can change its class when routed from one node to another where the routing probabilities depend on a packet's class. We assume a Markov routing policy [50, Ch. 10.6.2] which is described as follows. A class $c \in C$ packet that finishes service at $v \in V'$ is routed to the compute queue of node $w \in V'$ as a class $c' \in C$ packet with probability $p^{rou}_{v,w}(c, c')$ if its service is not completed. The probability that a class c packet departs from the network after service completion at v is

$$p^{rou}_{v,0}(c) = 1 - \sum_{w \in V'} \sum_{c' \in C} p^{rou}_{v,w}(c, c'), \tag{13}$$

where the second term in the right hand side denotes the total probability that the packets stay in the network. Since it is an open network model, for every class c there is at least one value of v so that $p^{rou}_{v,0}(c) > 0$. Thus all packets eventually leave the system.

A node $v \in V$ can forward the flows it receives through its incoming edges, or generate a flow of class $c \in C$ modeled via a self-loop by terminating equal amounts of incoming flows of class $c' \in C$ where conversion among classes is possible. In the stationary regime the total arrival rate λ_v^c of class c packets to node v is the lumped sum of the flows of class c packets through its incoming edges. The terminated flow due to computation is the difference $\lambda_v^c - \gamma_f(\lambda_v^c)$ between λ_v^c and the generated flow rate. Flow conservation for each class implies the following relation

$$\lambda_v^c = \begin{cases} \beta^c, \ v = s, \\ \beta_v^c + \sum\limits_{w \in V'} \sum\limits_{c' \in C} \gamma_f(\lambda_w^{c'}) p_{w,v}^{rou}(c', c), \ v \in V', \\ \sum\limits_{v \in V'} \gamma_f(\lambda_v^c) p_{v,0}^{rou}(c), \ v = 0, \end{cases} \tag{14}$$

where $\beta_v^c = \beta^c \cdot p_{s,v}^{rou}(c)$ denotes the original arrival rate of class c packets assigned to the compute queue of node v and is known a priori, and the second term in the right hand side denotes the aggregate arrival rate of packets that are routed to queue v as a class c packet after finishing service at other queues $w \in V$ as a class $c' \in C$. The term $\gamma_f(\lambda_w^{c'})$ denotes the total departure rate of class c' packets from node w rate of class c' packets to queue w (as a result of computation).

The processing factor $\gamma_f(\lambda_v^c)$ satisfies the lower bound in (5), ensuring that the flows routed from other nodes through the incoming edge of v is at least $\lambda_v^c \Gamma_c$. Using (12) we let $P^{rou}(c) = [p_v^{rou}(c)]_{v \in V'} \in [0,1]^{(|V|-2)\times(|V|-1)\times|C|}$ be a three dimensional routing array. We further let

$$\begin{aligned} \boldsymbol{p}^{rou}(c) &= \sum_{w \in V'} p_w^{rou}(c) \mathbf{1}_{|C|} \\ &= \left[\sum_{w \in V'} \sum_{c' \in C} p_{w,v}^{rou}(c, c') \right]_{v \in V \setminus \{s\}} \in \mathbb{R}^{(|V|-1)\times 1}, \end{aligned}$$

where $\mathbf{1}_n$ is a unit column vector of size n. Letting $\boldsymbol{\beta}^c = [\beta_v^c]_{v \in V \setminus \{s\}} \in \mathbb{R}^{(|V|-1)\times 1}$ and $\boldsymbol{\gamma}_f(\boldsymbol{\lambda}^c) = [\gamma_f(\lambda_v^c)]_{v \in V \setminus \{s\}} \in \mathbb{R}^{(|V-1|)\times 1}$ and using (14), we can rewrite (5) in vector form:

$$\begin{aligned} \boldsymbol{\gamma}_f(\boldsymbol{\lambda}^c) \geq \boldsymbol{\lambda}^c \Gamma_c &= \left[\boldsymbol{\beta}^c + \boldsymbol{p}^{rou}(c) \odot \boldsymbol{\gamma}_f(\boldsymbol{\lambda}^c) \right] \Gamma_c \\ &= \left(I_{|V|-1} - D_c \Gamma_c \right)^{-1} \boldsymbol{\beta}^c \Gamma_c, \quad \forall c \in C, \end{aligned} \tag{15}$$

where $\odot$ is the Hadamard (or element-wise) product, $I_{|V|-1} = \text{diag}(1, 1, \ldots, 1)$ is an $(|V|-1) \times (|V|-1)$ identity matrix, and $D_c = \text{diag}(\boldsymbol{p}^{rou}(c)) \in \mathbb{R}^{(|V|-1)\times(|V|-1)}$ is a matrix obtained by diagonalizing vector $\boldsymbol{p}^{rou}(c)$. The range of $\boldsymbol{\lambda}^c$ can be computed as function of Γ_c as determined by the function's surjectivity and its compute cost. Combining (15) which gives $\boldsymbol{\lambda}^c \geq \boldsymbol{\beta}^c + D_c \boldsymbol{\lambda}^c \Gamma_c$ and (14) which gives $\boldsymbol{\lambda}^c \leq \boldsymbol{\beta}^c + D_c \boldsymbol{\lambda}^c$ yields the following lower and upper bounds, respectively: $(I_{|V|-1} - D_c \Gamma_c)^{-1} \boldsymbol{\beta}^c \leq \boldsymbol{\lambda}^c \leq (I_{|V|-1} - D_c)^{-1} \boldsymbol{\beta}^c$.

In Fig. 2, we consider a node in isolation to illustrate the computational and communication flows at a typical node $v \in V'$ of the Jackson network. The min-cut that denotes the total arrival rate of computational flow c is λ_v^c. This via (14) captures the rate of original arrivals which is β_v^c, and the arrivals routed from any other node $w \in V'$ in the network. If there is no w such that $p_{w,v}^{rou}(c', c) > 0$, then $\lambda_v^c = \beta_v^c$. The cut $\gamma_f(\lambda_v^c)$ denotes the total generated rate (or processing factor) of computational flow c at node v. The processed flow can be routed to any $u \in V'$ in the network if $p_{v,u}^{rou}(c, c') > 0$. If there is no such node, then $\gamma_f(\lambda_v^c)$ departs the system.

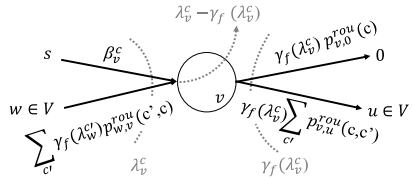

Fig. 2. Computational flow breakdown at $v \in V'$ where min cut λ_v^c is the total arrival rate of flow c at v that incorporates the original arrivals β_v^c and the arrivals routed from $w \in V'$. Min cut $\gamma_f(\lambda_v^c)$ is the total generated rate of c, at v, and the departures are routed to $u \in V'$.

For the networked setting we can refine the compute cost in (10) as

$$W_{v,comp}^c(m_v^c) = \frac{p_{v,0}^{rou}(c) m_v^c}{\lambda_v^c - \gamma_f(\lambda_v^c)} + \sum_{c' \in C} \sum_{w \in V} \frac{p_{v,w}^{rou}(c, c') m_v^c}{\lambda_v^c - \gamma_f(\lambda_v^c)}, \tag{16}$$

where the first term in the right hand side captures the wasted computation rate due to departing packets of class c from v of the network, and the second term represents the cost of additional processing due to the routing of the packets to other queues after finishing service from v.

B. Higher Order Properties

The communication network $\mathcal{N}$ under the Markov routing policy can be modeled using a discrete-time Markov chain (DTMC) $Y_1, Y_2, Y_3, \ldots$ The entropy rate for (Y_k) on a countable number of states is $H(Y) = -\sum\limits_{ij} \pi_i P_{ij} \log P_{ij}$ where π_i is the limiting distribution and (P_{ij}) is the state transition matrix. The rate $H(Y)$ can capture the routing information and characterize the distributions, i.e., the higher order properties, of the costs $W_{v,comm}^c$ and $W_{v,comp}^c$ incurred being in each state of the network. It can also explain the relation between routings and connectivity and how long it will take to compute. To that end, we next associate the states of the DTMC to a slotted model to determine the computing cost on a time-slot basis.

At each time slot t, the Markov chain is in state i with probability π_i. We infer the following quantities at t, each indicated by the superscript $^{(t)}$ as function of the state i:

1) $n_v^{c(t)}$ and $\gamma_f(\lambda_v^{c(t)})$, and $W_{v,comm}^{c(t)}(n_v^{c(t)})$ using (9),
2) $m_v^{c(t)} = n_v^{c(t)}(\frac{\lambda_v^{c(t)}}{\gamma_f(\lambda_v^{c(t)})} - 1)$ using (2), where $L_v^c = m_v^c + n_v^c$,
3) $W_{v,comp}^{c(t)}(m_v^{c(t)})$ using (7), and
4) $W_v^{c(t)} = W_{v,comp}^{c(t)} + W_{v,comm}^{c(t)}$ using (1), and the relations of $W_v^{c(t)}$, L_v^c, and $\gamma_f(\lambda_v^{c(t)})$ via (3).

A class of function is easy to compute if the generated rate per unit time of computation is high. More precisely, in a DTMC with fixed slot duration T, the generated flow rate across $t_{\max}$ slots is $T \sum\limits_{t=1}^{t_{\max}} \gamma_f^{(t)}(\lambda_v^{c(t)})$ in bits/sec. If this rate is large, f_c is easier to compute for given routings. Hence, this approach helps infer what classes of functions can be computed easily. If on the other hand, to account for the cost of computing more precisely, we assume a continuous-time chain, we can use a DTMC Y_k to describe the k^{th} jump of

the process where the variables $S_1, S_2, S_3, \ldots$ describe holding times in each state. In state i, the slot duration $W_v^{c(t)}$ (holding time) can be approximated by the average time needed to perform computation on the distribution of classes, $n_v^{c(t)}$, $c \in C$ in the current state, i.e., $\mathbb{E}[W_i]$. Hence, $W_v^{c(t)}$ is the realization of the holding time sampled from an exponential distribution with rate $\frac{1}{\mathbb{E}[W_i]}$. Then the total number of bits generated over $t_{\max}$ slots is $\sum_{t=1}^{t_{\max}} W_v^{c(t)} \cdot \gamma_f^{(t)}(\lambda_v^{c(t)})$ and the total cost is $\sum_{t=1}^{t_{\max}} W_v^{c(t)}$.

Given routings P_{ij}, the limiting distribution π_i determines the network state $[\mathbf{n}; \mathbf{m}]$, which provides insights into computing different classes of functions. To that end we seek the relation between $H(Y)$ and $\boldsymbol{H}(f_c(\mathbf{X})) = \Gamma_c H(\mathbf{X})$. The entropy rate of $\{\gamma_f(\lambda_v^c)\}_{v\in V, c\in C}$ is the same as $H(Y)$ because the DTMC (Y_k) fully describes $\gamma_f(\lambda_v^c)$. Thus, $H(Y)$ determines how balanced the flows for the different classes are, and can be a proxy for communication cost. If the process (Y_k) is i.i.d., $H(Y) = H(Y_k)$, $k = 1, \ldots, N$, the class distribution across different states is not distinguished, i.e., $\gamma_f(\lambda_v^c)$ is unchanged in $c \in C$. On the other hand, mixing tasks f_c of different complexities lower $H(Y)$ (due to concavity of entropy, i.e., $H(Y) \leq -\sum_j (\sum_i \pi_i P_{ij}) \log(\sum_i \pi_i P_{ij})$.) Hence, the DTMC states are not visited with the same frequency, i.e., π_i is not uniform, as some tasks require high n_v^c and $\gamma_f(\lambda_v^c)$. This may help reduce the use of communication resources. Mixing function classes can provide resource savings and help lower $H(Y)$ in networked environments. Hence, in a Jackson network with nodes having similar routing probabilities and each state being visited as often, it might not be possible to effectively compute different function classes versus in a more structured network with transition probabilities being tailored for tasks, i.e., task-based or weighted link reservations, to enable mixing of classes.

Next in Section IV we characterize the thresholds on $\gamma_f(\lambda_v^c)$ for successfully computing functions.

IV. Computation Flow Analysis

In the previous section, we explored how given routings or connectivity informs us of the class of functions we can compute. In this section we are interested in the reverse question of how to optimize the routings and how the connectivity looks like to compute a given class of functions. To that end we investigate where to compute and how to compute a function class f_c of known time complexity to optimize the average computation time in the Jackson network. Our goal is to understand whether a divide-and-conquer-based approach is more favorable than a centralized computation allocation approach. In divide-and-conquer a subset of nodes work on the sub-problems of a given task, which are to be combined to compute the target value either at any node, including the destination. In a centralized approach tasks are not split into sub-problems. Instead they are run at once. We next introduce our optimization framework.

A. Optimizing Average Task Completion Time

In this part we focus on optimal distribution of multiple classes of functions where we allow class conversions, via exploiting the information-theoretic limits of function computation and flow conservation principles. We contrast two centralized formulations to optimize the cost function aggregated over $c \in C$, $v \in V$ to understand the savings of computation in a network setting.

Average cost of no computing: We first formulate an optimization problem where there is no intermediate computing on the source data, and the per node cost $W_{v,comm}^c(n_v^c)$ is given in (9). This boils down to minimizing the aggregate communications cost, and is formulated as

$$\text{CommsCost:} \min_{\rho(out)<1} \sum_{v\in V} \sum_{c\in C} W_{v,comm}^c(n_v^c), \tag{17}$$

where the routing array $P^{rou}(c)$ is fixed. The solution to (17) is given by

$$\text{CommsCost} = \sum_{c\in C} \frac{1}{\mathbf{1}_{|V|-1}^{\mathsf{T}} (\boldsymbol{\mu}^c - (I_{|V|-1} - D_c)^{-1} \boldsymbol{\beta}^c)}.$$

Average cost of function computation: We next formulate an optimization problem accounting for the time complexities of computation tasks in (8) and the communication costs:

$$\begin{aligned} \text{MinCost:} \min_{\rho(out),\,\sigma} \quad & \sum_{v\in V} \sum_{c\in C} W_v^c \\ \text{s.t.} \quad & \rho_v^c(out) < 1,\ \sigma_v^c < 1, \quad \forall c \in V,\ v \in V, \\ & \boldsymbol{\mu}^c > \boldsymbol{\lambda}^c \geq \boldsymbol{\gamma}_f(\boldsymbol{\lambda}^c), \quad \forall c \in C, \\ & \boldsymbol{\gamma}_f(\boldsymbol{\lambda}^c) \geq \boldsymbol{\lambda}^c \Gamma_c \geq \mathbf{0}_{|V|-1}, \quad \forall c \in C, \end{aligned} \tag{18}$$

where the objective function W_v^c, as defined in (1), captures the total delay to be minimized over the $C \times |V|$-variable matrices $\rho(out)$ and σ. This framework captures in addition to the first constraint on stability, the second constraint captures the capacity constraint $\boldsymbol{\mu}^c > \boldsymbol{\lambda}^c$, the processing constraint $\boldsymbol{\lambda}^c \geq \boldsymbol{\gamma}_f(\boldsymbol{\lambda}^c)$, the constraint on surjection factor $\boldsymbol{\gamma}_f(\boldsymbol{\lambda}^c) \geq \boldsymbol{\lambda}^c \Gamma_c$, and the non-negativity $\boldsymbol{\lambda}^c \Gamma_c \geq \mathbf{0}_{|V|-1}$ constraint, where the relations among flow generation, conservation, and termination given in (14) precisely give the connection between λ_v^c and $\gamma_f(\lambda_v)$.

The values of Γ_c, λ_v^c, and $\gamma_f(\lambda_v^c)$ are coupled. It is intuitive that $W_{v,comp}^c(m_v^c)$ increases with $d_{f_c}(m_v^c)$ for any function class, and λ_v^c determines the rate of increase. As Γ_c increases, from (5) the nodes generate a higher $\gamma_f(\lambda_v^c)$ because compression is harder as the entropy $\boldsymbol{H}(f_c(\mathbf{X}))$ grows, and $W_{v,comm}^c(n_v^c)$ gets higher. However, the exact behavior of MinCost is determined by the complex relationships between the flows. Hence, the connection between $\gamma_f(\lambda_v^c)$ and $d_{f_c}(m_v^c)$ is not immediate. For example, while addition function has a low complexity $d_{f_c}(m_v^c)$ and a high entropy $\boldsymbol{H}(f_c(\mathbf{X}))$, multiplication function has a high $d_{f_c}(m_v^c)$ and a low $\boldsymbol{H}(f_c(\mathbf{X}))$.

Given the parametrizations $W_{v,comm}^c(n_v^c)$, $W_{v,comp}^c(m_v^c)$, and $d_{f_c}(m_v^c)$ of different classes, we can solve MinCost for the optimal values of $\rho(out)$, σ that minimize MinCost in (18) by determining the optimal values of $\gamma_f(\lambda_v^c)$ and λ_v^c. Then, using (5), and mapping Γ_c to the function class f_c, we can

infer the type of flows (i.e., functions) that we can compute effectively.

MinCost may be non-convex and a global optimal solution may not exist or it might be NP-hard in some instances. To find the local minima of MinCost for general computation cost functions, we can use the Karush-Kuhn-Tucker (KKT) approach in nonlinear programming [53], provided that some regularity conditions are satisfied [54]. Allowing inequality constraints, KKT conditions determine the local optimal solution. However, due to the lack of strong duality results for non-convex problems, MinCost can indeed be NP-hard in some instances.

CommsCost in (17) gives an upper bound to MinCost (18), which is jointly determined by the communication and compute costs as well as Γ_c. As the computation becomes more costly, the sources may only be partially compressed in order to optimize MinCost. We also infer that there is no computing beyond some Γ_c because allocating resources to computation no longer incurs less cost than CommsCost. A node can perform computation and forward (route) the processed data if the range of Γ_c that allows compression is flexible. This is possible when computation is cheap. However, if a node's compression range is small, i.e., when computation is very expensive, then the node only relays most of the time. While computing at the source and communicating the computation outcome might be feasible for some function classes, it might be very costly for some sets of functions due to the lack of cooperation among multiple sources. By making use of redundancy of data across geographically dispersed sources and the function to be computed, it is possible to decide how to distribute the computation in the network.

To demonstrate the achievable gains in MinCost via computation over a communications only scheme we will run some experiments in Section V both by (i) separating and (ii) mixing different classes of flows (Search, MapReduce, and Classification) without precisely optimizing (18).

B. Load Thresholds for Computing

In this part we explore the fundamental limits of the traffic intensities or loads associated with different computation tasks. By contrasting the optimization formulations in (17) and (18) which model the optimized average costs of no computing $\sum_{v\in V}\sum_{c\in C} W^c_{v,comm}(\frac{\rho^c_v}{1-\rho^c_v})$ versus with computing $\sum_{v\in V}\sum_{c\in C}(W^c_v)^*$, respectively, we decide whether a node conducts computation or not.

We note that MinCost may not have a globally optimal solution, and therefore propose a local solution where each node decides whether to perform computation or not by comparing the local costs $W^c_{v,comm}(\frac{\rho^c_v}{1-\rho^c_v})$ and $(W^c_v)^*$. A sufficient condition for $v \in V$ to perform computation on $c \in C$ is $(W^c_v)^* < W^c_{v,comm}(\frac{\rho^c_v}{1-\rho^c_v})$. If $(W^c_v)^* > W^c_{v,comm}(\frac{\rho^c_v}{1-\rho^c_v})$ then the intermediate computations no longer provide savings and they instead overwhelm the cost. In that case node v does not perform computation on the class c flow with rate λ^c_v and instead forwards it, hence $\gamma_f(\lambda^c_v)$ is set to be λ^c_v because the flow rate of computation equals $\lambda^c_v - \gamma_f(\lambda^c_v) = 0$. To that end we define a threshold on traffic intensity ρ_{th} such that for the range $\rho^c_v(out) = \frac{\gamma_f(\lambda^c_v)}{\mu^c_v} \le \rho_{th}$ computation is allowed as $(W^c_v)^* \le W^c_{v,comm}(\frac{\rho^c_v}{1-\rho^c_v})$, and no computation is allowed for $\rho^c_v(out) > \rho_{th}$.

Proposition 1 (A Load Threshold for Distributed Computing): A node $v \in V$ can perform computation of a class $c \in C$ function if the following condition is satisfied:

$$\rho_{th} = \min_{\rho^c_v \ge 0}\left[\rho^c_v \;\middle|\; \frac{(\rho^c_v)^2}{1-\rho^c_v} > d_{f_c}(m^c_v)\frac{1-\rho^c_v\Gamma_c}{1-\Gamma_c}\right], \qquad (19)$$

where $\rho_{th} \to 1$ as $m^c_v \to \infty$ since $d_{f_c}(m^c_v)$ monotonically increases in m^c_v.

Note that in (19) $d_{f_c}(m^c_v)$ is a function of m^c_v, whereas m^c_v is a function of $\gamma_f(\lambda^c_v)$, and thus a function of ρ^c_v. In other words, the dependence between ρ^c_v and $d_{f_c}(m^c_v)$ in Prop. IV-B is hidden.

Proof: The threshold ρ_{th} is obtained by comparing the delays with and without computation, on a per node basis. If the delay caused only by communication $W^c_{v,comm}(\frac{\rho^c_v}{1-\rho^c_v})$ is higher than the total delay of computation followed by communication $(W^c_v)^*$, i.e., the condition $\frac{1}{\mu^c_v(1-\rho^c_v)} > \frac{1}{\lambda^c_v}d_{f_c}(m^c_v) + \frac{1}{\mu^c_v(1-\rho^c_v\Gamma_c)}$ holds at $v \in V'$, where $\gamma_f(\lambda^c_v) = \lambda^c_v\Gamma_c$, then v decides to compute. ∎

We next provide a sufficient condition for the stability of the networked computation model.

Proposition 2: The computation network is stable if the computation delay is at least as much as the communications delay, i.e., if the following condition is satisfied:

$$d_{f_c}(m^c_v) \ge \frac{\lambda^c_v}{\mu^c_v - \gamma_f(\lambda^c_v)} \ge n^c_v, \quad c \in C,\ v \in V.$$

Proof: For stability, the long-term average number of packets in the communications queue of v, i.e., n^c_v, should be upper bounded by the long-term average number of packets in the compute queue of v, i.e., m^c_v. Assume that $\frac{1}{\lambda^c_v}d_{f_c}(m^c_v) \ge \frac{1}{\mu^c_v}$. If this were incorrect, we would have $d_{f_c}(m^c_v) < \frac{\lambda^c_v}{\mu^c_v} < \frac{\lambda^c_v}{\mu^c_v - \gamma_f(\lambda^c_v)}$, and if $d_{f_c}(m^c_v) < \frac{\lambda^c_v}{\mu^c_v-\gamma_f(\lambda^c_v)}$, then the following relation would hold:

$$W^c_{v,comp}(m^c_v) = \frac{d_{f_c}(m^c_v)}{\lambda^c_v} < W^c_{v,comm}(n^c_v) = \frac{1}{\mu^c_v - \gamma_f(\lambda^c_v)}.$$

In this case, $n^c_v = \frac{\gamma_f(\lambda^c_v)}{\mu^c_v-\gamma_f(\lambda^c_v)}$ will accumulate over time, which violates stability. Hence, delay of computation should be higher, i.e., $W^c_{v,comp}(m^c_v) = \frac{d_{f_c}(m^c_v)}{\lambda^c_v} \ge W^c_{v,comm}(n^c_v) = \frac{1}{\mu^c_v-\gamma_f(\lambda^c_v)}$. ∎

We next bound the long-term average number of packets in the networked computation model.

Proposition 3 (An Extension of Little's Law to Computing): The long-term average number L^c_v of packets in node v for class c flow with time complexity $d_{f_c}(m^c_v)$ is bounded as

$$L^c_v \ge b^{c-}_v \cdot \left[\frac{d_{f_c}(m^c_v)}{\lambda^c_v} + \frac{1}{\mu^c_v - b^{c-}_v}\right],$$
$$L^c_v \le b^{c+}_v \cdot \left[\frac{d_{f_c}(m^c_v)}{\lambda^c_v} + \frac{1}{\mu^c_v - b^{c+}_v}\right],$$

where $(b_v^c)^{\pm}$ satisfies $2(b_v^c)^{\pm} = \lambda_v^c(1+\frac{1}{d_{f_c}(m_v^c)}) + \mu_v^c \pm \sqrt{(\lambda_v^c(1+\frac{1}{d_{f_c}(m_v^c)})+\mu_v^c)^2 - 4\lambda_v^c\mu_v^c}$.

Proof: Using Little's law in (3) we can upper bound the time complexity of computation as

$$d_{f_c}(m_v^c) \le L_v^c = \gamma_f(\lambda_v^c)\left[\frac{d_{f_c}(m_v^c)}{\lambda_v^c} + \frac{1}{\mu_v^c - \gamma_f(\lambda_v^c)}\right].$$

Rearranging the above term for $2a_v^c = \lambda_v^c(1+1/d_{f_c}(m_v^c)) + \mu_v^c$ we obtain the following relation: $\gamma_f(\lambda_v^c)^2 - 2a_v^c\gamma_f(\lambda_v^c) + \lambda_v^c\mu_v^c \le 0$ that gives the following range for $\gamma_f(\lambda_v^c)$:

$$\left[a_v^c - \sqrt{(a_v^c)^2 - \lambda_v^c\mu_v^c},\ a_v^c + \sqrt{(a_v^c)^2 - \lambda_v^c\mu_v^c}\right], \quad (20)$$

from which we observe that $\gamma_f(\lambda_v^c)$ is sublinear, i.e., $\gamma_f(\lambda_v^c) = o(\lambda_v^c)$ which is true if $(a_v^c)^2 > \lambda_v^c\mu_v^c$, and is linear, i.e., $\gamma_f(\lambda_v^c) = O(\lambda_v^c)$ which is true if $(a_v^c)^2 \approx \lambda_v^c\mu_v^c$. This is also intuitive from the surjection factors. From (3), (7), and (9), we get the desired result. Furthermore, if $\gamma_f(\lambda_v^c) \approx a_v^c$, the range provided in (20) is tight and $L_v^c \approx \sqrt{\frac{\mu_v^c}{\lambda_v^c}}d_{f_c}(m_v^c) + \frac{1}{\sqrt{\mu_v^c/\lambda_v^c - 1}} \ge \sqrt{\frac{\lambda_v^c}{\mu_v^c}} + \frac{1}{\sqrt{\mu_v^c/\lambda_v^c - 1}}$. Hence, the best achievable scaling is $L_v^c = O(\sqrt{d_{f_c}(m_v^c)})$. ∎

Remark 1: From Prop. IV-B, we observe that as the time complexity $d_{f_c}(m_v^c)$ increases, a_v^c decreases and the processing factor $\gamma_f(\lambda_v^c)$ concentrates. Ignoring this principle, if $\gamma_f(\lambda_v^c)$ were increased with $d_{f_c}(m_v^c)$, then the costs $W_{v,comp}^c(m_v^c)$ and $W_{v,comm}^c(n_v^c)$ would both increase. However, $\gamma_f(\lambda_v^c)$ can decrease with $d_{f_c}(m_v^c)$, and the output rate may not be compressed below $\boldsymbol{H}(f_c(\mathbf{X}))$. Hence, $\boldsymbol{H}(f_c(\mathbf{X})) \le \gamma_f(\lambda_v^c) \le \boldsymbol{H}(\mathbf{X})$ is necessary, where the upper limit is due to the identity function.

We denote by M_v^c the long-term average number of class $c \in C$ packets in $v \in V'$ waiting for communications service in case of no computation, i.e., $M_v^c = \frac{\lambda_v^c}{\mu_v^c - \lambda_v^c}$. Intermediate computations reduce the long term average number of packets L_v^c in the system which we characterize next.

Proposition 1: The long-term average number of packets L_v^c satisfies

$$\frac{\boldsymbol{H}(f_c(\mathbf{X}))}{\mu_v^c - \boldsymbol{H}(f_c(\mathbf{X}))} \le L_v^c \le M_v^c, \quad v \in V, \quad c \in C. \quad (21)$$

Proof: In the case of no computation, $m_v^c = 0$ and $n_v^c = L_v^c$, and the long-term average number of packets in v satisfies $L_v^c = M_v^c$. This gives an upper bound to L_v^c.

When node v computes it is true that $m_v^c > 0$, and from (2) and employing $\frac{\lambda_v^c}{\gamma_f(\lambda_v^c)}$ we have $L_v^c = \frac{\lambda_v^c}{\gamma_f(\lambda_v^c)}n_v^c \ge n_v^c = \frac{\gamma_f(\lambda_v^c)}{\mu_v^c - \gamma_f(\lambda_v^c)}$. We next provide an upper bound to the above equality where using $\Gamma_c \approx \frac{\gamma_f(\lambda_v^c)}{\lambda_v^c}$ as a proxy for (5) the bound can be approximated as function of Γ_c as follows:

$$L_v^c \le \frac{\lambda_v^c}{\gamma_f(\lambda_v^c)} \cdot \frac{\gamma_f(\lambda_v^c)}{\lambda_v^c - \gamma_f(\lambda_v^c)} \approx \frac{1}{1-\Gamma_c} = \frac{1}{1 - \frac{\boldsymbol{H}(f_c(\mathbf{X}))}{H(\mathbf{X})}},$$

where $n_v^c \to 0$ as $\Gamma_c \to 0$, i.e., for deterministic functions where the gain of computing is the highest. On the other hand, as $\Gamma_c \to 1$, we obtain the no computation limit, i.e., $n_v^c \to M_v^c$.

Using Prop. IV-B and that $d_{f_c}(m_v^c)$ is monotonically increasing in m_v^c, the lower bound follows from Little's law in (3) and the minimal rate required for recovering $f_c(\mathbf{X})$ which gives (a):

$$L_v^c \ge \gamma_f(\lambda_v^c)\frac{2}{\mu_v^c - \gamma_f(\lambda_v^c)} \overset{(a)}{\ge} \frac{\boldsymbol{H}(f_c(\mathbf{X}))}{\mu_v^c - \boldsymbol{H}(f_c(\mathbf{X}))},$$

Manipulating the lower bound, we get $\gamma_f(\lambda_v^c) \ge \frac{\mu_v^c \boldsymbol{H}(f_c(\mathbf{X}))}{2\mu_v^c - \boldsymbol{H}(f_c(\mathbf{X}))}$ and the desired bound. ∎

Prop. 1 yields a better inner bound than that of Slepian and Wolf [2] as the lower bound in (21) is no more than $\boldsymbol{H}(f_c(\mathbf{X}))$ which we expect due to intermediate compression for computing.

C. Processing of a Single Class Flow

We study the cost of a single class flow in isolation and we drop the superscript c. More specifically, we consider two examples that account for task complexity and its distribution.

Example 1 (Bisection – Tree Branching Over the Network): The network computes $f(\mathbf{X}) = \min \mathbf{X}$ using the bisection method. The search function has time complexity $\log(N)$ for an input size N. The arrivals $\mathbf{X} = (X_1, \ldots, X_N)$ are uniformly split among nodes, i.e., $\beta_v = \beta$, $\forall v \in V'$. Node v works on the set $\mathbf{X}_v = (X_{(v-1)\frac{N}{|V'|}+1}, \ldots, X_{v\frac{N}{|V'|}})$ and computes $f(\mathbf{X}_v)$ locally, i.e., $p_{s,v}^{rou} = \frac{|V'|}{N}$.

(i) No intermediate routing is allowed, i.e., $p_{v,0}^{rou} = 1$. The computation outcome is directly routed to $0 \in V$ that decides the final outcome $\min_{v \in V'} f(\mathbf{X}_v) = f(\mathbf{X})$. The computation time complexity of the initial stage per node $v \in V'$ is $O(\log(\frac{N}{|V'|}))$ and the routing stage is $\log(|V'|)$. Hence, the total computation complexity is $|V'|O(\log(\frac{N}{|V'|})) + \log(|V'|)$. The communications cost is due to the routings (s, v), $(v, 0)$ for $v \in V'$, which in total yields $|V'|\exp(\frac{N}{|V'|}) + |V'|\exp(1)$.

(ii) Intermediate routings are allowed. We pick a subset of nodes $W \subset V'$, $|W| = O(\log(|V'|))$ to run the bisection algorithm, i.e., $p_{v,w}^{rou} \approx \frac{1}{\log(|V'|)}$, $w \in W$ and $p_{v,w}^{rou} = 0$, $w \in V' \backslash W$. Hence, the complexity of the initial stage per $v \in V' \backslash W$ is $O(\log(\frac{N}{|V'|}))$. Node $w \in W$ then computes $f(\mathbf{X}_w) = \min_{v; p_{v,w}^{rou} > 0} f(\mathbf{X}_v)$, and routes intermediate computation to $0 \in V$ which in turn computes $\min_{w \in W} f(\mathbf{X}_w) = f(\mathbf{X})$ which has complexity $O(\log(|W|))$. The total compute complexity is

$$(|V'| - |W|)O\left(\log\left(\frac{N}{|V'|}\right)\right) + |W|\log\left(\frac{|V'| - |W|}{|W|}\right) + \log(|W|).$$

The communications cost is due to the routings (s, v), (v, w), and $(w, 0)$ for $v \in V'$, $w \in W$, which in aggregation gives $|V'|\exp(\frac{N}{|V'|}) + (|V'| - |W|)\exp(1) + |W|\exp(1) = |V'|\exp(1)$.

From (i)-(ii) the total cost is determined by the computation complexity, which can be reduced via intermediate routings. The highest gains are possible when the depth of the network grows like $O(\log(N))$ that favors divide-and-conquer. A similar approach will follow for MapReduce.

Example 2 (Classification): Classification is the problem of identifying to which of a set of categories an observation $\mathbf{X} = (X_1, \ldots, X_N)$ belongs. The classification function has time complexity $\exp(N)$ for an input size N. In linear classification, the predicted category is the one with the highest score, where the score function has the following dot product-form $f_l(\mathbf{X}) = score(\mathbf{X}, l) = \sum_{k=1}^{N} \beta_{kl} \cdot X_k$ where $\boldsymbol{\beta}_l = (\beta_{1l}, \ldots, \beta_{Nl})$ is the vector of weights corresponding to category l, and $f_l(\mathbf{X})$

represents the utility or score associated with assigning $\mathbf{X}$ to category l.

(i) If data is not split, the compute cost is the cost of N multiplications, i.e., $O(N\log(N))$.

(ii) If data is split between a subset $W \subseteq V'$ such that $\mathbf{X}_w = (X_{(w-1)\frac{N}{|W|}+1}, \ldots, X_{w\frac{N}{|W|}})$, then $\mathbf{X}$ and $\boldsymbol{\beta}_l$ should be sent for accurately computing the score. This costs $O(\frac{N}{|W|}\log N)$ where $\log N$ bits quantify the locations of $\mathbf{X}$ for each w. The compute cost is $O(\frac{N}{|W|}\log N \log(\frac{N}{|W|}\log N))$. Coordinating the intermediate computations requires $|W|$ additions, yielding a total cost of

$$O\left(\frac{N}{|W|}\log N \log\left(\frac{N}{|W|}\log N\right)\right) + O(|W|).$$

To ensure that the computation cost in (ii) is smaller than that of (i), $|W| \geq O(\log N)$. Otherwise, it will be higher than $O(N\log N)$. The communication cost of forwarding the intermediate results of $w \in W$ is determined by the flow limit $\boldsymbol{H}(f_l(\mathbf{X}_w))$ at $w \in W$. If $|W|$ is small, score estimates are easy to obtain due to the weak law of large numbers (a large ratio $\frac{N}{|W|}$ gives a good estimate of scores $f_l(\mathbf{X}_w)$ as the sample average converges in probability to the expected value, yielding a low entropy), but if $|W|$ is large, this might not be possible, causing a high $\boldsymbol{H}(f_l(\mathbf{X}_w))$ (e.g., if the classifier is sensitive to $\boldsymbol{\beta}_l$ or $\boldsymbol{\beta}_l$ need to be trained, incurring a high cost for task distribution) and hence a high communication cost per $w \in W$. As the communication costs accumulate, distributing such flows might not be favorable. However, if $\boldsymbol{\beta}_l$ are not sensitive to the coordinates of $\mathbf{X}$, i.e., the observations are easy to classify, then the communications overhead can be lowered. Hence, if splitting does not cause jumps in $\boldsymbol{H}(f_l(\mathbf{X}_w))$, it might be favorable.

From Examples 1 and 2, the task distribution depends on the task complexity and the associated communication resources to leverage the distributed computation. Tasks with high complexity, as long as the compute resources are sufficient, require centralized processing. If one node is not that powerful, then the load needs to be distributed to prevent very high $\gamma_f(\lambda_v^c)$. It is intuitive that low complexity tasks, such as Search and MapReduce, as in Example 1, can be distributed over the network by splitting dataset. However, tasks with high complexity, such as Classification, might not be distributed because one needs the whole dataset, as we detailed in Example 2.

Both in the communication or computation intensive regimes, condition (6) ensures effective computing. To find out the role of $\gamma_f(\lambda_v^c)$ in computing, we next focus on multiple flows.

D. Processing of Multiple Flows and Class Conversions

We explore how to effectively allocate compute tasks without requiring the protocol information. To that end exploiting $\gamma_f(\lambda_v)$ we next study the complexity of 3 function classes.

Sublinear surjection factor: Let $\gamma_f(\lambda_v) = o(\lambda_v^c)$ which is sublinear. From (2) it is true that

$$m_v^c = L_v^c \cdot \left(1 - \frac{\gamma_f(\lambda_v^c)}{\lambda_v^c}\right) = O(L_v^c),$$

$$n_v^c = L_v^c \cdot \frac{\gamma_f(\lambda_v^c)}{\lambda_v^c} = o(L_v^c) = o(m_v^c).$$

From (9) $W_{v,comm}^c(n_v^c) = \frac{1}{\mu_v^c - \gamma_f(\lambda_v^c)} = O(\exp(\gamma_f(\lambda_v^c))) = O(\lambda_v^c)$ and $n_v^c = \frac{\gamma_f(\lambda_v^c)}{\mu_v^c - \gamma_f(\lambda_v^c)} = O(\lambda_v^c)$. Furthermore, from (7) and (8) $W_{v,comp}^c(m_v^c)$ corresponding to different function classes are

$$W_{v,comp}^c(m_v^c) \in \left\{\frac{1}{\lambda_v^c}O(n_v^c),\ \frac{1}{\lambda_v^c}O(\exp(n_v^c)),\ \frac{1}{\lambda_v^c}O(\exp(\exp(n_v^c)))\right\},$$

which is an ordered set for Search, MapReduce and Classification.

Linear surjection factor: Let $\gamma_f(\lambda_v) = O(\lambda_v^c)$. From (2) it is true that

$$L_v = O(m_v^c) = O(n_v^c), \quad 0 < \gamma_f(\lambda_v) < \lambda_v.$$

If $\gamma_f(\lambda_v^c) = \lambda_v^c > 0$, then $L_v^c = n_v^c > m_v^c = 0$, and if $\gamma_f(\lambda_v^c) = 0$, then $L_v^c = m_v^c > n_v^c = 0$. From (9) $n_v^c = O(\exp(\lambda_v^c))$. From (7) and (8) the compute cost respectively for Search, MapReduce and Classification satisfies the corresponding entries below:

$$W_{v,comp}^c(m_v^c) \in \left\{\frac{1}{\lambda_v^c}O(\log(n_v^c)),\ \frac{1}{\lambda_v^c}O(n_v^c),\ \frac{1}{\lambda_v^c}O(\exp(n_v^c))\right\}.$$

For sublinear $\gamma_f(\lambda_v^c)$, the computation cost is either very high because $(\lambda_v^c)^*$ is high (due to $W_{v,comp}^c(m_v^c) \geq W_{v,comm}^c(n_v^c)$) or relatively higher than the communication cost. In these regimes, low complexity tasks, such as Search and MapReduce should be done distributedly (because the dynamic range of λ_v^c is smaller), and high complexity tasks can be run centralized provided that compute resources suffice. On the other hand, for linear $\gamma_f(\lambda_v^c)$, the communication cost dominates the computation cost as $(\lambda_v^c)^*$ is low, leading to a more distributed setting.

We contrast the costs for sublinear versus linear $\gamma_f(\lambda_v^c)$. Task distribution suits for noncomplex tasks Search and MapReduce with low $\gamma_f(\lambda_v^c)$ since the maximum λ_v^c that v supports, $(\lambda_v^c)^*$, grows with the complexity, allowing centralized processing of complex tasks, e.g., Classification, as shown in Fig. 3. If $W_{v,comm}^c$ dominates the cost (high $\gamma_f(\lambda_v^c)$), a divide-and-conquer-based approach is favorable as $(\lambda_v^c)^*$ is low. However, when $W_{v,comm}^c$ is rather negligible, the dynamic range of λ_v^c grows and centralized processing may be feasible. Hence, for low (high) complexity tasks the network may operate in a connected (isolated) fashion. If the task complexity is heterogeneous, task-based link reservations become favorable, and the network may be sparsely connected as $W_{v,comm}^c$ starts to dominate. If the routings are symmetric, then the tasks are distributed, in which case realizing communication intensive tasks may incur high $W_{v,comm}^c$.

We provide insights into flow deviation among tasks in Fig. 3, where the application of successive flow deviations leads to local minima as in the gradient method [55]. For example, for given processing capability μ_v at $v \in V$ flows of high complexity tasks requiring centralized processing (with

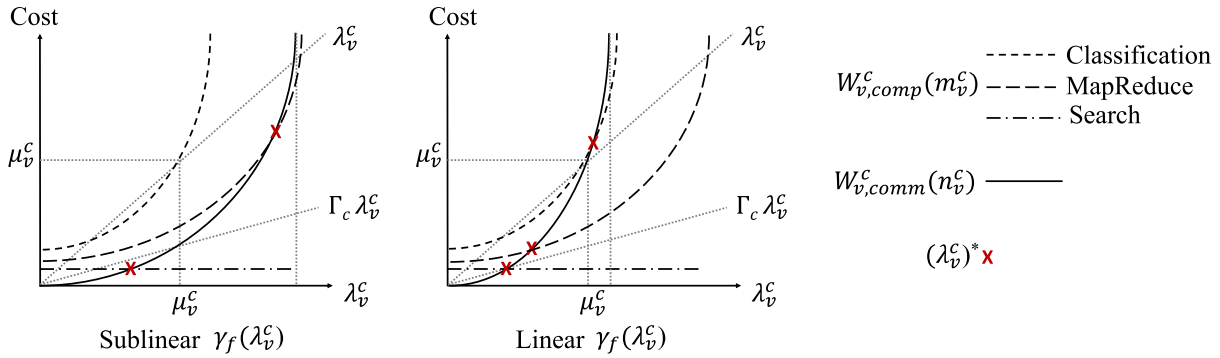

Fig. 3. Cost versus λ_v^c where different line types represent distinct functions and the solid convex curve corresponds to $W_{v,comm}^c$, as shown in the legend. (Left) Sublinear surjection where $\lambda_v^c \in [0, (\lambda_v^c)^*]$ has a high dynamic range. (Right) Linear surjection where $W_{v,comm}^c$ is steep, leading to distributed computing due to smaller range of λ_v^c.

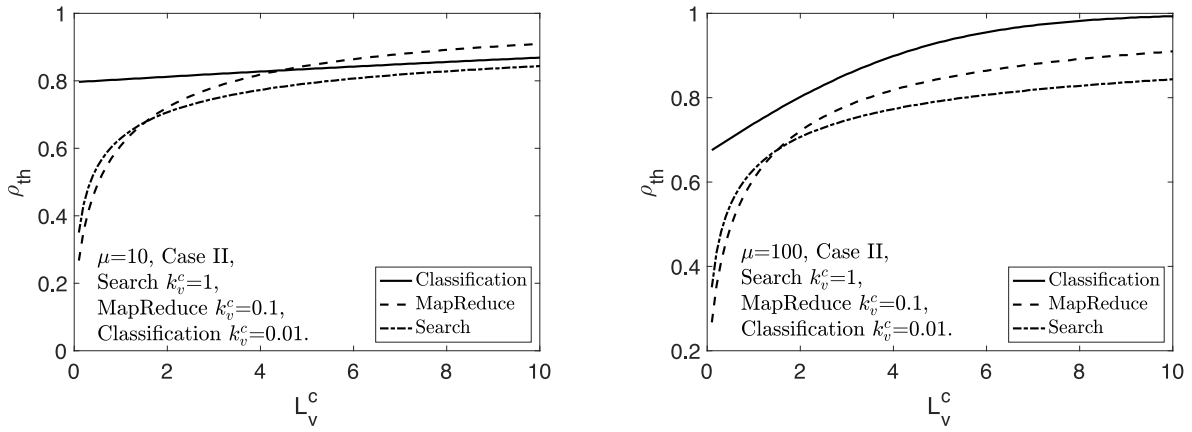

Fig. 4. The critical threshold ρ_{th} for computation versus L_v^c that satisfies Little's law for computation in (3).

high $(\lambda_v^c)^*$) versus low complexity tasks running in a distributed manner (with small $(\lambda_v^c)^*$) can be traded-off. Sublinear $\gamma_f(\lambda_v^c)$ is tailored for a hybrid regime, i.e., a regime which allows a mixture of distributed and centralized operation of tasks to be conducted via flow deviation among tasks, and linear $\gamma_f(\lambda_v^c)$ enforces distributed computing of the tasks.

Next in Section V, we will numerically evaluate the load threshold ρ_{th} in (19) for function classes with time complexities given in (8), and the local solution of MinCost given in (18).

V. Numerical Evaluation of Performance

Our objective in this section is to explore how to distribute computation in the light of the fundamental cost limits for the networked computation model detailed in Sections II-IV.

Leveraging Little's law in (3) for computing, we numerically evaluate the threshold on traffic intensity ρ_{th} in (19) of Prop. IV-B for the set of functions with time complexities $d_{f_c}(m_v^c)$ in (8) and compute cost models $W_{v,comp}^c(m_v^c)$ in (9). In Fig. 4 we contrast ρ_{th} as increasing functions of L_v^c for given processing power μ_v which is selected from the set $\{10^{-4}, 10^{-3}, 10^{1}, 10^{2}\}$ and the same for all c and all v. Our experiment shows that ρ_{th} is high for Classification, implying that a node can only compute if the flow is sufficient, and ρ_{th} grows slower for low complexity Search and MapReduce functions, i.e., nodes compute even for small flow rates. This is because if $\rho_v^c(out) < \rho_{th}$, for high complexity functions $d_{f_c}(m_v^c)$ grows much faster than $W_{v,comm}^c(m_v^c)$, rendering L_v^c sufficiently small to ensure a valid ρ_{th} for computing. The benefit of computing is eminent for MapReduce and even higher for Search. This implies that most of the processing power should be reserved for running simpler tasks to enable cost effective distributed computing.

We next consider a simple network topology with $|V'| = 3$ nodes, excluding the virtual source s and the virtual destination 0, with nodes having identical processing capabilities. We distribute the different function classes, i.e., Search, MapReduce, and Classification, with different $d_{f_c}(m_v^c)$ over V where we allow mixing of multiple flows. We assume that $P^{rou}(c)$ has entries chosen uniformly at random. We assume that the input rate is $H(X) = \lambda_v^c$, and from (6) which states that $\frac{H(f_c(\mathbf{X}))}{H(\mathbf{X})}\lambda_v^c \leq \gamma_f(\lambda_v^c) \leq \lambda_v^c < \mu_v^c$, the entropy rates for the Search, MapReduce, and Classification functions are $\boldsymbol{H}(f_c(X)) \approx k_v^c \log(\lambda_v^c)$, $\boldsymbol{H}(f_c(X)) \approx k_v^c \lambda_v^c$, and $\boldsymbol{H}(f_c(X)) \approx k_v^c \exp(\lambda_v^c)$, respectively, where $k_v^c > 0$ is a constant proxy for modeling the cost. We numerically evaluate this setting to investigate the behaviors of the surjection factor $\gamma_f(\lambda_v^c)$, $d_{f_c}(m_v^c)$, and MinCost versus Γ_c, and determine sufficient conditions on $\gamma_f(\lambda_v^c)$ and μ_v^c to support a higher L_v^c via mixing flows.

(i) Separating flows. Each node works on a distinct function class independently. Hence, the compute cost is separable

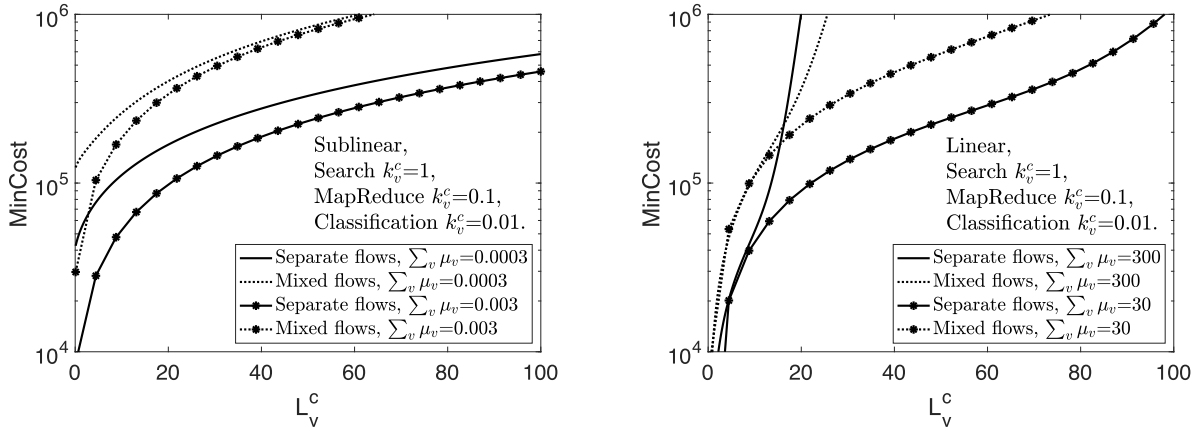

Fig. 5. (Left) Sublinear surjection factor MinCost vs L_v^c. (Right) Linear surjection factor MinCost vs L_v^c.

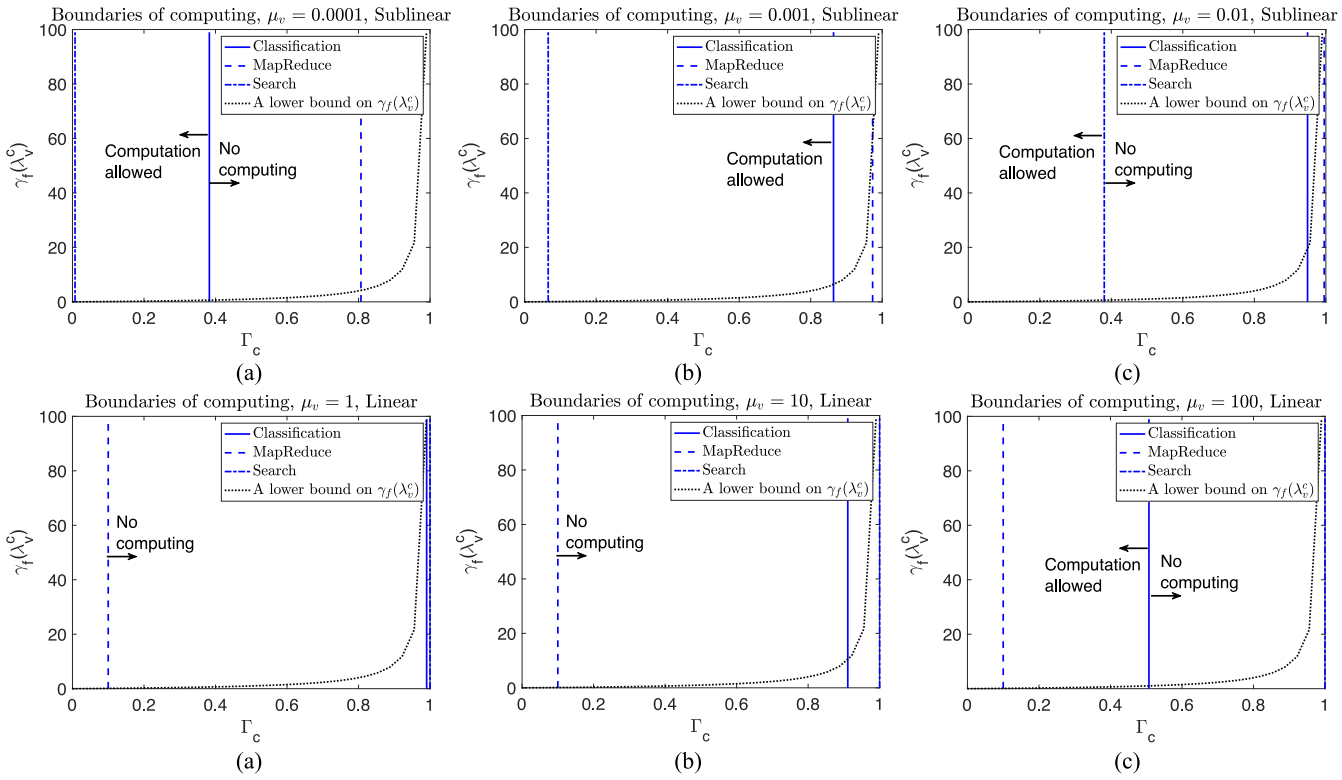

Fig. 6. (Top) A lower bound on the sublinear surjection factor $\gamma_f(\lambda_v^c)$ for (a) $\mu_v^c = 0.0001$, (b) $\mu_v^c = 0.001$, (c) $\mu_v^c = 0.01$. The boundaries of computation for each function class is indicated by vertical lines where computing of a function class is only allowed for Γ_c below the line corresponding to the function. (Bottom) A lower bound on the linear surjection factor $\gamma_f(\lambda_v^c)$ for (a) $\mu_v^c = 1$, (b) $\mu_v^c = 10$, (c) $\mu_v^c = 100$.

and independent from the routings. The communications cost is only between links $(v, 0)$, $v \in V'$ and can be computed using (9) where $\gamma_f(\lambda_v^c) = \boldsymbol{H}(f_c(\mathbf{X}))$. Using (2), m_v^c and n_v^c can be determined as function of L_v^c and W_v^c is nonzero at each node only for the function class it works on and it satisfies (1). Hence, total cost is easily determined.

(ii) Mixing flows. Each class is routed based on $p_s^{rou}(c) = [\frac{1}{3}, \frac{1}{3}, \frac{1}{3}]$, computation is evenly split among V', where $p_{v,0}^{rou}(c) = \frac{\boldsymbol{H}(f_c(\mathbf{X}))}{H(\mathbf{X})}$ and $p_{v,w}^{rou}(c) = \frac{1-p_{v,0}^{rou}(c)}{3}$ for $w, v \in V'$.

We compare MinCost of (i) separating and (ii) mixing flows for sublinear and linear $\gamma_f(\lambda_v^c)$, in Fig. 5, with varying μ_v^c. From Fig. 5 (Left), as μ_v^c increases MinCost decays for sublinear $\gamma_f(\lambda_v^c)$. The performances of (i)-(ii) are similar when μ_v^c is very low and mixing flows performs worse. When $\gamma_f(\lambda_v^c)$ sublinearly scales, as the processing power μ_v^c increases λ_v^c also increases, and the function $W_{v,comm}^c(n_v^c)$ (which is convex and increasing in $\gamma_f(\lambda_v^c)$ and decreasing in μ_v^c) decreases. Furthermore, as μ_v^c increases, we expect L_v^c to decay, and $d_{f_c}(m_v^c)$ and $W_{v,comp}^c(m_v^c)$ to decay (logarithmic in μ_v^c) as well. Hence, W_v^c decays. It is also intuitive due to Little's law which states $L_v^c = \gamma_f(\lambda_v^c) \cdot W_v^c$ that when L_v^c decays, W_v^c also decays, when v is considered in isolation. From Fig. 5 (Right) as μ_v^c increases MinCost increases for linear $\gamma_f(\lambda_v^c)$. Because $\gamma_f(\lambda_v^c)$ linearly scales, $W_{v,comm}^c$ is fixed. We note that

n_v^c and m_v^c increase in μ_v^c and $W_{v,comp}^c(m_v^c) \geq W_{v,comm}^c(n_v^c)$, and hence $W_{v,comp}^c(m_v^c)$ increases in μ_v^c. Therefore, if the flows are mixed among V nodes, while the processed flow intensity supported per node is higher in the heterogeneous computing scenario via mixing rather than separating flows, the processing resources will be used more effectively instead of being partially utilized for separately assigned tasks. The sublinear $\gamma_f(\lambda_v^c)$ will help reduce $W_{v,comm}^c(n_v^c)$. This will ensure at high μ_v^c that node v can serve a higher L_v^c without having to sacrifice MinCost. The savings of (ii) in MinCost are more apparent at high L_v^c.

Fig. 6 (Top) and (Bottom) show the behavior of a lower bound on $\gamma_f(\lambda_v^c)$ in (15), for sublinear and linear $\gamma_f(\lambda_v^c)$, respectively, as function of surjectivity Γ_c, and indicate that the boundary is shifted towards right as the node's processing capability μ_v^c increases, allowing higher Γ_c. If the communication delay is negligible, intermediate computing can improve MinCost.

VI. Conclusion

We provided a novel perspective to function computation in networks. We introduced the notion of entropic surjectivity to assess the functional complexity. Extending Little's law to computing, we derived regimes for which intermediate computations can provide significant savings. Our approach can be considered as an initial step towards understanding how to distribute computation and balance functional load in networks. Future directions include devising coding techniques for in network functional compression, by using compressed sensing and the compression theorem of Slepian and Wolf, employing the concepts of graph entropy, and exploiting function surjectivity. They also include more general network models beyond stationary and product-form.

References

[1] D. Malak, A. Cohen, and M. Médard, "How to distribute computation in networks," in *Proc. IEEE Infocom*, Los Angeles, CA, USA, Jul. 2020, pp. 327–336.

[2] D. Slepian and J. K. Wolf, "Noiseless coding of correlated information sources," *IEEE Trans. Inf. Theory*, vol. IT-19, no. 4, pp. 471–480, Jul. 1973.

[3] E. Candes, J. Romberg, and T. Tao, "Stable signal recovery from incomplete and inaccurate measurements," *Commun. Pure Appl. Math. J. Courant Inst. Math. Sci.*, vol. 59, no. 8, pp. 1207–1223, Aug. 2006.

[4] E. Candes and J. Romberg, "Sparsity and incoherence in compressive sampling," *Inverse Probl.*, vol. 23, no. 3, p. 969, Apr. 2007.

[5] D. Donoho, "Compressed sensing," *IEEE Trans. Inf. Theory*, vol. 52, no. 4, pp. 1289–1306, Apr. 2006.

[6] S. Feizi, M. Médard, and M. Effros, "Compressive sensing over networks," in *Proc. IEEE Allerton*, Monticello, IL, USA, Sep. 2010, pp. 1558–1562.

[7] S. Feizi and M. Médard, "A power efficient sensing/communication scheme: Joint source-channel-network coding by using compressive sensing," in *Proc. IEEE Allerton Conf.*, Monticello, IL, USA, Sep. 2011, pp. 1048–1054.

[8] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, "Coded distributed computing: Straggling servers and multistage dataflows," in *Proc. IEEE Allerton Conf.*, Monticello, IL, USA, Sep. 2016, pp. 164–171.

[9] S. S. Pradhan and K. Ramchandran, "Distributed source coding using syndromes (DISCUS): Design and construction," *IEEE Trans. Inf. Theory*, vol. 49, no. 3, pp. 626–643, Mar. 2003.

[10] T. P. Coleman, A. H. Lee, M. Médard, and M. Effros, "Low-complexity approaches to Slepian–Wolf near-lossless distributed data compression," *IEEE Trans. Inf. Theory*, vol. 52, no. 8, pp. 3546–3561, Aug. 2006.

[11] A. Wyner and J. Ziv, "The rate-distortion function for source coding with side information at the decoder," *IEEE Trans. Inf. Theory*, vol. IT-22, no. 1, pp. 1–10, Jan. 1976.

[12] H. Yamamoto, "Wyner–Ziv theory for a general function of the correlated sources," *IEEE Trans. Inf. Theory*, vol. IT-28, no. 5, pp. 803–807, Sep. 1982.

[13] T. Ho, M. Médard, R. Koetter, D. Karger, M. Effros, J. Shi, and B. Leong, "A random linear network coding approach to multicast," *IEEE Trans. Inf. Theory*, vol. 52, no. 10, pp. 4413–4430, Oct. 2006.

[14] J. Körner, "Coding of an information source having ambiguous alphabet and the entropy of graphs," in *Proc. Prague Conf. Inf. Theory*, Prague, Czech Republic, Sep. 1973, pp. 411–425.

[15] N. Alon and A. Orlitsky, "Source coding and graph entropies," *IEEE Trans. Inf. Theory*, vol. 42, no. 5, pp. 1329–1339, Sep. 1996.

[16] A. Orlitsky and J. R. Roche, "Coding for computing," *IEEE Trans. Inf. Theory*, vol. 47, no. 3, pp. 903–917, Mar. 2001.

[17] H. Feng, M. Effros, and S. Savari, "Functional source coding for networks with receiver side information," in *Proc. IEEE Allerton Conf.*, Monticello, IL, USA, Sep. 2004, pp. 1419–1427.

[18] S. Feizi and M. Médard, "On network functional compression," *IEEE Trans. Inf. Theory*, vol. 60, no. 9, pp. 5387–5401, Sep. 2014.

[19] P. Delgosha and V. Anantharam, "Distributed compression of graphical data," in *Proc. IEEE Int. Symp. Inf. Theory*, Vail, CO, USA, Jun. 2018, pp. 2216–2220.

[20] R. Gallager, "Finding parity in a simple broadcast network," *IEEE Trans. Inf. Theory*, vol. 34, no. 2, pp. 176–180, Mar. 1988.

[21] S. Kamath and D. Manjunath, "On distributed function computation in structure-free random networks," in *Proc. IEEE Int. Symp. Inf. Theory*, Toronto, ON, Canada, Jul. 2008, pp. 647–651.

[22] V. Shah, B. Dey, and D. Manjunath, "Network flows for function computation," *IEEE J. Sel. Areas Commun.*, vol. 31, no. 4, pp. 714–730, Apr. 2013.

[23] S. Feizi, A. Zhang, and M. Médard, "A network flow approach in cloud computing," in *Proc. IEEE Conf. Inf. Sci. Syst.*, Baltimore, MD, USA, Mar. 2013, pp. 1873–1877.

[24] C. Huang, Z. Tan, S. Yang, and X. Guang, "Comments on cut-set bounds on network function computation," *IEEE Trans. Inf. Theory*, vol. 64, no. 9, pp. 6454–6459, Sep. 2018.

[25] A. Giridhar and P. Kumar, "Computing and communicating functions over sensor networks," *IEEE J. Sel. Areas Commun.*, vol. 23, no. 4, pp. 755–764, Apr. 2005.

[26] H. Kowshik and P. R. Kumar, "Optimal computation of symmetric Boolean functions in Tree networks," in *Proc. IEEE Int. Symp. Inf. Theory*, Austin, TX, USA, Jun. 2010, pp. 1873–1877.

[27] R. Ahlswede, N. Cai, S. Y. Li, and R. W. Yeung, "Network information flow," *IEEE Trans. Inf. Theory*, vol. 46, no. 4, pp. 1204–1216, Jul. 2000.

[28] R. Koetter and M. Médard, "An algebraic approach to network coding," *IEEE/ACM Trans. Netw.*, vol. 11, no. 5, pp. 782–795, Oct. 2003.

[29] R. Koetter, M. Effros, T. Ho, and M. Médard, "Network codes as codes on graphs," in *Proc. IEEE Conf. Inf. Sci. Syst.*, Princeton, NJ, USA, Mar. 2004, pp. 1–19.

[30] R. Appuswamy and M. Franceschetti, "Computing linear functions by linear coding over networks," *IEEE Trans. Inf. Theory*, vol. 60, no. 1, pp. 422–431, Jan. 2014.

[31] M. Sefidgaran and A. Tchamkerten, "Computing a function of correlated sources: A rate region," in *Proc. Int. Symp. Inf. Theory*, St. Petersburg, Russia, Jul./Aug. 2011, pp. 1856–1860.

[32] A. Tripathy and A. Ramamoorthy, "Zero-error function computation on a directed acyclic network," in *Proc. IEEE Inf. Theory Wkshp*, Guangzhou, China, Nov. 2018, pp. 1–5.

[33] Y. Yang, P. Grover, and S. Kar, "Rate distortion for lossy in-network linear function computation and consensus: Distortion accumulation and sequential reverse water-filling," *IEEE Trans. Inf. Theory*, vol. 63, no. 8, pp. 5179–5206, Aug. 2017.

[34] R. Appuswamy, M. Franceschetti, N. Karamchandani, and K. Zeger, "Network coding for computing: Cut-set bounds," *IEEE Trans. Inf. Theory*, vol. 57, no. 2, pp. 1015–1030, Feb. 2011.

[35] S. Gitzenis, G. S. Paschos, and L. Tassiulas, "Asymptotic laws for joint content replication and delivery in wireless networks," *IEEE Trans. Inf. Theory*, vol. 59, no. 5, pp. 2760–2776, May 2013.

[36] J. D. Ullman, "Designing good MapReduce algorithms," *Crossroads ACM Mag. Students*, vol. 19, no. 1, pp. 30–34, Sep. 2012.
[37] S. Li, M. A. Maddah-Ali, Q. Yu, and A. S. Avestimehr, "A fundamental tradeoff between computation and communication in distributed computing," *IEEE Trans. Inf. Theory*, vol. 64, no. 1, pp. 109–128, Jan. 2018.
[38] Y. H. Ezzeldin, M. Karmoose, and C. Fragouli, "Communication vs distributed computation: An alternative trade-off curve," in *Proc. IEEE Inf. Theory Wkshp*, Kaohsiung, Taiwan, Nov. 2017, pp. 279–283.
[39] M. Kiamari, C. Wang, and A. S. Avestimehr, "On heterogeneous coded distributed computing," in *Proc. IEEE Globecom*, Dec. 2017, pp. 1–7.
[40] S. Banerjee, P. Gupta, and S. Shakkottai, "Towards a queueing-based framework for in-network function computation," *Queueing Syst.*, vol. 72, no. 3, pp. 219–250, Dec. 2012.
[41] I. A. Gillani, P. Vyavahare, and A. Bagchi, "Random walk based in-network computation of arbitrary functions," Feb. 2017. [Online]. Available: arXiv:1702.03741.
[42] I. A. Gillani, P. Vyavahare, and A. Bagchi, "Lower bounds for in-network computation of arbitrary functions," *Distrib. Comput.*, vol. 34, pp. 1–13, Apr. 2021.
[43] W. Wei *et al.*, "Impact of in-network aggregation on target tracking quality under network delays," *IEEE J. Sel. Areas Commun.*, vol. 31, no. 4, pp. 808–818, Apr. 2013.
[44] A. Anand and N. B. Mehta, "Quick, decentralized, energy-efficient one-shot max function computation using timer-based selection," *IEEE Trans. Commun.*, vol. 63, no. 3, pp. 927–937, Mar. 2015.
[45] Y. Yang, "Coded computing systems decoded: Dealing with unreliability and elasticity in modern computing," Ph.D. dissertation, Dept. Comput. Sci., Carnegie Mellon Univ., Pittsburgh, PA, USA, 2019.
[46] Y. Yang, S. Kar, and P. Grover, "Graph codes for distributed instant message collection in an arbitrary noisy broadcast network," *IEEE Trans. Inf. Theory*, vol. 63, no. 9, pp. 6059–6084, Sep. 2017.
[47] T. Cover, "A proof of the data compression theorem of Slepian and Wolf for ergodic sources (corresp.)," *IEEE Trans. Inf. Theory*, vol. IT-21, no. 2, pp. 226–228, Mar. 1975.
[48] L. Kleinrock, *Queuing Systems Vol. I: Theory*. New York, NY, USA: Wiley, 1975.
[49] F. P. Kelly, *Reversibility and Stochastic Networks*. Cambridge, U.K.: Cambridge Univ. Press, 2011.
[50] R. Nelson, *Probability, Stochastic Processes, and Queueing Theory: The Mathematics of Computer Performance Modeling*. New York, NY, USA: Springer, 2013.
[51] R. Srikant and L. Ying, *Communication Networks*. Cambridge, U.K.: Cambridge Univ. Press, 2014.
[52] A. E. Gamal and Y. H. Kim, *Network Information Theory*. Cambridge, U.K.: Cambridge Univ. Press, Dec. 2011.
[53] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, U.K.: Cambridge Univ. Press, 2009.
[54] D. P. Bertsekas, *Nonlinear Programming*. Belmont, MA, USA: Athena Sci., 1999.
[55] L. Fratta, M. Gerla, and L. Kleinrock, "The flow deviation method: An approach to store-and-forward communication network design," *Networks*, vol. 3, no. 2, pp. 97–133, Jan. 1973.

Derya Malak (Member, IEEE) received the B.S. degree in electrical and electronics engineering (EEE) with minor in physics from Middle East Technical University, Ankara, Turkey, in 2010, the M.S. degree in EEE from Koc University, Istanbul, Turkey, in 2013, and the Ph.D. degree in electrical and computer engineering from the University of Texas at Austin in 2017, where she was affiliated with the Wireless Networking and Communications Group. She is an Assistant Professor with the Communication Systems Department, Eurecom. She was an Assistant Professor with the Department of Electrical, Computer, and Systems Engineering, Rensselaer Polytechnic Institute from 2019 to 2021, and a Postdoctoral Associate with MIT from 2017 to 2019. He has held visiting positions with INRIA and LINCS, Paris, France, and Northeastern University, Boston, MA, USA. She has held summer internships with Huawei Technologies, Plano, TX, USA, and Bell Laboratories, Murray Hill, NJ, USA. She was awarded the Graduate School Fellowship by UT Austin from 2013 to 2017. She was selected to participate in the Rising Stars Workshop for women with EECS, MIT, Cambridge, MA, USA, in 2018, and the 7th Heidelberg Laureate Forum, Heidelberg, Germany, in 2019.

Muriel Médard (Fellow, IEEE) is the Cecil H. Green Professor with the Electrical Engineering and Computer Science (EECS) Department, MIT, where he leads the Network Coding and Reliable Communications Group, Research Laboratory for Electronics. She received the 2019 Best Paper Award for IEEE TRANSACTIONS ON NETWORK SCIENCE AND ENGINEERING, 2009 IEEE Communication Society and Information Theory Society Joint Paper Award, the 2009 William R. Bennett Prize in the Field of Communications Networking, the 2002 IEEE Leon K. Kirchmayer Prize Paper Award, the 2018 ACM SIGCOMM Test of Time Paper Award and several conference paper awards. She was a co-winner of the MIT 2004 Harold E. Egerton Faculty Achievement Award, received the 2013 EECS Graduate Student Association Mentor Award, and served as undergraduate Faculty in Residence for seven years. In 2007, she was named a Gilbreth Lecturer by the U.S. National Academy of Engineering. She received the 2016 IEEE Vehicular Technology James Evans Avant Garde Award, the 2017 Aaron Wyner Distinguished Service Award from the IEEE Information Theory Society and the 2017 IEEE Communications Society Edwin Howard Armstrong Achievement Award. She has co-founded CodeOn, Steinwurf, and Chocolate Cloud for technology transfer of network coding. She has served as an editor for many publications of the Institute of Electrical and Electronics Engineers (IEEE), of which she was an elected fellow, and she has served as the Editor-in-Chief of the IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS. She was President of the IEEE Information Theory Society in 2012, and served on its board of governors for eleven years. She has served as technical program committee co-chair of many of the major conferences in information theory, communications and networking. She was an elected Member of the National Academy of Engineering for her contributions to the theory and practice of network coding in 2020. She is a member of the National Academy of Inventors.