On Distributed Computation Rate Optimization for Deploying Cloud Computing Programming Frameworks

Jia Liu[†] Cathy H. Xia[‡] Ness B. Shroff[†*] Xiaodong Zhang[*]
[†]Dept. of Electrical and Computer Engineering
[‡]Dept. of Integrated Systems Engineering
[*]Dept. of Computer Science and Engineering
The Ohio State University, Columbus, OH 43210, U.S.A.
[†]{liu, shroff}@ece.osu.edu, [‡]xia.52@osu.edu, [*]zhang@cse.ohio-state.edu

ABSTRACT

With the rapidly growing challenges of big data analytics, the need for efficient and distributed algorithms to optimize cloud computing performances is unprecedentedly high. In this paper, we consider how to optimally deploy a cloud computing programming framework (e.g., MapReduce and Dryad) over a given underlying network hardware infrastructure to maximize the end-to-end computation rate and minimize the overall computation and communication costs. The main contributions in this paper are three-fold: i) we develop a new network flow model with a generalized flow-conservation law to enable a systematic design of distributed algorithms for computation rate utility maximization problems (CRUM) in cloud computing; ii) based on the network flow model, we reveal key separable properties of the dual functions of Problem CRUM, which further lead to a distributed algorithm design; and iii) we offer important networking insights and meaningful economic interpretations for the proposed algorithm and point out their connections to and distinctions from distributed algorithms design in traditional data communications networks. This paper serves as an important first step towards the development of a theoretical foundation for distributed computation analytics in cloud computing.

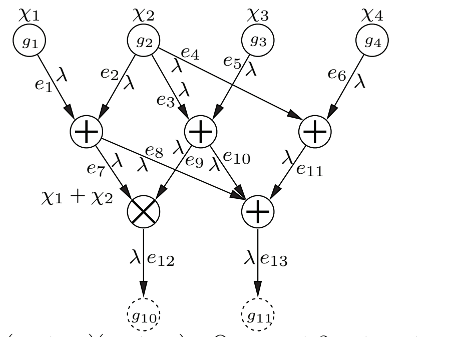
1. INTRODUCTION

With the rapid advances in information technologies, recent years have witnessed the growing challenges in storing, processing, and analyzing large data sets in many areas, such as social networks web-services, genomic research, network traffic management, complex physics simulations, environmental research, just to name a few [1, 2]. Traditional centralized relational database management systems, first developed more than four decades earlier, cannot handle the unstructured and dynamic nature of the large data sets nowadays and the performance of relational database management systems scales poorly as the data sets' sizes increase. As a result, distributed cloud computing platforms that have a large number of networked computing nodes with massively parallel structures have emerged as an attractive solution for handling big data analytics.

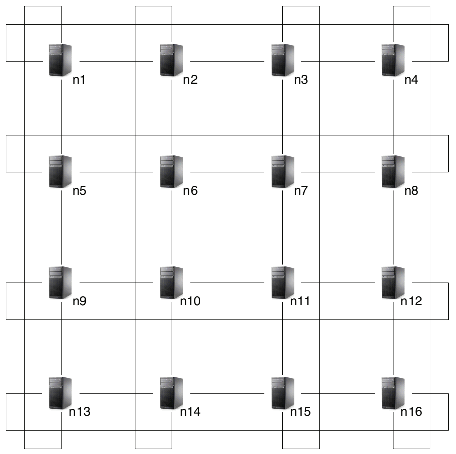
Among cloud computing programming frameworks for big data analytics, perhaps the most famous example is the so-called MapReduce [3], designed initially by Google for scanning large amounts of textual data to create web search indexes for the entire Internet. Another notable example is Dryad [4], which is Microsoft's counterpart to MapReduce. Dryad has been seen by some researchers as an approach to improve the pitfalls of the MapReduce framework [5] in that: i) it allows for general styles of computation (by using directed-acyclic graph (DAG)) that are much more than just "map" and "reduce" phases; and ii) it allows communications between stages to happen over more than just files stored in distributed file systems. Furthermore, recent research showed that MapReduce and Dryad can be mathematically unified under a well-defined matrix-based model [6].

However, as promising as they are, the actual deployments of cloud computing frameworks such as MapReduce and Dryad remain in their infancy and there are many technical issues to be resolved. For example, in the most popular MapReduce implementation known as "the Hadoop project" [7], there is only one active job tracker to schedule and monitor all "map" and "reduce" tasks. This not only poses the single-point-of-failure vulnerability, but also has a poor scalability that defeats the whole purpose of distributedness in cloud computing. To date, although there exist some heuristic design rule-of-thumbs (e.g., moving computation nodes closer to their data sources to avoid unnecessary communications), little effort has been made to establish a mathematical foundation to systematically develop *distributed* algorithms and control schemes that optimize the deployments and performances of cloud computing programming frameworks. Hence, the goal of our paper is to take the first step to fill this gap.

In this paper, we study how to optimally decompose and allocate the subcomputation tasks in a cloud computing programming framework over an underlying hardware infrastructure, such that the end-to-end computation rate can be maximized and the overall computation and communication costs can be minimized. Further, algorithms for solving this problem need to be implemented in a *distributed fashion.* Toward this end, motivated by Dryad (which also subsumes MapReduce as pointed out earlier), we model a cloud computing programming framework as computing a set of generic functions $\{\Theta_k\}$ that share a common set of distributed incoming data sources, where the data dependency can be represented by a multiple-input multiple-output directed acyclic graph (MIMO-DAG), as shown in the illus-



(a) An illustrative example of a cloud computing programming framework



(b) A 16-node 2-D torus interconnection topology.

Figure 1: An illustrative example of deploying a cloud computing programming framework over an underlying network infrastructure: (a) Computing functions Θ_1 and Θ_2 at computation rate λ; the data dependencies are represented by a MIMO-DAG structure; (b) An illustration of a 2-D torus interconnection topology commonly seen in large data centers or supercomputers.

trated example in Fig. 1(a).[1] On the other hand, the underlying networked system (upon which any cloud computing framework needs to be deployed) can also be represented by a graph. For example, Fig. 1(b) illustrates a 2-D torus interconnection topology commonly found in large data centers or supercomputers that support cloud computing jobs. As will be shown later, to capture the key features of cloud computing programming frameworks and incorporate computation and communication constraints of the underlying networks, we develop a generic mathematical modeling framework, based on which we formulate the computation rate utility maximization problem (CRUM). Our main results and contributions in this paper are three-fold:

- By accounting for in-network data aggregation flows, we show that one can construct a new network flow model with a *generalized flow-conservation law* to address both communication limits and computation costs in Problem CRUM. We point out that this utility framework and the associated generalized flow-conservation law are important in that they shares the same separable structure as in the classical network utility maximization problems in data communication networks [8, 9]. Thus, they enable the design of *polynomial-time* distributed algorithms for cloud computing by drawing experiences gained from the classical network utility maximization theory.
- By appropriately reformulating Problem CRUM in the Lagrangian dual domain, we reveal key separable properties of the dual function of Problem CRUM. These key structural properties enable us to obtain closed-form expressions for the primal and dual update schemes, which further leads to a fully distributed algorithmic implementation for big data analytics in cloud computing.
- We offer important networking insights and economics interpretations for the proposed distributed algorithm. We also point out the connections to and distinctions from the dual decomposition based distributed algorithms in the classical network utility maximization theory for data communications networks, thus further advancing our understanding of distributed approaches in cloud computing network optimization theory. We also provide numerical examples to show the efficacy of our proposed distributed algorithm.

To our knowledge, this paper is among the first that treat the design of distributed algorithms in MIMO-DAG based cloud computing frameworks (e.g., Dryad) via a rigorous theoretical network-flow optimization approach. The remainder of this paper is organized as follows. In Section 2, we review some related work in the literature, putting our work in a comparative perspective. Section 3 introduces the new network flow model with a generalized flow-conservation law for deploying cloud computing frameworks. Section 4 develops the principal components of our proposed distributed algorithm for solving Problem CRUM. Section 5 provides numerical results and Section 6 concludes this paper.

2. RELATED WORK

Our work is closely related to i) distributed cross-layer utility maximization theory for data communication networks (see, e.g., [10] for an overview) and ii) in-network computation techniques. In the in-network computation literature, our work is most related to [11], where Shah *et al.* developed a network flow model for the in-network computation in sensor network applications. The model in [11] extends the conventional flow-conservation law in the network flow literature [12] to in-network computation applications. However, the model in [11] is restricted to simple tree topologies that are used for data aggregation in sensor networks. In contrast, our network flow model works with generic MIMO-DAG, which are the most appropriate models for complex cloud computing programming frameworks, such as Dryad and MapReduce. Also, in our network model, we incorporate general network utility and communication/computation cost functions, which were not considered in [11].

Our network model also shares some similarities with the load shedding and distributed resource control problem of stream processing networks (e.g., [13–15]). But our work

[1]The addition and multiplication operations in Fig. 1(a) are only for illustrative purposes. Each vertex could be any general data processing operations.

differs from these works in the following two important aspects: First, although the issue of flow imbalance in stream processing networks was also pointed out in [14,15], the flow imbalance in [14,15] was caused by different flow production and consumption rates between upstream and downstream nodes. In contrast, the flow imbalance in this work is due to subcomputation in clouds, which is a fundamentally different cause. Second, in [13], the task-to-server assignment relationship is assumed to be given and the authors only studied the end-to-end utility rate maximization. In contrast, the task-to-sever assignment relationship is also part of the overall optimization in our work. Our network flow model also has connections with the graph embedding problems in graph theory (e.g., [16–18]). But these problems differ from ours in that their embedding objectives were to minimize some graph-theoretic performance metrics, such as dilation (i.e., the maximum distance in the network between adjacent tree nodes).

3. NETWORK MODEL AND PROBLEM FORMULATION

In this section, we first present the modeling details of cloud computing programming frameworks and the underlying network infrastructure in Section 3.1. Then, the concept of mapping between a programming framework and network infrastructure is introduced in Section 3.2. Next, we develop a new network flow model with a generalized flow-conservation law in Section 3.3. Finally, based on the network flow model, we formulate the computation rate optimization problem in Section 3.4.

3.1 Modeling Cloud Computing Frameworks and Network Infrastructure

As mentioned earlier, we model a cloud computing programming framework by a MIMO-DAG. Here, we denote a MIMO-DAG by $\mathcal{D} = \{\mathcal{V}, \mathcal{E}\}$, where $\mathcal{V}$ and $\mathcal{E}$ represent the sets of vertices and edges, respectively, with $|\mathcal{V}| = V$ and $|\mathcal{E}| = E$. We note that this MIMO-DAG structure captures the parallel processing and multi-stage aggregation nature of typical cloud computing programming frameworks, such as Dryad or multiple concatenations of Map/Reduce.

We suppose that there are S vertices of in-degree-zero in $\mathcal{D}$, corresponding to the S distributed input data sources. Without loss of generality (w.l.o.g), we label these source vertices as $g_1, \ldots, g_S$. For modeling convenience, there are K vertices in $\mathcal{D}$ having in-degree-one and out-degree-zero, corresponding to the virtual sinks that absorb the final output of Θ_k, $k = 1, \ldots, K$. Again, w.l.o.g, we label such sinks as $g_{V-K+1}, \ldots, g_V$. The remaining nodes $g_{S+1}, \ldots, g_{V-1}$ perform the computations prescribed by Θ. The input data stream at each source g_i is denoted as $\{\chi_i(k)\}_{k=0}^{\infty}$, where $\chi_i(k)$ represents the i-th input element at time instant k. The infinite input data streams could represent, for example, the "big data" phenomenon exemplified in the large data sets processing in by the MapReduce or Dryad frameworks in cloud computing. In this paper, all sources are assumed to be synchronized. We note that the synchronization in cloud computing systems is also an actively research topic (see, e.g., [19] and references therein) and its details are beyond the scope of this paper. For simplicity, we assume that all input data streams and all directed edges in the MIMO-DAG structure have a homogeneous input rate λ, which can also be thought of as the end-to-end computation rate of $\{\Theta_k\}$. We remark that the cases with heterogeneous rates (due to compression or expansion processing) can be exercised similarly by introducing coefficients in front of the data rates [14] in our utility maximization formulation described later. One can further transform such cases into a homogeneous case using modeling techniques in [13] by changing the measure and absorbing the coefficient into the resource usage parameters. Hence, this homogeneous rate assumption does not lose any generality.

We let $h(e)$ and $t(e)$ denote the head and tail vertices of each directed edge e in $\mathcal{E}$, respectively. We let $\mathcal{S} \triangleq \{e \in \mathcal{E} | h(e) \in \{g_1, \ldots, g_S\}\}$ be the set of all edges originating from the sources. We note that $|\mathcal{S}| \geq S$ in general. Likewise, we let $\mathcal{K} \triangleq \{e \in \mathcal{E} | t(e) \in \{g_{V-K+1}, \ldots, g_V\}\}$ denote the set of edges terminated at the sinks. Note that, since each sink edge represents a unique output (see Fig. 1(a)), we have $|\mathcal{K}| = K$. W.l.o.g, we label the edges in $\mathcal{E}$ in such a way that the first $|\mathcal{S}|$ edges $e_1, \ldots, e_{|\mathcal{S}|}$ and the last K edges $e_{E-K+1}, \ldots, e_E$ are the source and sink edges, respectively. In Fig. 1(a), for example, $\mathcal{S} = \{e_1, \ldots, e_6\}$ and $\mathcal{K} = \{e_{12}, e_{13}\}$.

We point out that each directed edge in $\mathcal{D}$ can be viewed as a subcomputation task. For example, in Fig. 1(a), edge e_7 corresponds to the computation result $\chi_1 + \chi_2$. Therefore, the terms "edge" and "subcomputation" are sometimes used interchangeably in this paper. It should be noted, however, that a subcomputation does not necessarily correspond to a unique edge. For example, in Fig. 1(a), edges e_7 and e_8 all carry the same subcomputation $\chi_1+\chi_2$. The set of successor edges of e in $\mathcal{D}$ is defined as $\Psi(e) \triangleq \{e' \in \mathcal{E} | t(e') = h(e)\}$. For example, in Fig. 1(a), the successor edges of e_2 are e_7 and e_8. Note that from the structure of $\mathcal{D}$, we have $\Psi(e_i) = \varnothing$ for all $e_i \in \mathcal{K}$. Finally, we remark that the MIMO-DAG structure degenerates into a tree topology if it has a single sink and each edge only has one successor edge. Therefore, the tree-structure model considered in [11] can be viewed as a special case of our model.

On the other hand, in cloud computing environments (e.g., data centers or supercomputers), we have a networked system as the physical computing and communication infrastructure (i.e., a cloud computing hardware platform). Usually, the hardware platform also exhibits certain carefully constructed parallel structures. For example, Fig. 1(b) illustrates a 2-D torus interconnection topology that is commonly seen in cloud hardware platforms. Advanced supercomputers nowadays could employ torus interconnections with even higher dimensions (e.g., 5-D torus in IBM Bue Gene/Q architecture). But we point out that these special topology properties are not essential to our network models and our distributed algorithm design, both of which apply to general interconnection topologies. In this paper, we also model the underlying cloud computing hardware platform by a graph $\mathcal{G} = \{\mathcal{N}, \mathcal{L}\}$, where $\mathcal{N}$ and $\mathcal{L}$ are the sets of nodes and links with $|\mathcal{N}| = N$ and $|\mathcal{L}| = L$, respectively. Depending on the size and scale of the hardware system, each node in $\mathcal{N}$ could represent, e.g., a CPU core, a node card, or even a computer rack. Each link in $\mathcal{L}$ represents the communication connection between the nodes. It could represent a local high-speed bus if the nodes are co-located on the same card or a fiber link if the nodes are located at different racks.

As mentioned earlier, to avoid unnecessary data communications in cloud computing models (such as Map/Reduce and Dryad), it is highly desirable to allocate subcomputation tasks involving input data sources and outputs at their physical locations. Therefore, w.l.o.g, we label the nodes of $\mathcal{N}$ as $n_1, \ldots, n_N$ in such a way that the first S nodes $n_1, \ldots, n_S$ correspond to the physical locations of the S data sources of $\mathcal{D}$.

3.2 Mapping a Cloud Computing Framework onto a Network Infrastructure

Clearly, deploying a given cloud computing framework on a cloud computing hardware platform amounts to mapping *all edges* in $\mathcal{D}$ onto $\mathcal{G}$ in an appropriate fashion. As in standard graph theory terms, we define a path P as a sequence of nodes in $\mathcal{N}$ such that every two adjacent nodes n_k and n_l in P satisfy $(n_k, n_l) \in \mathcal{L}$. The first and the last nodes in P are called the start and end nodes and are denoted as $\alpha(P)$ and $\beta(P)$, respectively. In the extreme case, P could contain only one node, say n_k. In this case, $\alpha(P) = \beta(P) = n_k$ and the link (n_k, n_k) degenerates into a self-loop. Here, if an edge is mapped to a path P, it means that the data are transmitted from $\alpha(P)$ via the specified links to $\beta(P)$ and then the corresponding subcomputation is performed at $\beta(P)$. In the special case, if an edge is mapped to a self-loop, then the corresponding subcomputation task is performed locally and no communication is needed.

We denote the set of all paths in $\mathcal{G}$ as $\mathcal{P}$. A *valid mapping* of $\mathcal{D}$ onto $\mathcal{G}$ is defined as follows:

Definition 1. *A mapping $M : \mathcal{E} \to \mathcal{P}$ is valid if: (1) $\alpha(M(e_i)) = n_k$ if $e_i \in \mathcal{S}$ and $h(e_i) = g_k$, $k = 1, \ldots, S$; (2) $\beta(M(e_i)) = n_k$ if $e_i \in \mathcal{K}$ and n_k is the physical output location of e_i; and (3) $\alpha(M(e_j)) = \beta(M(e_i))$ if $e_j \in \Psi(e_i)$.*

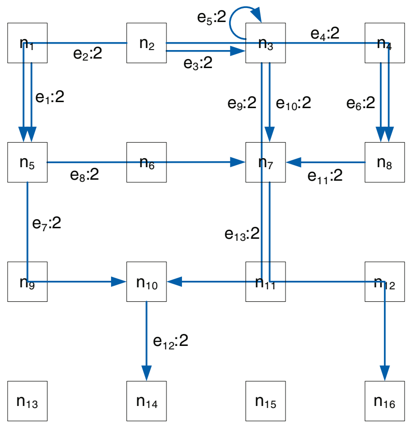
In Definition 1, conditions (1) and (2) imply that the source and the sink nodes in $\mathcal{D}$ have to match to their physical locations in the underlying network, and condition (3) represents that the logical successor relationships of the edges in $\mathcal{D}$ need to be respected after the mapping.

In general, there are many different ways to map $\mathcal{D}$ onto $\mathcal{G}$. For example, it is not difficult to see that Figs. 2(a) and 2(b) illustrate two valid mappings of the MIMO-DAG structure in Fig. 1(a) onto the network in Fig. 1(b). Note that in a valid mapping, the node sequence corresponding to an edge in $\mathcal{D}$ could consist of multiple connected links in $\mathcal{G}$. For example, in Figure 2(b), edge e_8 consists of links (n_5, n_9), (n_9, n_{10}), and (n_{10}, n_{11}).

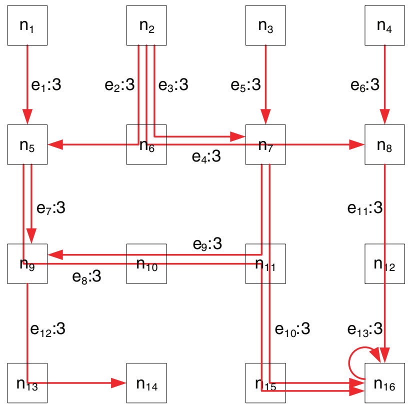
We note that each mapping may use a different computation rate, as shown in Figs. 2(a) and 2(b). For example, in Fig. 2(a), the label "$e_i : 2$" represents that the computation rate of each edge e_i is 2 in this mapping. Further, a time-sharing among valid mappings also yields a valid mapping. For instance, Fig. 3 shows a time-sharing between the two mappings in Figs. 2(a) and 2(b), each accounting for 50% of time. It is not difficult to see that the computation rate of this time-sharing mapping is $0.5 \times 2 + 0.5 \times 3 = 2.5$.

3.3 A Network Flow Model with a Generalized Flow-Conservation Law

It can be seen from the above discussions that the *computation rate region* of a cloud computing framework over a given network infrastructure can be exhausted by all time-sharing strategies among all possible valid mappings. As a result, any performance metrics related to the computation rate can be optimized by determining an optimal time-sharing strategy. However, since the total number of valid mappings is exponential in N, finding an optimal time-sharing strategy directly is intractable. Therefore, we need to exploit additional structure of the networked system to address this challenge.



(a) Mapping 1 with computation rate $\lambda = 2$.



(b) Mapping 2 with computation rate $\lambda = 3$.

Figure 2: Two valid mappings of the MIMO-DAG structure in Fig. 1(a) onto the network in Fig. 1(b). Each mapping has a different computation rate.

To this end, our basic idea is to develop a new *network flow model* with a *generalized flow-conservation law* for computation analytics. The fundamental rationale behind this approach is that, similar to the classical model in the multi-commodity network flow literature [12], the structural properties of the new network flow model would also lead to polynomial time solutions if designed appropriately.

Unfortunately, developing a flow-conserved network flow model for computation analytics is not-trivial. Due to performing computations at each node in the network, the classical flow-conservation law [12] fails to hold in computation analytics network flows. For example, from the zoom-in view for node n_{10} in Fig. 4, it can be seen that the total incoming flow rate is:

$$x^{(e_7)}_{(n_9,n_{10})} + x^{(e_8)}_{(n_9,n_{10})} + x^{(e_9)}_{(n_{10},n_{11})} + x^{(e_9)}_{(n_{10},n_{11})} = 5.$$

However, due to the addition computations performed at

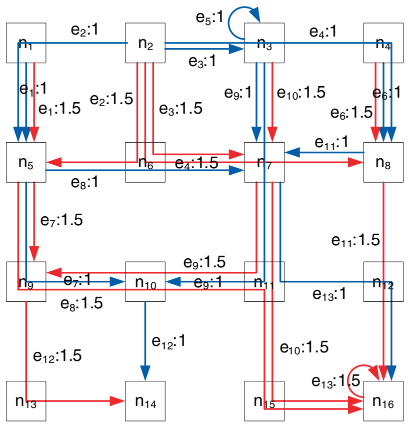


Figure 3: A time-sharing between two mappings in Fig. 2(a) and Fig. 2(b), each accounting for 50% of time.

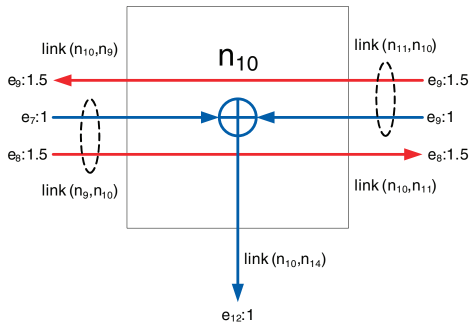


Figure 4: Zoom-in view of node n_{10} in Fig. 3. Due to the addition computations, the total incoming and outgoing flow rates at n_{10} are not equal.

node n_{10}, the total outgoing flow rate is:

$$x^{(e_9)}_{(n_{10},n_9)} + x^{(e_8)}_{(n_{10},n_{11})} + x^{(e_{12})}_{(n_{10},n_{14})} = 4.$$

This shows that, for computation analytics in cloud computing, the total input and output network flow are unbalanced.

Although the traditional flow-conservation law no longer holds here, upon a closer look at each incoming edge and their corresponding outgoing or successor edges at n_7, it is not difficult to observe another form of flow-conservation law. As shown in Fig. 4, for the incoming edge e_4 that does not involve any computation at n_7, we have:

$$x^{(e_9)}_{(n_{11},n_{10})} = x^{(e_9)}_{(n_{10},n_9)},$$

i.e., the incoming rate equals the outgoing rate. The same is also true for the incoming edge e_8. On the other hand, for the incoming edge e_7 that involves in the addition with incoming edge e_9, we have:

$$x^{(e_7)}_{(n_9,n_{10})} = x^{(e_{12})}_{(n_{10},n_{14})} \quad \text{and} \quad x^{(e_9)}_{(n_{11},n_{10})} = x^{(e_{12})}_{(n_{10},n_{14})},$$

i.e., the outgoing rate of the successor edge of e_7 is equal to the incoming rate of e_7.

To model this new form of flow-conservation law, it is important to recognize that: for a subcomputation, say e, that is injecting to node n, a portion of its flow is terminated at n to compute the successor edges of e (i.e., $\Psi(e)$). For convenience, we label the links in the physical network as $1, \ldots, L$ instead of using node pairs (n_j, n_k). We let $x^{(e)}_l \geq 0$ represent the flow amount of subcomputation task e on link l. In all valid mappings M, since the start node of a source edge $e_i \in \mathcal{S}$ and the end node of a sink edge $e_{i'} \in \mathcal{K}$ are always mapped to their respective physical nodes in the network, we let $\text{Src}(e_i) \triangleq \alpha(M(e_i))$ and $\text{Dst}(e_{i'}) \triangleq \beta(M(e_{i'}))$ be the physical source and end nodes of e_i and $e_{i'}$ for simplicity. Then, based on the previous observations, we have the following result:

Lemma 1. *When computation analytics are performed over an underlying networked system, the following generalized flow-conservation law holds:*

$$\sum_{l\in\mathcal{O}(n_k)} x^{(e_i)}_l + y^{(e_j)}_{n_k} - \sum_{l\in\mathcal{I}(n_k)} x^{(e_i)}_l - y^{(e_i)}_{n_k} = \lambda,\ \forall e_i \in \mathcal{S},$$
$$\forall e_j \in \Psi(e_i),\ n_k = \text{Src}(e_i), \quad (1)$$

$$\sum_{l\in\mathcal{O}(n_k)} x^{(e_i)}_l + y^{(e_j)}_{n_k} - \sum_{l\in\mathcal{I}(n_k)} x^{(e_i)}_l - y^{(e_i)}_{n_k} = 0,\ \forall e_i \in \mathcal{E}\backslash\mathcal{K},$$
$$\forall e_j \in \Psi(e_i),\ \textit{and}\ n_k \neq \text{Src}(e_i)\ \textit{if}\ e_i \in \mathcal{S}, \quad (2)$$

$$\sum_{l\in\mathcal{O}(n_k)} x^{(e_i)}_l - \sum_{l\in\mathcal{I}(n_k)} x^{(e_i)}_l - y^{(e_i)}_{n_k} = 0,\ \forall e_i \in \mathcal{K},\ n_k \neq \text{Dst}(e_i), \quad (3)$$

$$\sum_{l\in\mathcal{O}(n_k)} x^{(e_i)}_l - \sum_{l\in\mathcal{I}(n_k)} x^{(e_i)}_l - y^{(e_i)}_{n_k} = -\lambda,$$
$$\forall e_i \in \mathcal{K},\ n_k = \text{Dst}(e_i), \quad (4)$$

where $\mathcal{O}(n_k)$ and $\mathcal{I}(n_k)$ represent the sets of outgoing and incoming links at node n_k, respectively; and $y^{(e_i)}_{n_k}$ represents the subcomputation e_i generated at node n_k.

Here, the equalities in (1) and (2) represent that the total incoming flow rate of any non-sink edge e_i at node n_k should be equal to the sum of the outgoing flow rates of e_i and its successor edge e_j, and this relationship holds for every successor edge of e_i. Moreover, Eq.(1) states that if e_i is a source edge and n_k happens to be its source node, then the net injection rate of e_i at n_k is λ. On the other hand, Eqs.(3) and (4) say that, for a sink edge e_i that does not have any successor edge, the net output flow rate should be equal to λ at its physical location and zero elsewhere.

3.4 Problem Formulation

We let C_l denote the capacity of link l. Since the total network flow traversing a link cannot exceed the link's capacity, we have $\sum_{i=1}^{E} x^{(e_i)}_l \leq C_l$, $l = 1, \ldots, L$. We let the network utility be a function of λ, denoted by $U(\lambda) : \mathbb{R}_+ \to \mathbb{R}$. We assume that $U(\lambda)$ is concave, monotonically increasing, and twice continuously differentiable. The concavity of $U(\cdot)$ represents the "diminishing returns" effect. When $U(\cdot)$ is linear (as a special case of concavity), we are simply maximizing the computation rate itself.

In the physical network, each outgoing edge at a node represents a subcomputation (see, e.g., edges e_9, e_{10}, and e_{13} in Fig. 4), which incurs certain costs (e.g., consumed energy per unit amount of computation). We let ρ_k represent the unit cost for performing computation at node n_k. Then, the total cost due to computation at node n_k can be computed as:

$$\sum_{i=1}^{|\mathcal{S}|}\rho_k\left(\lambda+y_{n_k}^{(e_i)}+\sum_{l\in\mathcal{I}(n_k)}x_l^{(e_i)}-\sum_{l\in\mathcal{O}(n_k)}x_l^{(e_i)}\right)+\sum_{i=|\mathcal{S}|+1}^{E-K}\rho_k\left(y_{n_k}^{(e_i)}+\sum_{l\in\mathcal{I}(n_k)}x_l^{(e_i)}-\sum_{l\in\mathcal{O}(n_k)}x_l^{(e_i)}\right). \quad (5)$$

In (5), the term $\lambda+y_{n_k}^{(e_i)}+\sum_{l\in\mathcal{I}(n_k)}x_l^{(e_i)}-\sum_{l\in\mathcal{O}(n_k)}x_l^{(e_i)}$ is the total flow rate of source edge e_i that is terminated at node n_k and used to compute its successor edges. Likewise, the term $y_{n_k}^{(e_i)}+\sum_{l\in\mathcal{I}(n_k)}x_l^{(e_i)}-\sum_{l\in\mathcal{O}(n_k)}x_l^{(e_i)}$ represents the total flow rate of non-source and non-sink edge e_i terminated at n_k. Our objective is to maximize the net network utility, defined as network utility minus the total costs. Putting together the discussions earlier, we can formulate the problem as follows:

CRUM:

$$\text{Max } U(\lambda)-\sum_{k=1}^{N}\left(\sum_{i=1}^{|\mathcal{S}|}\rho_k\left(\lambda+y_{n_k}^{(e_i)}+\sum_{l\in\mathcal{I}(n_k)}x_l^{(e_i)}-\sum_{l\in\mathcal{O}(n_k)}x_l^{(e_i)}\right)+\sum_{i=|\mathcal{S}|+1}^{E-K}\rho_k\left(y_{n_k}^{(e_i)}+\sum_{l\in\mathcal{I}(n_k)}x_l^{(e_i)}-\sum_{l\in\mathcal{O}(n_k)}x_l^{(e_i)}\right)\right)$$

s.t. Eqs. (1), (2), (3), (4),

$$\sum_{i=1}^{E}x_l^{(e_i)}\le C_l, \qquad \forall l=1,\ldots,L,$$

$$x_l^{(e_i)}\ge 0,\ \forall i,l;\ \lambda\ge 0.$$

Now, it is not difficult to recognize that with the proposed network flow model in Lemma 1, Problem CRUM is a convex program. Moreover, the nice separable structure of the objective function enables the design of distributed algorithm to solve Problem CRUM, which will be the focus of the next section.

4. DISTRIBUTED SOLUTION PROCEDURE

In this section, we will present the key steps in designing a distributed algorithm based on dual decomposition to solve Problem CRUM. In Section 4.1, we will first reformulate Problem CRUM into its Lagrangian dual problem and show how to appropriately decompose the Lagrangian dual problem. Based on these results, we introduce the basic idea in designing a distributed algorithm in Section 4.2. In Section 4.3, we offer some interesting networking and economics interpretations of the proposed distributed algorithm. Then, we will present some numerical results in Section 5.

4.1 Lagrangian Dual Reformulation and Decomposition

As mentioned earlier, since Problem CRUM is a convex program, it can be equivalently solved in its Lagrangian dual domain because of a zero duality gap [20]. To solve the Problem CRUM in its Lagrangian dual domain, we first slightly modify the constraints in (1)–(4) into inequality constraints as follows:

$$\sum_{l\in\mathcal{O}(n_k)}x_l^{(e_i)}+y_{n_k}^{(e_j)}-\sum_{l\in\mathcal{I}(n_k)}x_l^{(e_i)}-y_{n_k}^{(e_i)}\ge\lambda,\ \forall e_i\in\mathcal{S},\quad \forall e_j\in\Psi(e_i),\ n_k=\mathrm{Src}(e_i), \quad (6)$$

$$\sum_{l\in\mathcal{O}(n_k)}x_l^{(e_i)}+y_{n_k}^{(e_j)}-\sum_{l\in\mathcal{I}(n_k)}x_l^{(e_i)}-y_{n_k}^{(e_i)}\ge 0,\ \forall e_i\in\mathcal{E}\backslash\mathcal{K},\quad \forall e_j\in\Psi(e_i),\text{ and } n_k\ne\mathrm{Src}(e_i)\text{ if } e_i\in\mathcal{S}, \quad (7)$$

$$\sum_{l\in\mathcal{O}(n_k)}x_l^{(e_i)}-\sum_{l\in\mathcal{I}(n_k)}x_l^{(e_i)}-y_{n_k}^{(e_i)}\ge 0,\ \forall e_i\in\mathcal{K},\ n_k\ne\mathrm{Dst}(e_i), \quad (8)$$

$$\sum_{l\in\mathcal{O}(n_k)}x_l^{(e_i)}-\sum_{l\in\mathcal{I}(n_k)}x_l^{(e_i)}-y_{n_k}^{(e_i)}\ge-\lambda,\quad \forall e_i\in\mathcal{K},\ n_k=\mathrm{Dst}(e_i), \quad (9)$$

It is not difficult to show that these modifications do not affect the solution at optimality. Interestingly, these modifications can also be interpreted from a network stability perspective: the total service rate at each node is no less than the total arrival rate.

Next, we associate dual variables $\mu_k^{(ij)}\ge 0$ and $w_k^{(i)}\ge 0$ for each constraint in (6)–(7) and (8)–(9), respectively. For notational simplicity, we use Ψ^i to represent the index set $\{j : e_j\in\Psi(e_i)\}$. Also, we use vectors $\mathbf{x}$, $\boldsymbol{\mu}$, and $\mathbf{w}$ to group all x-, μ- and w-variables. By accommodating the constraints into the objective function and combining related terms, we have that the Lagrangian can be written as follows:

$$\begin{aligned}L(\lambda,\mathbf{x},\boldsymbol{\mu},\mathbf{w})=&U(\lambda)-\sum_{k=1}^{N}\left(\sum_{i=1}^{|\mathcal{S}|}\rho_k\left(\lambda+y_{n_k}^{(e_i)}+\sum_{l\in\mathcal{I}(n_k)}x_l^{(e_i)}-\sum_{l\in\mathcal{O}(n_k)}x_l^{(e_i)}\right)+\sum_{i=|\mathcal{S}|+1}^{E-K}\rho_k\left(y_{n_k}^{(e_i)}+\sum_{l\in\mathcal{I}(n_k)}x_l^{(e_i)}-\sum_{l\in\mathcal{O}(n_k)}x_l^{(e_i)}\right)\right)\\&+\sum_{i=1}^{E}\sum_{j\in\Psi^i}\sum_{k=1}^{N}\mu_k^{(ij)}\left(\sum_{l\in\mathcal{O}(n_k)}x_l^{(e_i)}+y_{n_k}^{(e_j)}-\sum_{l\in\mathcal{I}(n_k)}x_l^{(e_i)}-y_{n_k}^{(e_i)}\right)\\&+\sum_{i=E-K-1}^{E}\sum_{k=1}^{N}w_k^{(i)}\left(\sum_{l\in\mathcal{O}(n_k)}x_l^{(e_i)}-\sum_{l\in\mathcal{I}(n_k)}x_l^{(e_i)}-y_{n_k}^{(e_i)}\right)\\&+(-\lambda)\sum_{i=1}^{|\mathcal{S}|}\sum_{j\in\Psi^i}\mu_{\mathrm{Src}(i)}^{(ij)}+\lambda\sum_{i=E-K+1}^{E}w_{\mathrm{Dst}(i)}^{(i)}. \end{aligned} \quad (10)$$

Then, the Lagrangian dual function can be written as:

$$\Phi(\boldsymbol{\mu},\mathbf{w})=\sup_{\lambda,\mathbf{x}}\left\{L(\lambda,\mathbf{x},\boldsymbol{\mu},\mathbf{w})\;\middle|\;\begin{array}{l}\sum_{i=1}^{E}x_l^{(e_i)}\le C_l,\ \forall l,\\ x_l^{(e_i)}\ge 0,\forall i,l;\lambda\ge 0.\end{array}\right\}. \quad (11)$$

Finally, the dual problem can be written as:

$$\begin{array}{rl}\textbf{D:}\ \text{Minimize} & \Phi(\mathbf{u})\\ \text{subject to} & \mathbf{u}\ge\mathbf{0}.\end{array} \quad (12)$$

Next, it is important to recognize that the Lagrangian function in (11) possesses a *decomposable* structure, which leads to a distributed computation scheme. More specifically, by appropriately switching summation orders and rearranging terms in (11), we have the following result (see Appendix A for proof details):

PROPOSITION 2. *The Lagrangian in (11) can be decomposed in a source-wise and link-wise fashion as follows:*

$$\Phi(\boldsymbol{\mu},\mathbf{w}) = \Phi_{\mathrm{FC}}(\boldsymbol{\mu},\mathbf{w}) + \sum_{l=1}^{L}\Phi_{\mathrm{R}}^{(l)}(\boldsymbol{\mu},\mathbf{w}) + \sum_{k=1}^{N}\Phi_{\mathrm{C}}^{(k)}(\boldsymbol{\mu},\mathbf{w}),$$

where $\Phi_{\mathrm{FC}}(\boldsymbol{\mu},\mathbf{w})$, $\Phi_{\mathrm{R}}^{(l)}(\boldsymbol{\mu},\mathbf{w})$, and $\Phi_{\mathrm{C}}^{(n)}(\boldsymbol{\mu},\mathbf{w})$ represent the flow control, per-link routing, and per-node computation subproblems at each source, each link, and each node, respectively. Here, $\Phi_{\mathrm{FC}}(\boldsymbol{\mu},\mathbf{w})$ and $\Phi_{\mathrm{R}}^{(l)}(\boldsymbol{\mu},\mathbf{w})$ are defined as follows, respectively:

$$\Phi_{\mathrm{FC}}(\boldsymbol{\mu},\mathbf{w}) \triangleq \max_{\lambda\geq 0}\Bigg\{U(\lambda) - \lambda\Bigg[\sum_{i=1}^{|\mathcal{S}|}\Big(\sum_{j\in\Psi^i}\mu_{\mathrm{Src}(i)}^{(ij)} + \sum_{k=1}^{N}\rho_k\Big) - \sum_{i=E-K+1}^{E} w_{\mathrm{Dst}(i)}^{(i)}\Bigg]\Bigg\};$$

$$\Phi_{\mathrm{R}}^{(l)}(\boldsymbol{\mu},\mathbf{w}) \triangleq \Bigg\{\max\sum_{i=1}^{E}\Big(\tilde{w}_{\mathrm{Tx}(l)}^{(i)} - \tilde{w}_{\mathrm{Rx}(l)}^{(i)}\Big)x_l^{(e_i)} \quad \text{s.t.} \ \sum_{i=1}^{E}x_l^{(e_i)} \leq C_l, x_l^{(e_i)} \geq 0, \forall i.\Bigg\},$$

where $\tilde{w}_k^{(i)}$, $\forall k=1,\ldots,N$, $\forall i=1,\ldots,E$, are defined as:

$$\tilde{w}_k^{(i)} \triangleq \begin{cases}\rho_k + \sum_{j\in\Psi^i}\mu_k^{(ij)}, & i=1,\ldots,E-K,\\ w_k^{(i)} + \sum_{j\in\Psi^i}\mu_k^{(ij)}, & i=E-K+1,\ldots,E;\end{cases} \tag{13}$$

$$\Phi_{\mathrm{C}}^{(k)}(\boldsymbol{\mu},\mathbf{w}) \triangleq \max_{y_{n_k}^{(e_i)}\geq 0,\forall i}\Bigg\{\sum_{i=1}^{E}\Bigg[\sum_{j\in\Psi^i}\mu_k^{(ij)}(y_{n_k}^{(e_j)} - y_{n_k}^{(e_i)}) - \hat{w}_k^{(i)}y_{n_k}^{(e_i)}\Bigg]\Bigg\},$$

where $\hat{w}_k^{(i)}$, $\forall k=1,\ldots,N$, $\forall i=1,\ldots,E$, are defined as:

$$\tilde{w}_k^{(i)} \triangleq \begin{cases}\rho_k, & i=1,\ldots,E-K,\\ w_k^{(i)}, & i=E-K+1,\ldots,E.\end{cases} \tag{14}$$

Based on Proposition 2, the Lagrangian dual problem D in (12) can be transformed into the following master dual problem:

$$\textbf{MD:}\ \text{Minimize}\ \Phi_{FC}(\boldsymbol{\mu},\mathbf{w}) + \sum_{l=1}^{L}\Phi_R^{(l)}(\boldsymbol{\mu},\mathbf{w}) + \sum_{k=1}^{N}\Phi_C^{(k)}(\boldsymbol{\mu},\mathbf{w})$$
$$\text{subject to}\ \boldsymbol{\mu}\geq 0,\ \mathbf{w}\geq 0.$$

Then, the task of solving the Lagrangian dual problem D boils down to distributedly solving the subproblems $\Phi_{FC}(\boldsymbol{\mu},\mathbf{w})$, $\Phi_R^{(l)}(\boldsymbol{\mu},\mathbf{w})$, and $\Phi_C^{(k)}(\boldsymbol{\mu},\mathbf{w})$, and then the master dual problem MD.

4.2 Design of Distributed Algorithm

Note that at each source node, the flow control and computation subproblems Φ_{FC} has a concave objective function with a single non-negative decision variable. Thus, Φ_{FC} can be trivially and efficiently solved (e.g., by simple bisection search method). Also, it can be observed that the routing subproblem $\Phi_R^{(l)}$ at each link is a lower dimensional linear programming problem (with a single constraint, $O(EV^2)$ variables, and all problem coefficients are locally available), which means that it can be efficiently solved as well. For the master dual problem MD, since its objective function is piece-wise differentiable, one can apply subgradient method [20]. More specifically, starting with an initial $\boldsymbol{\mu}^{(0)}$ and $\mathbf{w}^{(0)}$ and after evaluating $\Phi_{FC}(\boldsymbol{\mu}^{(m)},\mathbf{w}^{(m)})$ and $\Phi_R^{(l)}(\boldsymbol{\mu}^{(m)},\mathbf{w}^{(m)})$ in the m-th iteration, we update the dual variables as follows:

$$\mu_k^{(ij)}(m+1) = \max\{\mu_k^{(ij)}(m) - \pi^m d_{\mu,k}^{(ij)}(m), 0\}, \tag{15}$$
$$w_k^{(i)}(m+1) = \max\{w_k^{(i)}(m) - \pi^m d_{w,k}^{(i)}(m), 0\}, \tag{16}$$

where $\pi^m > 0$ denotes a positive scalar step size, and $d_{\mu,k}^{(ij)}(m)$ and $d_{w,k}^{(ij)}(m)$ represent the subgradients of the dual variables $\mu_k^{(ij)}$ and $w_k^{(i)}$ in the m-th iteration, respectively.

It is known that for the subgradient iterative schemes in (15) and (16) to converge, a sufficient condition is that the step size π^m satisfies $\pi^m \to 0$ as $m\to\infty$, $\sum_{m=0}^{\infty} = \infty$, and $\sum_{m=0}^{\infty}(\pi^m)^2$ [20]. A popular step size selection strategy is the divergent harmonic series: $\pi^m = \frac{\beta}{m}$, $m=1,2,\ldots$, where β is some given system parameter. For the master dual problem MD, the subgradient for the Lagrangian dual function in (10) can be computed as:

$$d_{\mu,k}^{(ij)}(m) = \sum_{l\in\mathcal{O}(n_k)}x_l^{(e_i)} + y_{n_k}^{(e_j)} - \sum_{l\in\mathcal{I}(n_k)}x_l^{(e_i)} - y_{n_k}^{(e_i)} - \lambda\mathbb{1}_{\{e_i\in\mathcal{S},n_k=\mathrm{Src}(e_i)\}}, \tag{17}$$
$$d_{w,k}^{(i)}(m) = \sum_{l\in\mathcal{O}(n_k)}x_l^{(e_i)} - \sum_{l\in\mathcal{I}(n_k)}x_l^{(e_i)} - y_{n_k}^{(e_i)} + \lambda\mathbb{1}_{\{e_i\in\mathcal{S},n_k=\mathrm{Dst}(e_i)\}}, \tag{18}$$

where $\mathbb{1}_{\{\cdot\}}$ represents the indicator function, which equals 1 if the condition in $\{\cdot\}$ is satisfied and 0 otherwise. To summarize, the design of distributed subgradient algorithm for solving Problem CRUM is illustrated in Algorithm 1.

Algorithm 1 A subgradient algorithm for solving Problem CRUM.

```
Initialization:
 1. Choose initial starting points $\boldsymbol{\mu}^{(0)}$ and $\mathbf{w}^{(0)}$. Let $m = 0$.
Main Iteration:
 2. Compute the source computation rate $\lambda(m)$ by solving
    the flow control subproblem $\Phi_{FC}(\boldsymbol{\mu}^{(m)},\mathbf{w}^{(m)})$ by using,
    e.g., bisection method.
 3. Compute the routing decisions $x_l^{(e_i)}(m)$ at each link by
    solving the routing subproblem $\Phi_R(\boldsymbol{\mu}^{(m)},\mathbf{w}^{(m)})$, a linear
    programming problem.
 4. Choose an appropriate step size $\pi^m$. Compute the sub-
    gradients $d_{\mu,k}^{(ij)}(m)$ and $d_{w,k}^{(i)}(m)$ using (17) and (18) with
    $\lambda(m)$ and $x_l^{(e_i)}(m)$.
 5. Update dual variables $\boldsymbol{\mu}^{(m+1)}$ and $\mathbf{w}^{(m+1)}$ with $d_{\mu,k}^{(ij)}(m)$
    and $d_{w,k}^{(i)}(m)$.
 6. If $\|\boldsymbol{\mu}^{(m+1)} - \boldsymbol{\mu}^{(m)}\| < \epsilon$ and $\|\mathbf{w}^{(m+1)} - \mathbf{w}^{(m)}\| < \epsilon$, then
    return $\lambda(m)$ and $x_l^{(e_i)}(m)$. Otherwise, let $k \leftarrow k+1$ and
    go to Step 2.
```

4.3 Networking and Economics Interpretations

Several interesting networking and economics insights for the subgradient-based distributed algorithm are in order.

Queue length interpretations of the dual variables:

It can be seen that by dividing the step size π^m on both sides of (15) and (16) and letting $Q_k^{(ij)}(m) \triangleq \mu_k^{(ij)}/\pi^m$ and $\bar{Q}_k^{(i)}(m) \triangleq w_k^{(i)}/\pi^m$, we have:

$$Q_k^{(ij)}(m+1) = \max\Bigg\{ Q_k^{(ij)}(m) - \sum_{l\in\mathcal{O}(n_k)} x_l^{(e_i)} - y_{n_k}^{(e_j)} + y_{n_k}^{(e_i)} + \sum_{l\in\mathcal{I}(n_k)} x_l^{(e_i)} + \lambda \mathbb{1}_{\{e_i\in\mathcal{S}, n_k=\mathrm{Src}(e_i)\}}, 0 \Bigg\}, \quad (19)$$

$$\bar{Q}_k^{(i)}(m+1) = \max\Bigg\{ \bar{Q}_k^{(i)}(m) - \sum_{l\in\mathcal{O}(n_k)} x_l^{(e_i)} - \lambda \mathbb{1}_{\{e_i\in\mathcal{S}, n_k=\mathrm{Dst}(e_i)\}} + \sum_{l\in\mathcal{I}(n_k)} x_l^{(e_i)} + y_{n_k}^{(e_i)}, 0 \Bigg\}. \quad (20)$$

A closer look reveals that (19) and (20) behave exactly the same as "queue length evolution" as seen in traditional data communications networks. Indeed, if we let $Q_k^{(ij)}(m)$ represent the queuing backlog for computing the successive edge j of edge i at node k in time instant m, then this queuing backlog and the dual variable $\mu_k^{(ij)}(m)$ are intimately related (differ only by a scaling factor π^m). By the same token, $\bar{Q}_k^{(i)}(m)$ can be similarly interpreted as the queuing backlog of the terminating edges $e_i \in \mathcal{K}$ at node m.

Connections to the back-pressure algorithm: Clearly, in cases where in-network computation is disabled, i.e., by forcing $x_l^{(e_j)} = 0$ for all $e_j \in \Psi(e_i)$ and for all l, the network degenerates into a traditional communication network. As expected, it is easy to check that the queue length evolutions in (19) and (20) reduce to the conventional queue length evolution:

$$Q_k^{(i)}(m+1) = \max\Bigg\{ Q_k^{(i)}(m) - \sum_{l\in\mathcal{O}(n_k)} x_l^{(e_i)} + \sum_{l\in\mathcal{I}(n_k)} x_l^{(e_i)} + \lambda\Big(\mathbb{1}_{\{e_i\in\mathcal{S}, n_k=\mathrm{Src}(e_i)\}} - \mathbb{1}_{\{e_i\in\mathcal{S}, n_k=\mathrm{Dst}(e_i)\}}\Big), 0 \Bigg\}. \quad (21)$$

Moreover, since all dual μ-variables become 0, it is easy to verify that the computation subproblem $\Phi_C^{(k)}$ admits a trivial optimal solution: $y_{n_k}^{(e_i)} = 0$, $\forall k, i$. Also, with dual μ-variables being 0, it is not difficult to see that the routing subproblem $\Phi_{\mathrm{R}}^{(l)}(\mathbf{w})$ admits a trivial solution as follows: for a given link l, pick a subcomputation task e_i that has the largest $(\tilde{w}_{\mathrm{Tx}(l)}^{(i)} - \tilde{w}_{\mathrm{Rx}(l)}^{(i)})$-value, say e_i^*, and let e_i^* use up the link capacity C_l. This is exactly the same strategy used in the celebrated "back-pressure" algorithm that was first discovered in [21].

Pricing interpretation of the dual subgradient updating scheme: The dual variables $\mu_k^{(ij)}(m)$ and $w_k^{(i)}(m)$ can also be economically interpreted as "congestion prices" at node k during the m-iteration. The dual updating scheme of the subgradient algorithm can then be viewed as a pricing scheme. When node k becomes increasingly congested, then $d_{\mu,k}^{(ij)} < 0$. From (15), we can see the price of node k will be increased. On the other hand, when node k becomes less congested, then $d_{\mu,k}^{(ij)} > 0$. Again, from (16), it can be sen that the price will be decreased.

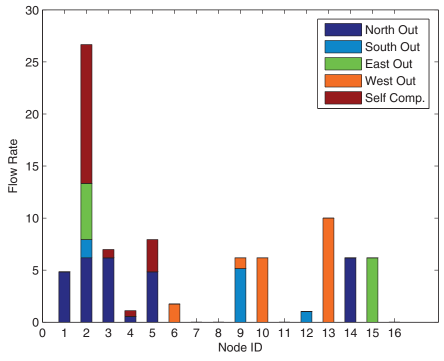


Figure 5: Outgoing links and self-loop flow rates at each node for edge e_{12}.

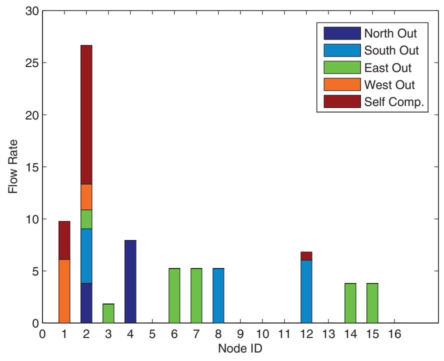


Figure 6: Outgoing links and self-loop flow rates at each node for edge e_{13}.

5. NUMERICAL RESULTS

In this section, we use Fig. 1 as an example to illustrate our proposed network flow model and distributed solution procedure. That is, we want to optimize the deployment of the MIMO-DAG computation framework in Fig. 1(a) onto the 16-node 2-D torus interconneced network in Fig. 1(b). The physical locations of both Θ_1 and Θ_2 are at node n_{16}. Here, we let the capacity of each link in Fig. 1(b) be 10 and let the per-node unit computation cost be 0.001. Our objective is to maximize the computation rate, i.e., letting $U(\lambda) = \lambda$. After optimization, the maximum computation rate is 15.95. Due to space limitation and the large number of optimal link flow rate variables for this example ($5 \times 16 \times 13 = 1040$ variables), we only plot in Figs. 5 and 6 the outgoing links and self-loops flow rates for edges e_{12} and e_{13} to illustrate part of the optimal solution. The "North Out," "South Out," "East Out," and "West Out" in Figs. 5 and 6 represent the outgoing links at each node in Fig. 1(b) along the specified directions, respectively. Recall from Fig. 1(a) that edges e_{12} and e_{13} are sink edges, which correspond to the final results of functions Θ_1 and Θ_2, respectively. Surprisingly, it can be seen that the computations of these sink edges are not deployed close to the physical output node n_{16}. The majority of the computations of e_{12} and e_{13} are done at node 2 (see the rates of self-loops at node 2 and node 1). This shows that the heuristic "proximity" deployment rule may not be optimal. From Figs. 5 and 6, it is also not difficult to see the optimal routing paths for edges e_{12} and

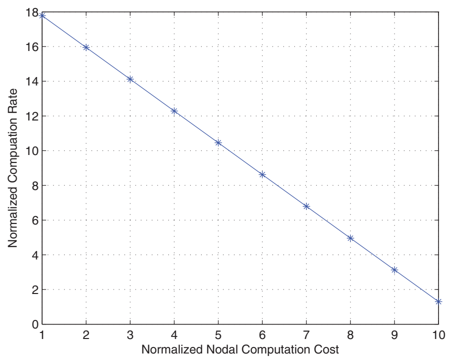


Figure 7: The change of maximum end-to-end computation rate with respect to the change of per-node unit computation cost.

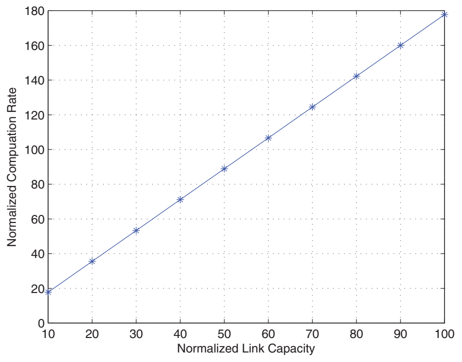


Figure 8: The change of maximum end-to-end computation rate with respect to the change of link capacity.

e_{13}. For example, the optimal routing paths for edge e_{13} are:

$$n_1 \xrightarrow{W} n_4 \xrightarrow{N} n_{16}, \qquad n_2 \xrightarrow{E} n_3 \xrightarrow{E} n_4 \xrightarrow{N} n_{16},$$

$$n_2 \xrightarrow{S} n_6 \xrightarrow{E} n_7 \xrightarrow{E} n_8 \xrightarrow{S} n_{12} \xrightarrow{S} n_{16}, \qquad n_2 \xrightarrow{W} n_1 \xrightarrow{W} n_4 \xrightarrow{N} n_{16},$$

$$n_2 \xrightarrow{N} n_{14} \xrightarrow{E} n_{15} \xrightarrow{E} n_{16}, \qquad n_{12} \xrightarrow{S} n_{16},$$

where the letter above each arrow denotes the routing direction at that hop. Next, we illustrate in Figs. 7 and 8 the changes of maximum end-to-end computation rate with respect to the changes of per-node unit computation cost and link capacity, respectively. In Fig. 7, we can see as expected that the end-to-end computation rate decreases as the unit computation cost at each node increases from 1 to 10 units (step size is 0.001). Likewise, we can see from Fig. 8 the end-to-end computation rate increases as the link capacity increases from 10 to 100.

6. CONCLUSION

In this paper, we investigated the design of distributed algorithms for cloud computing programming frameworks deployments. We formulated the computation rate utility maximization problem (CRUM) by developing a new network flow model with a generalized flow-conservation law. Based on this enabling framework, we developed a dual decomposition based distributed algorithm to solve Problem CRUM. We provided important networking interpretations and key implementation insights for our proposed algorithm and pointed out the connections and distinctions to distributed algorithms design in traditional data communications networks. Collectively, these results serve as the first building block of a new theoretical framework for the deployment of cloud computing programming frameworks.

7. REFERENCES

[1] A. S. Szalay, "Extreme data-intensive scientific computing," *IEEE Computing in Science and Engineering*, vol. 13, no. 6, pp. 34–41, Nov. 2011.

[2] "Data, data everywhere," Economist, 2010. [Online]. Available: http://www.economist.com

[3] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in *Proc. USENIX OSDI*, San Francisco, CA, Dec. 6-8, 2004, pp. 137–149.

[4] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: Distributed data-parallel programs from sequential building blocks," in *Proc. ACM SIGOPS/Eurosys*, Lisboa, Portugal, Mar. 21-23, 2007, pp. 59–72.

[5] K.-H. Lee, Y.-J. Lee, H. Choi, Y. D. Chung, and B. Moon, "Parallel data processing with MapReduce: A survey," *ACM SIGMOD Record*, vol. 40, no. 4, pp. 11–20, Dec. 2011.

[6] Y. Huai, R. Lee, S. Zhang, C. H. Xia, and X. Zhang, "DOT: A matrix model for analyzing, optimizing and deploying software for big data analytics in distributed systems," in *Proc. ACM SOCC*, Cascais, Portugal, Oct. 27-28, 2011.

[7] Hadoop. `http://hadoop.apache.org`.

[8] M. Chiang, S. H. Low, A. R. Calderbank, and J. C. Doyle, "Layering as optimization decomposition: A mathematical theory of network architecture," *Proc. IEEE*, vol. 95, no. 1, pp. 255–312, Jan. 2007.

[9] F. P. Kelly, A. K. Malullo, and D. K. H. Tan, "Rate control in communications networks: Shadow prices, proportional fairness and stability," *Journal of the Operational Research Society*, vol. 49, pp. 237–252, 1998.

[10] X. Lin, N. B. Shroff, and R. Srikant, "A tutorial on cross-layer optimization in wireless networks," *IEEE J. Sel. Areas Commun.*, vol. 24, no. 8, pp. 1452–1463, Aug. 2006.

[11] V. Shah, B. K. Dey, and D. Manjunath, "Network flows for functions," in *Proc. IEEE International Symposium on Information Theory (ISIT)*, St. Petersburg, Russia, Jul.31–Aug.5, 2011, pp. 234–238.

[12] M. S. Bazaraa, J. J. Jarvis, and H. D. Sherali, *Linear Programming and Network Flows*, 4th ed. New York: John Wiley & Sons Inc., 2010.

[13] H. Feng, Z. Liu, C. Xia, and L. Zhang, "Load shedding and distributed resource control of stream processing networks," *Performance Evaluation*, vol. 64, no. 9-12, pp. 1102–1120, Oct. 2007.

[14] H. Zhao, C. H. Xia, Z. Liu, and D. Towsley, "A unified modeling framework for distributed resource allocation of general fork and join processing networks," in *Proc. ACM Sigmetrics*, New York, NY, Jun. 14-18, 2010.

[15] Z. Liu, A. Tang, C. H. Xia, and L. Zhang, "A decentralized control mechanism for stream processing

networks," *Annals of Operations Research*, vol. 170, no. 1, pp. 161–182, Sep. 2009.

[16] F. T. Leighton, M. J. Newman, A. G. Ranade, and E. J. Schwabe, "Dynamic tree embeddings in butterflies and hypercubes," *SIAM Journal of Computing*, vol. 21, no. 4, pp. 639–654, Aug. 1992.

[17] O. Wohlmuth and F. Mayer-Lindenberg, "A method for the embedding of arbitrary trees into hypercubes," in *Proc. ACM Symposium on Applied Computing*, 1998, pp. 569–574.

[18] V. Heun and E. W. Mayr, "Efficient dynamic embeddings of arbitrary binary trees into hypercubes," *Journal of Algorithms*, vol. 43, pp. 51–84, 2002.

[19] A. W. Malik, A. Park, and R. M. Fujimoto, "Optimistic synchronization of parallel simulations in cloud computing environements," in *Proc. IEEE International Conference on Cloud Computing*, Bangalore, India, Sep. 21-25, 2009, pp. 49–56.

[20] M. S. Bazaraa, H. D. Sherali, and C. M. Shetty, *Nonlinear Programming: Theory and Algorithms*, 3rd ed. New York, NY: John Wiley & Sons Inc., 2006.

[21] L. Tassiulas and A. Ephremides, "Stability properties of constrained queuing systems and scheduling policies for maximum throughput in multihop radio networks," *IEEE Trans. Autom. Control*, vol. 37, no. 12, pp. 1936–1948, Dec. 1992.

APPENDIX

A. PROOF OF PROPOSITION 2

To derive the subproblem $\Phi_{\mathrm{FC}}(\boldsymbol{\mu}, \mathbf{w})$, we combine all terms in (10) related to λ, which leads to:

$$U(\lambda) - \lambda\Big[\sum_{i=1}^{|\mathcal{S}|}\Big(\sum_{j\in\Psi^i}\mu^{(ij)}_{\mathrm{Src}(i)} + \sum_{k=1}^{N}\rho_k\Big) - \sum_{i=E-K+1}^{E} w^{(i)}_{\mathrm{Dst}(i)}\Big].$$

Note that the above expression is exactly the objective function of Φ_{FC} in Proposition 2.

Next, we derive the objective function of $\Phi_R^{(l)}$, which is relatively more involved. First, it is not difficult to verify that by switching the summation order from node-based to link-based, we can rewrite the fouth term in (10) as:

$$\sum_{i=E-K-1}^{E}\sum_{k=1}^{N} w_k^{(i)}\left(\sum_{l\in\mathcal{O}(n_k)} x_l^{(e_i)} - \sum_{l\in\mathcal{I}(n_k)} x_l^{(e_i)} - y_{n_k}^{(e_i)}\right)$$
$$= \sum_{l=1}^{L}\sum_{i=E-K+1}^{E}\Big(w^{(i)}_{\mathrm{Tx}(l)} - w^{(i)}_{\mathrm{Rx}(l)}\Big)x_l^{(e_i)} - \sum_{k=1}^{N}\sum_{i=E-K+1}^{E} w_k^{(i)} y_{n_k}^{(e_i)}. \quad (22)$$

By the same token, we can immediately rewrite the partial second term (excluding the summation involving λ) in (10) as:

$$\sum_{i=1}^{E-K}\sum_{k=1}^{N}\rho_k\left(\sum_{l\in\mathcal{O}(n_k)} x_l^{(e_i)} - \sum_{l\in\mathcal{I}(n_k)} x_l^{(e_i)}\right) - \sum_{k=1}^{N}\sum_{i=|\mathcal{S}|+1}^{E-K}\rho_k y_{n_k}^{(e_i)}$$
$$= \sum_{l=1}^{L}\sum_{i=1}^{E-K}\Big(\rho_{\mathrm{Tx}(l)} - \rho_{\mathrm{Rx}(l)}\Big)x_l^{(e_i)} - \sum_{k=1}^{N}\sum_{i=|\mathcal{S}|+1}^{E-K}\rho_k y_{n_k}^{(e_i)}. \quad (23)$$

Now, for the more complex third term in (10), we have

$$\sum_{i=1}^{E}\sum_{j\in\Psi^i}\sum_{k=1}^{N}\mu_k^{(ij)}\left(\sum_{l\in\mathcal{O}(n_k)} x_l^{(e_i)} + y_{n_k}^{(e_j)} - \sum_{l\in\mathcal{I}(n_k)} x_l^{(e_i)} - y_{n_k}^{(e_i)}\right)$$
$$\overset{(a)}{=}\sum_{i=1}^{E}\sum_{k=1}^{N}\left(\left(\sum_{l\in\mathcal{O}(n_k)} x_l^{(e_i)} - \sum_{l\in\mathcal{I}(n_k)} x_l^{(e_i)} - y_{n_k}^{(e_i)}\right)\sum_{j\in\Psi^i}\mu_k^{(ij)} + \sum_{j\in\Psi^i}\mu_k^{(ij)} y_{n_k}^{(e_j)}\right)$$
$$= \sum_{i=1}^{E}\sum_{k=1}^{N}\left(\sum_{l\in\mathcal{O}(n_k)} x_l^{(e_i)} - \sum_{l\in\mathcal{I}(n_k)} x_l^{(e_i)}\right)\sum_{j\in\Psi^i}\mu_k^{(ij)} + \sum_{i=1}^{E}\sum_{k=1}^{N}\sum_{j\in\Psi^i}\mu_k^{(ij)}\Big(y_{n_k}^{(e_j)} - y_{n_k}^{(e_i)}\Big)$$
$$\overset{(b)}{=}\sum_{l=1}^{L}\sum_{i=1}^{E}\left(\sum_{j\in\Psi^i}\mu^{(ij)}_{\mathrm{Tx}(l)} - \sum_{j\in\Psi^i}\mu^{(ij)}_{\mathrm{Rx}(l)}\right)x_l^{(e_i)} + \sum_{k=1}^{N}\sum_{i=1}^{E}\sum_{j\in\Psi^i}\mu_k^{(ij)}\Big(x_l^{(e_j)} - y_{n_k}^{(e_i)}\Big), \quad (24)$$

where (a) holds because the summations of the $x_l^{(e_i)}$-variables do not involve index j and can be taken outside of the summation with respect to index j; and (b) follows from the same token as in (22) and (23) and the fact that switching the summation orders of the $x_l^{(e_j)}$-variables does not change their sum value.

Next, by adding (22), (23), (24) together and defining new $\tilde{w}$- and $\hat{w}$-variables as in (13) and (14), we arrive at a summation of the following two terms:

$$\sum_{l=1}^{L}\sum_{i=1}^{E}\left\{\Big(\tilde{w}^{(i)}_{\mathrm{Tx}(l)} - \tilde{w}^{(i)}_{\mathrm{Rx}(l)}\Big)x_l^{(e_i)}\right\}, \quad (25)$$

$$\sum_{k=1}^{N}\sum_{i=1}^{E}\left[\sum_{j\in\Psi^i}\mu_k^{(ij)}(y_{n_k}^{(e_j)} - y_{n_k}^{(e_i)}) - \hat{w}_k^{(i)} y_{n_k}^{(e_i)}\right], \quad (26)$$

which is exactly the objective function of $\Phi_R^{(l)}$ and $\Phi_C^{(k)}$. Note that the outer summation in (25) is with respect to link indices, each summand can be decomposed and computed at each link locally. Likewise, the outer summation in (26) is with respect to node indices, each summand can be decomposed and computed at each node locally.

Finally, noting that the constraints $\sum_{i=1}^{E} x_l^{(e_i)}$, $\forall l$, can also be separated in a link-wise fashion, we then have that the maximization of $L(\lambda, \mathbf{x}, \boldsymbol{\mu}, \mathbf{w})$ can be equivalently computed as the sum of a series of maximization subproblems at the source and each link, which are defined as in Proposition 2. This completes the proof.