Towards Minimal Tardiness of Data-intensive Applications in Heterogeneous Networks

Tong Li[*], Ke Xu[*], Meng Sheng[†], Haiyang Wang[‡], Kun Yang[§] and Yuchao Zhang[*]
[*]Tsinghua University, Beijing, China. Email: (litong12@mails, xuke@, zhangyc14@mails).tsinghua.edu.cn
[†]Corresponding author: Beijing Institute of Technology, Beijing, China. Email: shenmeng@bit.edu.cn
[‡]University of Minnesota at Duluth, Minnesota, United States. Email: haiyang@d.umn.edu
[§]University of Essex, Colchester, United Kingdom. Email: kunyang@essex.ac.uk

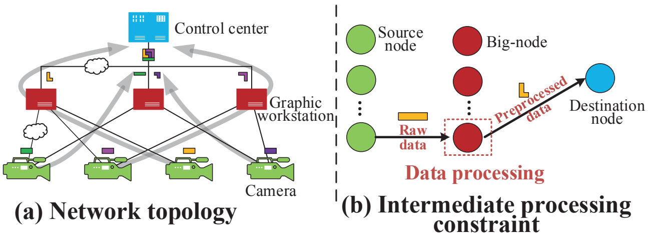
***Abstract*—The increasing data requirement of Internet applications has driven a dramatic surge in developing new programming paradigms and complex scheduling algorithms to handle data-intensive workloads. Due to the expanding volume and the variety of such flows, their raw data are often processed on intermediate processing nodes before being sent to servers. The intermediate processing constraints are however not yet considered in existing task and flow computing models.**

In this paper, we aim to minimize the total tardiness of all flows in the presence of intermediate processing constraints. We build a model to consider Tardiness-aware Flow Scheduling with Processing constraints (TFS-P), which is unfortunately NP-Hard. Hence, we propose a heuristic Routing and Scheduling duplex MATching (RSMAT) framework based on the classic Gale-Shapley Matching Theory. We find that the problem can be well-addressed by classic Deferred Acceptance (DA) algorithm, in which the match is stable but inefficient for the model. We therefore propose the Tardiness-aware Deferred Acceptance algorithm with Dynamical Quota (TDA-DQ). This algorithm is enhanced by overcoming the *inefficient stability* and smartly considering the *dynamical quota* in the system. The evaluation compares TDA-DQ to the lower bound obtained by a modified subgradient optimization algorithm. The result indicates that TDA-DQ can achieve near-optimal performance for data-intensive applications.

***Index Terms*—Heterogeneous Network, Data-intensive Application, Matching Theory**

I. Introduction

Based on the real-time analysis of large volumes of complex sensor data, real-time Big Data applications in heterogeneous networks have received considerable attention in recent years [1] [2]. Whereas it is cost-inefficient to indiscriminately send all raw data to the server, a solution towards effective processing of data-intensive workloads is enabling the networked nodes with processing capacity [2] [3]. However, capability anisotropy, stage treatment, and load balancing promote an increasing number of special data-intensive workloads, requiring being processed (such as size changing, format conversion, and noise handling) at intermediate nodes on its route to destination, which is called *intermediate processing constraint*. An intermediate processing node is usually a dedicated high-performance machine with sufficient resources (e.g., CPU cores, memory). For simplicity, we call it **big-node** in this paper. On the other hand, for the *flows*[1] of a soft real-time application, their expected completion time varies due to the interdependence of flows, i.e., deadline of each flow might be discrepant [4]. Meanwhile, the flow is still meaningful and contributes to the application even if it misses the deadline [5]. Thus, optimizing the application tardiness means minimizing the *total tardiness* of all distributed flows [6]. Figure 1(a) shows the example of a smart city, cameras distributed in each district aggregate to a monitoring network of intelligent transportation system (ITS) [3]. A typical application is that when capturing all the peccant vehicles (traffic violations) at some point, distributed cameras transmit their snapshots to the control center. As shown in Fig. 1(b), due to limited processing ability of the camera embedded system [7], image processing needs to be migrated to one of the capable big-nodes (e.g., graphic workstations).



(a) Network topology **(b) Intermediate processing constraint**

Fig. 1: An example to handle data-intensive workloads with intermediate processing constraints

Figure 2 elaborates that flow completion time is the sum of handling time and routing delay. Handling time is composed of the queueing time and the processing time for a flow to be processed on a big-node. While routing delay equals to the propagation time on the path from source to destination via big-node. Assume big-nodes are heterogeneous with different processing speed, then the processing time and routing delay depends on the selection of big-nodes, and the queueing time depends on the scheduling scheme on big-nodes. However, the interaction of routing and scheduling makes it challenging to simultaneously reduce the queueing time, processing time, and routing delay. For instance, as to a flow, being processed at a high-performance big-node gains smaller processing time. However, the path length might be long. Besides, achieving the minimal routing delay is to select the big-node on its shortest path. However, the queueing time might be extremely long on

[1] Different from the 5-tuple traffic flow, the flow here is composed of two sub-flows, i.e., sub-flow from source to intermediate node, and sub-flow from intermediate node to destination.

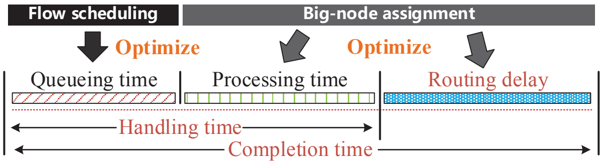


Fig. 2: Components of flow completion time

an overloaded big-node.

Under these circumstances, to minimize the total completion time, flow scheduling should take into account the collaborative routing strategies (see Section II for details). It is worth noticing that individual scheduling inevitably results in longer propagation delay, which means using more links to complete an application. Hence, individual scheduling may suffer from bandwidth waste and serious risk of link failure, which leads to congestion. Therefore, even in the case that the handling time is much longer than the routing delay, this joint optimization remains significant.

This paper is devoted to addressing the following question: **how to assign big-nodes to flows, and which flow should be preferentially processed, to minimize the total tardiness of all flows?** In addition to much work on individual routing [8]–[10] and individual scheduling [11]–[17], the joint optimization of routing and scheduling is attracting more attention currently [18]–[22]. Different from the prior solutions integrating routing and scheduling, we focus on the intermediate processing constraints, which are new and challenging in achieving the minimal tardiness of data-intensive applications in heterogenous networks.

Motivated by these observations, we formulate the problem to minimize the total tardiness of all flows in the presence of the intermediate processing constraints. Due to its NP-hardness, we apply the Gale-Shapley Matching Theory [23] in the cooperativity between scheduling and routing. By setting a quota for each big-node and heuristically generating the preference lists, we map our problem into the College Admission Problem (CAP) [23]. As a result, the framework of Routing and Scheduling duplex MATching (RSMAT) is designed. To realize our RSMAT framework, (a we start from the Gale-Shapley Deferred Acceptance (DA) algorithm, which has already been proved to result in stable and the Pareto efficient match [23]. However, this stability is inefficient for the RSMAT framework, called *inefficient stability*. Meanwhile, the performance of DA shows dependence on the quota of big-nodes, called *quota dependency*. (b By considering the expected handling time of flows and the dynamical quota in the system, we develop the Tardiness-aware Deferred Acceptance algorithm with Dynamical Quota (TDA-DQ). Our major contributions are summarized as follows:

- We model the problem as the joint optimization of routing with intermediate processing constraints and multi-processer scheduling with capability constraints (Section III). And a modified subgradient optimization algorithm is designed to obtain the lower bound (Section IV).

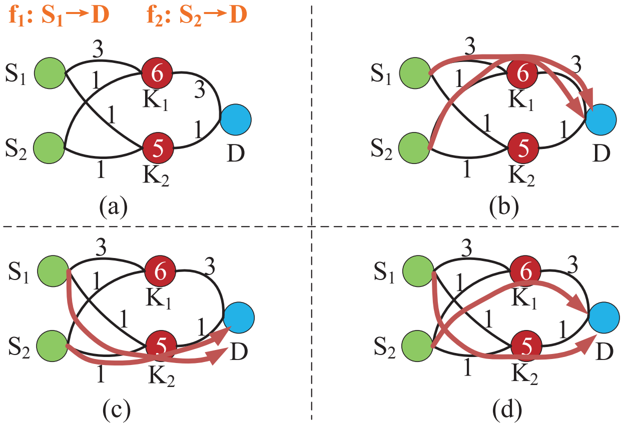


Fig. 3: Case study for motivation

- By applying the Matching Theory in the cooperativity between scheduling and routing, we map our problem into CAP and propose the RSMAT framework (Section V).
- We overcome the challenge of inefficient stability and quota dependency, and design the enhanced duplex matching algorithm TDA-DQ (Section VI).
- We conduct an evaluation to show that TDA-DQ can achieve near-optimal performance for data-intensive workloads, and gains better applicability in real-time scenarios (Section VII).

II. Motivating Cases

In this section, we use a case study to motivate the joint optimization of routing and scheduling.

As illustrated in Fig. 3(a), two flows of a soft real-time application (f_1 from S_1 to D with deadline $4t$, and f_2 from S_2 to D with deadline $6t$, t is a time period) are scheduled together with intermediate processing constraints. Resource requirements of f_1 and f_2 are 2 units and 4 units, respectively. Resource on big-nodes K_1 and K_2 is 6 units and 5 units, respectively. Propagation delay on link $\langle S_1, K_1\rangle$ and $\langle K_1, D\rangle$ are both $3t$, with $1t$ on other links. K_2 takes $2t$ and $4t$ to process f_1 and f_2, respectively. The processing speed of K_1 is twice of K_2. Here we assume that one flow assigned to a full-loaded node has to wait for idle state of the node. We define the total tardiness [6] of the application as $\sum_{m\in\mathcal{M}} T_m$, where $T_m = \max\{0, C_m - d_m\}$ is the tardiness of flow m, C is the flow completion time, and d is the deadline.

Scheduling-only. Individual scheduling aims to achieve the minimal handling time of flows. Since K_1 owns sufficient resource and its processing speed is faster than K_2, individual scheduling results in assigning K_1 to f_1 and f_2 simultaneously without queueing time, as shown in Fig. 3(b). The completion time of f_1 and f_2 are $7t$ and $6t$, respectively. Thus, the total tardiness is $3t$.

Routing-only. Tardiness-aware routing [9] aims to seek a big-node on the shortest path. As shown in Fig. 3(c), individual routing results in assigning K_2 to f_1 first with completion time $4t$. When f_1 is processed on K_2, f_2 has to wait because the residual resource of K_2 cannot meet its requirement. As a result, its queueing time is $2t$ and the completion time becomes $8t$. The total tardiness is $2t$.

Combination. However, as illustrated in Fig. 3(d), the global optimal strategy is to collaborate routing and scheduling by assigning K_2 to f_1 with completion time $4t$, while assigning K_1 to f_2 with completion time $6t$, during which the total tardiness is 0.

Case study reveals that it hardly achieves the minimal total tardiness of application within a single dimension. A joint optimization of routing and scheduling shows the potential to achieve the global optimal results. Motivated by these observations, this paper focuses on the data-intensive workloads with intermediate processing constraints, which are involved with both routing and scheduling. The former is to choose which big-node to process the flow (based on that selection of big-nodes affects the routing delay and processing time), and the latter is to decide which flow among a set of requests is to be preferentially processed at the big-nodes (based on that scheduling sequence affects the queueing time).

III. Problem Formulation

This section models the joint optimization for routing and scheduling with intermediate processing constraints. Table I summarizes part of the notations in our modeling.

TABLE I: Part of the Notations in Problem Formulation

Notation	Description
$\mathcal{M}$	Set of flows requiring intermediate processing;
$\mathcal{K}$	Set of big-nodes;
v_k	Processing speed of big-node k;
p	Processing time of a flow;
q_m	Amount of resource requirement of flow m;
d_m	Deadline of flow m;
C_m	Completion time of flow m;
ω_{ij}	Shortest path length from node i to node j;
l	Relay-path length of a flow;
Q_k	Amount of resource on big-node k;
T_m	Tardiness of flow m;
τ_m	Start time for flow m to be processed;
x^t_{mk}	1 if flow m is processed at big-node k at discrete time t.

We express the network in triples of vertices, edges, and big-nodes, i.e., $\mathcal{G}(\mathcal{N},\mathcal{E},\mathcal{K})$. $\mathcal{K}$ denotes the set of K big-nodes equipped with computing power. $\mathcal{M}$ denotes the set of M flows integrated to finish a soft real-time application. For a given flow $m \in \mathcal{M}$, we use p_m, q_m and d_m to denote the processing time, the amount of resource requirement, and the deadline, respectively. Particularly, p_{mk} denotes the processing time of flow m processed at big-node k ($k \in \mathcal{K}$), which is in inverse proportion to the big-node's processing speed v_k. Set C_m as the completion time of flow m. Our objective is to minimize the total tardiness of all flows, i.e., $\min \sum_{m\in\mathcal{M}} T_m$, where the tardiness of flow m is $T_m = \max\{0, C_m - d_m\}$. For convenience, T_m can also be denoted as $\{C_m - d_m\}^+$, where we define the expression $\{x\}^+ = \left\{ \begin{array}{ll} x, & x \geq 0 \\ 0, & \text{otherwise} \end{array} \right.$.

Let x^t_{mk} be the decision variable that is 1 if flow m is processed at big-node k at discrete time t. τ_m denotes the start time period of flow m, which can be denoted as

$$\tau_m = \min\{t | x^t_{mk} = 1\}. \tag{1}$$

We define the **relay-path** of a flow as the shortest path from source to destination, via a big-node. Use l_m to denote the relay-path length of flow m, and l_{mk} denotes the relay-path length of flow m that is assigned big-node k. Let ω_{ik} be the shortest path length between the source node i and the big-node k, and ω_{kj} be the shortest path length between the big-node k and the destination node j. Thus $l_{mk} = \omega_{ik} + \omega_{kj}$.

Assume propagation delay is proportional to path length. For convenience, we regard l as the routing delay. According to Fig. 2, we obtain $C_m = \tau_m + p_m + l_m$. By simultaneously considering the big-node assignment and flow scheduling, we present a model of Tardiness-aware Flow Scheduling with Processing constraints (TFS-P), expressed as follows:

$$\min \quad \sum_{m\in\mathcal{M}} T_m \tag{2}$$

$$\text{s.t.} \quad \sum_{k\in\mathcal{K}} x^t_{mk} = \left\{ \begin{array}{ll} 1, & t \in [\tau_m, \tau_m + p_m] \\ 0, & \text{otherwise} \end{array} \right. \tag{3}$$

$$\sum_{m\in\mathcal{M}} x^t_{mk} \cdot q_m \leq Q_k, \quad \forall k \in \mathcal{K} \tag{4}$$

$$\sum_{k\in\mathcal{K}} \sum_{t=\tau_m}^{\tau_m+p_m} x^t_{mk} = p_m, \quad \forall m \in \mathcal{M} \tag{5}$$

$$x^t_{mk} = 0 \text{ or } 1, \quad \forall m \in \mathcal{M}, k \in \mathcal{K} \tag{6}$$

where Q_k is the amount of resource on big-node k. Formula (3) refers to that no big-node (resource) is allocated when flows are inactive[2], and one flow is only assigned one big-node during its processing period. Formula (4) is the capability constraint, ensuring that the cumulative resource usage on big-node k at any given time cannot exceed its capacity. Formula (5) is to maintain the allocation state of a flow, i.e., once active, a flow is ensured to be processed on its specified big-node during its processing period.

According to Formulas (1), (2) and (6), TFS-P is a non-linear integer programming problem, and it essentially turns out to be a combinational optimization of the routing with intermediate processing constraints and multi-processer scheduling with capability constraints. Brucker et al. summarized the complexity results for scheduling problems [24]. Acting as the subproblem of TFS-P, the multi-processer scheduling problem for minimizing the total tardiness is still open. Since TFS-P owns higher complexity than multi-processer scheduling, we focus on the lower bound instead of the optimal value of TFS-P, which will be detailed in the next section.

IV. Lower Bound

This section applies the Lagrangian Relaxation Theory [25] to the combinational optimization to obtain the lower bound.

[2] When a flow is started and assigned a certain big-node, it is called active, otherwise inactive.

A. Lagrangian Relaxation

With respect to the constraints of TFS-P model, capacity constraint (Formula (4)) links every flow, which enhances the difficulty of the optimization. Applying the Lagrangian Relaxation Theory to handle the capacity constraint, and according to Formula (2), we obtain the Lagrangian Relaxation Problem (LRP) as follows:

LRP: $Z = \max_{\lambda} Z(\lambda)$

$$= \max_{\lambda} \min_{X} \{ \sum_{m \in \mathcal{M}} T_m + \sum_{k \in \mathcal{K}} \sum_{t=1}^{T^*} \lambda_{kt} \{ \sum_{m \in \mathcal{M}} x_{mk}^{t} q_m - Q_k \} \} \tag{7}$$

s.t. (3) , (5) , (6) , and $\lambda_{kt} \geq 0$.

where we use T^* to denote the maximum completion time across the flows, X is a matrix of x_{mk}^t, i.e., $X = (x_{mk}^t)_{M \times K \times T^*}$, loose coefficient λ is a matrix of λ_{kt}, i.e., $\lambda = (\lambda_{kt})_{K \times T^*}$. Define

$$h(m, X, K, T^*, \lambda) = T_m + \sum_{k \in \mathcal{K}} \sum_{t=1}^{T^*} \lambda_{kt} x_{mk}^t q_m \tag{8}$$

According to Equations (7) and (8), LRP turns into

$$Z = \max_{\lambda} \{ \sum_{m \in \mathcal{M}} \min_{X} h(m, X, K, T^*, \lambda) - \sum_{k \in \mathcal{K}} \sum_{t=1}^{T^*} \lambda_{kt} Q_k \}$$

s.t. (3) , (5) , (6) , and $\lambda_{kt} \geq 0$.

To obtain the lower bound of TFS-P, we divide LRP into two steps. Firstly, to solve the subproblem (SUBP) with the given λ_{kt} ($\lambda_{kt} \geq 0$). SUBP can be formulated as follows:

SUBP:

$$\min_{X} h(m, X, K, T^*, \lambda) = \min_{X} \{ T_m + \sum_{k \in \mathcal{K}} \sum_{t=1}^{T^*} \lambda_{kt} x_{mk}^t q_m \}$$

s.t. (3) , (5) , and (6).

Once the SUBP is solved, we have

$$Z(\lambda) = \sum_{m \in \mathcal{M}} \min h(m, X, K, T^*, \lambda) - \sum_{k \in \mathcal{K}} \sum_{t=1}^{T^*} \lambda_{kt} Q_k \tag{9}$$

Secondly, to find the maximum $Z(\lambda)$ in Formula (9) with constraint $\lambda \geq 0$, we regard this maximum $Z(\lambda)$ as the lower bound Z_{LB}.

B. Subgradient Optimization Algorithm

As mentioned above, we use two steps to solve LRP, of which the first step is to solve SUBP. According to the definition of x_{mk}^t, to solve the SUBP, we only need to estimate

$$\{t^* + l_{mk^*} + p_{mk^*} - d_m\}^+ + q_m \sum_{t=t^*}^{t^* + p_{mk^*}} \lambda_{k^* t^*}$$
$$\leq \{\tau_m + l_{mk} + p_{mk} - d_m\}^+ + q_m \sum_{t=\tau_m}^{\tau_m + p_{mk}} \lambda_{kt} \tag{10}$$

According to Formula (10), for each flow, t^* and k^* can be calculated with computation complexity $O(KT^*)$. Therefore, the optimal solution $X = (x_{mk}^t)_{M \times K \times T^*}$ can be easily obtained with regard to SUBP.

As to the second step, we introduce a subgradient vector $S(X) = \{s(k, t, X), k \in \mathcal{K}, 1 \leq t \leq T^*\}$, then the subgradient of LRP can be

$$s(k, t, X) = \{ \sum_{m \in \mathcal{M}} x_{mk}^t q_m - Q_k \}^+. \tag{11}$$

The subgradient optimization algorithm aims to find a λ to maximize $Z(\lambda)$ (the lower bound) by means of iteration. Here we give two stop conditions:

1) Stop when reaching the maximum iteration times;
2) Given an infinitesimally small number ε, stop in the i^{th} iteration if $\|S(X^i)\|^2 = \sum_{k \in \mathcal{K}} \sum_{t=1}^{T^*} s^2(k, t, X^i) < \varepsilon$, where $\varepsilon > 0$, and $\| \cdot \|^2$ denotes the L2 norm.

Algorithm 1 Subgradient Optimization Algorithm

```
1: $T^* = max(p_{mk}) + 1$, $i = 0$. Initialize $\lambda^0(K, T^*) = 0$;
2: Solve the SUBP. For each flow, calculate the optimal $x_{mk}^t$
   according to Formula (10). Then the subgradient $S(X)$ is
   obtained according to Formula (11). Set $\lambda^{i+1} = \{\lambda^i + \theta_i S(X^i)\}^+$.
   When it does not meet the stop conditions:
     1) If $\lambda^{i+1}(K, T^*) = 0$, $i = i + 1$, return to Step 2;
     2) If $\lambda^{i+1}(K, T^*) > 0$, $T^* = T^* + 1$, $\lambda^{i+1}(K, T^*) = 0$,
        $i = i + 1$, return to Step 2;
3: Calculate $Z_{LR}(\lambda^i)$ as the lower bound according to $X^i$ if
   it meets the stop conditions.
```

The subgradient optimization algorithm can be described as Algorithm 1. Since $0 < C_m < T^*$, a larger T^* broadens the solution space. Here we start from a small T^* and gradually increase it during the iterations. The coefficient θ_i in Step 2 is defined by [26]. There exist multiple expressions of θ_i, for example, $\theta_i = \theta_0 \rho^i$ $(0 < \rho < 1)$, in which θ_i decreases exponentially, leading to fewer iterations. However, in this paper we denote θ_i as in [25] $\theta_i = \frac{Z_{UP}(i) - Z_{LB}(i)}{\|S(X^i)\|^2} \pi_i$, where $0 \leq \pi_i \leq 2$ and $\pi_0 = 2$. Z_{UP} is an upper bound equals to the objective value of a certain feasible solution (e.g., a greedy algorithm). Z_{LB} is a lower bound of $Z_{LR}(\lambda^i)$ (the objective value of LRP with λ^i). Here we set $Z_{LB}(i) = Z_{LR}(\lambda^i)$. We technically remain π_i when $Z_{LR}(\lambda)$ increases, while reduce it by half when $Z_{LR}(\lambda)$ stays the same within a given number of iterations.

So far, we have obtained the lower bound of TFS-P, which will be used to estimate the algorithms proposed in the following sections.

V. Routing and Scheduling Matching Framework

TFS-P focuses on the joint optimization of flow routing and scheduling. The former aims to obtain the optimal bignode assignment, and the latter aims to achieve the optimal sequential scheduling. In this section, we argue that this joint optimization can be heuristically abstracted to a Routing

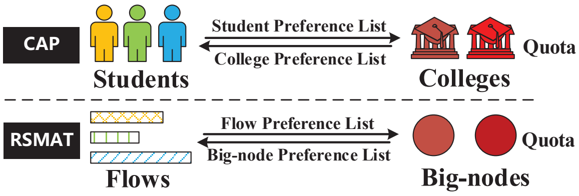


Fig. 4: Form CAP to RSMAT

and Scheduling MATching (RSMAT) framework, which is similar to the real-life College Admission Problem (CAP) [23]. Table II summarizes part of the notations in the algorithms.

TABLE II: Summary of Notations in Duplex Matching

Notation	Description
$\mathcal{A}$	Preference list for a flow to select big-nodes;
$\mathcal{B}$	Preference list for a big-node to select flows;
$\mathcal{Y}$	Prospective admission list of a big-node;
φ	Quota limit of flows for a big-node to serve;
$\triangle\varphi$	Integer that denotes the quota step size($1 \leq \triangle\varphi \leq \lfloor \frac{M}{K} \rfloor$);
η	Parameter of a big-node related to service capability;
θ_l	Evaluation score in inverse proportion to relay-path length;
θ_v	Evaluation score in proportion to big-node processing speed;
θ_d	Evaluation score in inverse proportion to flow deadline;
θ_q	Evaluation score in inverse proportion to resource requirement;
θ_e	Evaluation score in proportion to flow's expected handling time.

As shown in Fig. 4, regarding to the CAP, students are considered by a college which can admit a quota of only φ. According to the applicant qualifications, the college decides which one to admit. Since students may apply multiple colleges according to their various preference lists, it is not generally satisfactory for the college to offer admission to its φ best-qualified applicants. In this paper, we regard flows as students, and big-nodes as colleges. However, flows are applying big-nodes who can theoretically admit infinite flows. Without binding an appropriate quota to each big-node, our problem can never be mapped into CAP. On the other hand, both flows and big-nodes own diverse properties, and their preference lists are variable within a large range. Therefore, the mapping challenges are a) how to set an appropriate quota for each big-node, and b) how to define the preference lists reasonably and efficiently.

A. Mapping Challenges from CAP to RSMAT

Quota Setting. We define the service limit (quota) as an additive attribute of big-nodes. This service limit is reasonable since we seldom process all flows on the same big-node. In other words, it contributes to load balance in a way by avoiding overloading the most popular big-node.

The number of flows and big-nodes is denoted by M and K, respectively. We define a variable η_k ($k \in \mathcal{K}$) relevant with the service capability of big-node k. To ensure $\varphi \in [1, M]$, the quota of each big-node can be denoted as

$$\varphi_k = \lceil \eta_k \cdot \frac{M}{K} \rceil, (\eta_k \in (0, K]) \quad (12)$$

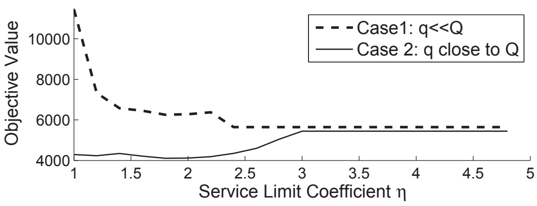


Fig. 5: Objective value vs. service limit coefficient η

Preference List Generation. The preference of flow m to select the alternative big-nodes results in preference list $\mathcal{A}_m$ (big-node set in $\mathcal{A}_m$ is included in $\mathcal{K}$). Also, every big-node owns a preference list $\mathcal{B}_k$ (flow set in $\mathcal{B}_k$ is included in $\mathcal{M}$).

- $\mathcal{A}_m$: For a flow to select a big-node, define standardized variable θ_l as the evaluation score in inverse proportion to relay-path length l_{mk}, and θ_v in proportion to the processing speed v_k. We regard the top of the preference list as the first choice, and the list is generated by a weighted sum of scores among multiple evaluation indicators. Thus, $\mathcal{A}_m$ is decided by $\theta_l + \theta_v$.
- $\mathcal{B}_k$: Similarly, for a big-node to select a flow, define standardized variable θ_d as the evaluation score in inverse proportion to flow deadline d_m, and θ_q in inverse proportion to the amount of resource requirement q_m. $\mathcal{B}_k$ depends on $\theta_q + \theta_d$.

B. Framework of RSMAT

Algorithm 2 Description of the RSMAT Framework

```
Input:
    $\mathcal{G}(\mathcal{N}, \mathcal{E}, \mathcal{K})$, $\mathcal{M}$, $\theta_l$, $\theta_v$, $\theta_d$, $\theta_q$, $\varphi_k$ $(k \in \mathcal{K})$;
Output:
    Big-node assignment and scheduling sequence;
1: for all $m$ $k$ that $m \in \mathcal{M}$, $k \in \mathcal{K}$ do
2:     $\mathcal{A}_m = GetRankingListA(\theta_l, \theta_v)$;
3:     $\mathcal{B}_k = GetRankingListB(\theta_d, \theta_q)$;
4: end for;
5: $\mathcal{Y}_k = DuplexMatchingAlgorithm(\mathcal{A}, \mathcal{B}, \varphi)$;
6: for all $k$ $y$ that $k \in \mathcal{K}$, $y \in \mathcal{Y}_k$ do
7:     Schedule flow $y$ according to preference list $\mathcal{B}_k$;
8: end for;
```

By overcoming the challenges of quota setting and preference list generation, our problem is reduced to the Routing and Scheduling duplex MATching (RSMAT) framework, in which Match Theory [23] is applied. Defining $\mathcal{Y}_k$ as the prospective admission list of big-node k, the RSMAT framework can be described as Algorithm 2, in which *Step 5* is a certain duplex matching algorithm. We will discuss it next.

VI. Duplex Matching Algorithms

A. Gale-Shapley Deferred Acceptance Algorithm

Since our RSMAT framework is based on CAP, *Step 5* of the RSMAT framework can further be implemented by the classic Gale-Shapley Deferred Acceptance (DA) algorithm, which has already been proved to result in stable and the Pareto efficient match [23]. DA is described as follows:

Iteration 1: Every flow applies to its first choice of big-node in preference list $\mathcal{A}$. Each applicant can only apply to one big-node that has never rejected it. And then each big-node may own a set of x applicants. According to the applicant ranking in list $\mathcal{B}$, every big-node puts top φ ($x > \varphi$) flows into its prospective admission list $\mathcal{Y}$, then rejects the other $x - \varphi$ applicants. If $x \leq \varphi$, then just put the x applicants into $\mathcal{Y}$.

Iteration i: All flows eliminated in the last iteration continue to apply to their i^{th} choice in the preference list $\mathcal{A}$, and then each big-node takes both the new applicants and the ones already in $\mathcal{Y}$ into account, assuming the total number of applicants is x. According to list $\mathcal{B}$, every big-node puts top x ($x > \varphi$) flows into $\mathcal{Y}$ and rejects the other $x - \varphi$ applicants. If $x \leq \varphi$, then put the x applicants into $\mathcal{Y}$.

Repeat *Iteration* i until every flow is either in a prospective admission list of a certain big-node, or rejected by all big-nodes in its preference list. It is worth mentioning that here the preference list $\mathcal{A}$ and $\mathcal{B}$ are both constant during each iteration.

B. Applicability Consideration of DA

Inefficient Stability. Gale and Shapley have demonstrated the stability and the Pareto efficiency of DA with regard to both colleges and students. However, this Pareto efficiency is inapplicable for the RSMAT framework, in which the optimization objective is minimizing the total tardiness of all flows. Tardiness is tightly coupled with the handling time on the big-nodes. Since the handling time of a flow is composed of the queueing and processing time, considering the processing speed of a big-node can only contribute to the processing time. Without taking the queueing time into consideration, DA can hardly achieve approximately minimal tardiness. For example in Fig. 3(a), assume both f_1 and f_2 prefer K_2, meanwhile, K_2 prefers f_1. In this case, DA assigns K_2 to both f_1 and f_2. However, f_2 has to be suspended when f_1 is being processed, which results in the tardiness of f_2. Therefore, although the Pareto efficient, the stable match of DA is inefficient with regard to the RSMAT framework, which is called *inefficient stability*.

Quota Dependency. The quota of big-nodes in DA is obtained by Formula (12). Here we argue that the match in DA has severe dependence on this quota. For instance, Figure 5 shows the objective value varies with the service limit coefficient η_k ($k \in \mathcal{K}$). Two random-generated networks (cases) with $K = 10$ big-nodes and $M = 50$ flows are scheduled using DA. For simplicity, we set $\eta_k = \eta$ ($k = 1, 2, ..., 10$). In the case that big-nodes equipped with sufficient resource, i.e., one big-node can serve several flows at a time ($q_m \ll Q_k$). The objective value decreases with the increase of η. However, the objective value increases with η in the case that one big-node can only serve one or two flows at a time (q_m close to Q_k). Thus, the performance of DA depends on the quota setting, called *quota dependency*.

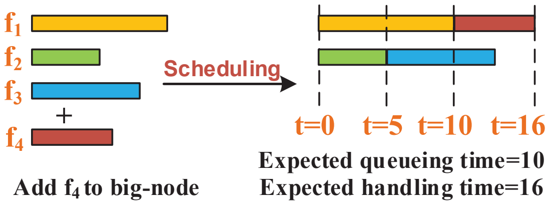


Fig. 6: Expected handling time

C. Tardiness-aware Deferred Acceptance with Dynamical Quota

To simultaneously mitigate inefficient stability and quota dependency, we finally propose the Tardiness-aware Deferred Acceptance algorithm with Dynamical Quota (TDA-DQ).

In terms of inefficient stability, the preference list $\mathcal{A}$ is constant during all iterations, which ensures the matching stability of DA. Here we break the stability by taking into account the flow's expected handling time on the big-node. Handling time depends on the scheduling sequence according to preference list $\mathcal{B}$. To calculate the expected handling time for a flow, we just temporarily add it into the big-node's prospective admission list $\mathcal{Y}$, and schedule it with the other flows on the big-node. Take Fig. 6 for example, f_1, f_2, and f_3 are flows already in $\mathcal{Y}$ before f_4 is added. $Q = 18, q_1 = 10, q_2 = 5, q_3 = 8, q_4 = 6, p_m = q_m (m = 1, 2, 3, 4)$. Assuming the preference in $\mathcal{B}$ is $f_1 > f_2 > f_3 > f_4$, and scheduling with capacity constraints, the expected handling time of f_4 is 16.

Since the flow's expected handling time depends on the interaction of all flows in the queue, $\mathcal{A}$ changes dynamically during each iteration. To calculate $\mathcal{A}$ in TDA-DQ, we define a standardized variable θ_e as the evaluation score in inverse proportion to the flow's expected handling time at the big-node. Considering the tradeoff between the relay-path length and the flow's expected handling time, the preference $\mathcal{A}$ for a flow to select a big-node in TDA-DQ can be obtained based on $\theta_l + \theta_e$.

On the other hand, to mitigate the challenge of quota dependency, we propose a technical solution that gives a dynamical quota to each big-node, i.e., gradually increase the quota of the preferred big-node in each iteration. Here we call it the preferred big-node whose applicant amount x meets $x > \varphi$.

TDA-DQ is detailed in Algorithm 3, of which the iterations are as outlined below.

Iteration i: Every flow that is not assigned a big-node applies to its first choice of big-node in the preference list $\mathcal{A}$. And then each big-node owns a set of x applicants. According

Algorithm 3 Description of TDA-DQ

```
Input:
    $\mathcal{G}(\mathcal{N},\mathcal{E},\mathcal{K})$, $\mathcal{M}$, $\mathcal{A}$, $\mathcal{B}$, $\varphi_k(k\in\mathcal{K})$;
Output:
    Big-node assignment $\mathcal{Y}_k$;
$i \leftarrow 0$, $\mathcal{Y}_k \leftarrow \emptyset(k\in\mathcal{K})$;
while ($\bigcup_{k\in\mathcal{K}}\mathcal{Y}_k \neq \mathcal{M}$) do
    for all $m$ $k$ that $m\in(\mathcal{M}\setminus\cup_{k\in\mathcal{K}}\mathcal{Y}_k)$, $k\in\mathcal{K}$ do
        Define $H_{mk}$ as the the flow's expected handling time at the big-node;
        $H_{mk} = GetExpectedHandleTime(\mathcal{Y}_k)$;
        $\mathcal{A}_m = GetRankingListA(\theta_l, \theta_e(H_{mk}))$;
        $\mathcal{Y}_k = \mathcal{Y}_k \bigcup\{m|GetTopItem(\mathcal{A}_m)=k\}$;
    end for;
    for all $k$ that $k\in\mathcal{K}$ do
        Sort $\mathcal{Y}_k$ according to flow ranking in $\mathcal{B}_m$;
        if $GetElementCount(\mathcal{Y}_k) > \varphi_k$ then
            Define a temporary set $\mathcal{H}$ as the set of the bottom ($GetElementCount(\mathcal{Y}_k)-\varphi_k$) flow(s) in $\mathcal{Y}_k$;
            $\mathcal{Y}_k = \mathcal{Y}_k \setminus \mathcal{H}$;
            $\varphi_k = \varphi_k + \triangle\varphi$;
        end if;
    end for;
end while;
return $\mathcal{Y}_k$;
```

to the preference list $\mathcal{B}$, every big-node puts top φ ($x > \varphi$) flows into its prospective admission list $\mathcal{Y}$ before rejecting the other $x-\varphi$ applicants. Then add $\triangle\varphi$ ($1 \leq \triangle\varphi \leq \lfloor\frac{M}{K}\rfloor$) to the φ of these preferred big-nodes. While if $x \leq \varphi$, put the x applicants into $\mathcal{Y}$.

Repeat *Iteration* i until every flow is in a prospective admission list of a certain big-node. Note that no matter whether a big-node has rejected the flow ever before, the flow can always apply to the big-node in a new iteration.

TDA-DQ tends to result in unstable matches, as we do not care whether flows have ever been rejected by big-nodes. However, in the problem of this paper, this algorithm can effectively obtain the approximately minimal tardiness, which is estimated in Section VII.

VII. Performance Evaluation

In this section, we conduct the Matlab-based implementation, in which different network topologies were randomly generated and fed to these programs to estimate the duplex matching algorithms for the RSMAT framework. Before presenting the results, we summarize the algorithms we compared in our simulations.

A. Algorithm Summary

- **Greedy**: The big-node achieving the smallest $l_{mk}+\frac{p_{mk}}{Q_k}$ are assigned to flows, and then the flows are scheduled using the earliest due date (EDD) priority rule [27] .
- **DA and TDA-DQ**: Algorithms designed for the RSMAT framework in Section V.
- **LSPF**: The routing-only strategy, in which the big-node with the shortest relay-path is assigned to flows.
- **TFS**: The scheduling-only strategy, which is a modified TDA-DQ algorithm that only considers individual flow scheduling, without taking into account the θ_l during updating preference list $\mathcal{A}$.

B. Numerical Evaluation

We consider a scenario of the ITS in a smart city. A random network with 500 nodes acting as the cameras, and then randomly choose 2% of nodes acting as the big-nodes (workstations distributed randomly across different districts). We consider an application with 500 flows from cameras to the control center, i.e., $M=500, K=10$. According to the prior works [4] [15], with random uniform distribution, we set $l_{mk}\in[40,400], q_m\in[200,400], d_m\in[1000,3000], v_k\in[1,K]$. $p_{mk}=\frac{q_m}{v_k}, Q_k\in[\frac{1}{2}\max(q_m), 2\max(q_m)]$ (all time unit is ms). For DA, the quota of big-node $\varphi_k=\lceil\eta_k\frac{M}{K}\rceil$, ($\eta_k=1$), and for TDA-DQ, we set $\triangle\varphi=1$. Finally, we generate 100 applications to evaluate the proposed algorithms.

Total Tardiness. As illustrated in Fig. 7, by applying TDA-DQ, the total tardiness in 60% cases is 0. This reveals that 60% applications are finished without tardiness. We can see that due to inefficient stability and quota dependency, DA performs even worse than the routing-only or greedy strategy.

To quantitatively analyze the algorithm optimality, for each application I we introduce the optimality gap

$$Gap(I)=\frac{Z(I)-Z_{LB}(I)}{Z(I)} \tag{13}$$

where $Z(I)$ is the objective value in case I, and $Z_{LB}(I)$ is the lower bound. Note that the lower bound is 0 if an application can be completed without tardiness. Thus we set $Gap(I)=\begin{cases}0, & \text{if } Z(I)=0\\ 1, & \text{if } Z(I)\neq 0, Z_{LB}(I)=0\end{cases}$.

Figure 8 further shows the gap on the total tardiness between the proposed algorithms and the lower bound. More than 90% of the gaps in TDA-DQ are less than 0.4, however, only 5% of the gaps in the other algorithms are less than 0.4. This indicates TDA-DQ outperforms any other proposed algorithms in optimality and approximates the lower bound well.

Completion Time. Figures 9 shows that TDA-DQ gains the min-max completion times, of which 94% are less than 3300 ms. Assume that the due date of the soft real-time applications is 5000 ms, Fig. 9 illustrates that more than 80% applications can be completed within 5000 ms when applying TFS and Greedy, which is acceptable in terms of the maximum completion time. However, systems applying LSPF and DA have to suffer from high application tardiness in this case. Similar insights involved with the average completion time are also illustrated in Fig. 10.

Routing Delay. Figures 11 shows the average routing delay. The routing-only strategy LSPF gains the lowest propagation delay, as it assigns the big-nodes with the shortest relay-path to flows. However, the scheduling-only strategy TFS results in the highest propagation delay. Assume that all links have the same

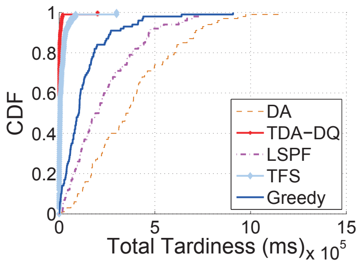


Fig. 7: Total tardiness

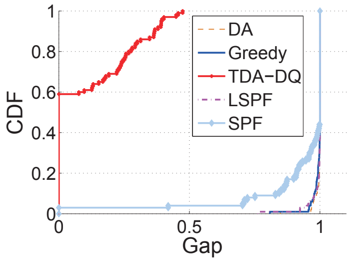


Fig. 8: Optimality gap

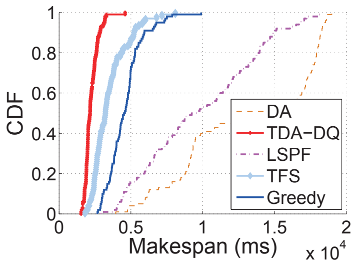


Fig. 9: Maximum completion time

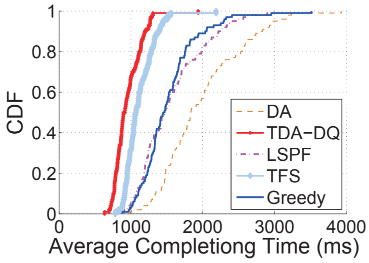


Fig. 10: Average completion time

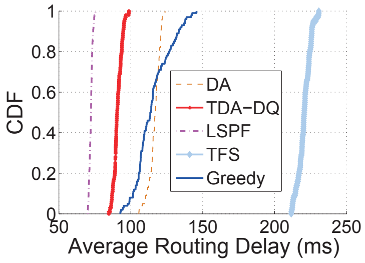


Fig. 11: Average routing delay

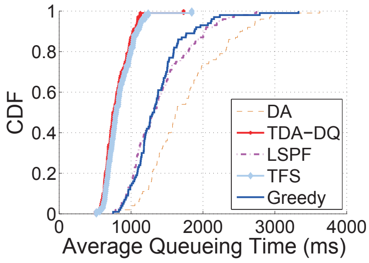


Fig. 12: Average queueing time

propagation delay, we can infer that higher propagation delay means using more links to complete an application. Hence, systems applying TFS may suffer from bandwidth waste and higher risk of link failure. This further enhances the necessity of integrating routing and scheduling even in the cases that routing delay is much less than handling time.

Queueing Time. The handling time of a flow is composed of the processing time and the queueing time. Figures 12 shows the average queueing time of different algorithms in the 100 applications. We find that TDA-DQ and TFS result in the relatively less queueing delay.

Applicability Evaluation. Tightly involved with the completion time of flows, minimum total tardiness reflects maximum throughput rate in most cases. Throughput rate here equals to the proportion of flows whose completion time does not exceed its deadline. Besides, since calculating the expected handling time is time-consuming in each iteration of algorithms, we use the iteration times to denote the algorithm overhead. Therefore, the algorithm applicability is defined as the tradeoff between overhead and throughput, i.e., $O + \xi \cdot \Theta$, where ξ ($\xi \geq 0$) is a coefficient relevant to throughput tolerance, O and Θ denote the normalized average overhead and the normalized average throughput rate, respectively.

Before making decisions, operators should estimate their application throughput tolerance ξ first. Figure 13 shows the applicability of the proposed algorithms, of which axes denote the zero-mean nomalization value. The green dashed reference line is drawn for the case that overhead and throughput get the same weight, i.e., $\xi = 1$, while the red one is drawn for the case that $\xi = 2$. In terms of applicability, points below the line are always better than those above the line. As a result, we find that Greedy are superior to TDA-DQ when $\xi = 1$. However, TDA-DQ outperforms all the others when $\xi = 2$. More generally, for operators the optimal choice is TDA-DQ when $\xi > 1.72$, while just Greedy when $0 \leq \xi \leq 1.72$. Whereas most flows in real-time systems are tardiness-sensitive, we usually have $\xi \gg 2$ in fact [5]. This reveals that TDA-DQ is more applicative in real-time scenarios.

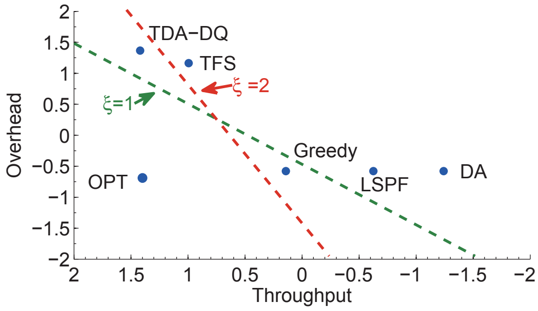


Fig. 13: Normalized overhead vs. normalized throughput

VIII. Related Work

The related work is expounded in three aspects including individual routing, individual scheduling, and the integration of them.

Routing. Traditional networking methods usually fall in the general category of Multi-Constrained Path (MCP) allocation problems [8]–[10], devoted to network resource such as bandwidth. However, these MCP solutions seldom meet the intermediate processing constraints, which necessitates the coupling of communication and computation, as not only the bandwidth is the constraint that should be considered, but also the computing resource.

Scheduling. Our envisioned research is also inspired from existing packet-level scheduling [11] [12], flow-level scheduling [13] [14], application-level scheduling [15], and job packing [16] [17] algorithms that aim to achieve the best performance or target fairness. However, these do not work well in scenarios characterized by real-time and intermediate processing constraints, as they do not take into account the collaborative routing strategies.

Integration of Routing and Scheduling. Zhao et al. [18] proposed RAPIER, which for the first time demonstrates that routing and scheduling must be jointly considered to optimize the average Coflow Completion Time (CCT) in Data Center Networks (DCNs). Works in [19]–[21] focused on the power-down model for network devices and proposed the energy-efficient flow scheduling and routing strategies. Han et al. [22] proposed a RUSH framework dealing with both the batch-arrival and the sequential-arrival of network flow demands in the hybrid DCNs. Although the joint optimization of routing and scheduling is attracting more attention currently, previous work seldom focuses on the intermediate processing constraints in heterogenous networks, thus their solutions are inapplicable to the problem in this paper.

IX. Conclusion

This paper is devoted to the joint optimization of flow routing and scheduling for scenarios with intermediate processing constraints, minimizing total tardiness of all flows. To the best of our knowledge, we for the first time apply the Gale-Shapley Matching Theory in the cooperativity between routing and scheduling. As a result, the duplex matching algorithm is proposed for our RSMAT framework, i.e., DA, in which the match is stable with the Pareto efficiency, but inefficient for RSMAT. To avoid inefficient stability and quota dependency, we develop TDA-DQ. Evaluation shows TDA-DQ achieves the near-optimal performance in handling data-intensive workloads. Applicability analysis reveals that TDA-DQ gains better applicability than the others in real-time cases, and different algorithms can adapt to different scenarios.

Future work includes the extension to environments with multi-resource management [28] and strategies on intermediate processing node placement [29].

Acknowledgment

This work was supported by National Natural Foundation of China (61472212), National Science and Technology Major Project of China (2015ZX03003004), National High Technology Research and Development Program of China (863 Program) (2013AA013302, 2015AA015601), EU Marie Curie Actions CROWN (FP7-PEOPLE-2013-IRSES-610524), and Beijing Natural Science Foundation (4164098).

References

[1] B. Yu and J. Pan, “Location-aware associated data placement for geo-distributed data-intensive applications,” in *IEEE INFOCOM*, 2015.

[2] C. P. Chen and C.-Y. Zhang, “Data-intensive applications, challenges, techniques and technologies: A survey on big data,” *Information Sciences*, vol. 275, pp. 314–347, 2014.

[3] R. Kitchin, “The real-time city? big data and smart urbanism,” *GeoJournal*, vol. 79, no. 1, pp. 1–14, 2013.

[4] K. Gardner, M. Harchol-Balter, and S. Borst, “Optimal scheduling for jobs with progressive deadlines,” in *IEEE INFOCOM*, 2015.

[5] R. Liu, A. F. Mills, and J. H. Anderson, “Independence thresholds: Balancing tractability and practicality in soft real-time stochastic analysis,” in *IEEE RTSS*, 2014, pp. 314–323.

[6] M. A. Sharaf, P. K. Chrysanthis, A. Labrinidis, and C. Amza, “Optimizing i/o-intensive transactions in highly interactive applications,” in *ACM SIGMOD*, 2009, pp. 785–798.

[7] I. F. Akyildiz and X. Wang, “A survey on wireless mesh networks,” *IEEE Communications Magazine*, vol. 43, no. 9, pp. S23–S30, 2005.

[8] S. Chen, M. Song, and S. Sahni, “Two techniques for fast computation of constrained shortest paths,” *IEEE/ACM TON*, vol. 16, no. 1, pp. 105–115, 2008.

[9] T. Korkmaz and M. Krunz, “Multi-constrained optimal path selection,” in *IEEE INFOCOM*, 2001, pp. 834–843.

[10] Y.-H. Wei, Q. Leng, S. Han, A. K. Mok, W. Zhang, and M. Tomizuka, “Rt-wifi: Real-time high-speed communication protocol for wireless cyber-physical control applications,” in *IEEE RTSS*, 2013, pp. 140–149.

[11] Z. Mao, C. E. Koksal, and N. B. Shroff, “Online packet scheduling with hard deadlines in multihop communication networks,” in *IEEE INFOCOM*, 2013, pp. 2463–2471.

[12] B. Vamanan, J. Hasan, and T. Vijaykumar, “Deadline-aware datacenter tcp (d2tcp),” in *ACM SIGCOMM*, 2012, pp. 115–126.

[13] F. Ciucu, F. Poloczek, and J. Schmitt, “Sharp per-flow delay bounds for bursty arrivals: The case of fifo, sp, and edf scheduling,” in *IEEE INFOCOM*, 2014, pp. 1896–1904.

[14] C. Y. Hong, M. Caesar, and P. B. Godfrey, “Finishing flows quickly with preemptive scheduling,” in *ACM SIGCOMM*, 2012, pp. 127–138.

[15] H. Wu, X. Lin, X. Liu, and Y. Zhang, “Application-level scheduling with deadline constraints,” in *IEEE INFOCOM*, 2014, pp. 2436–2444.

[16] S. Albagli-Kim, H. Shachnai, and T. Tamir, “Scheduling jobs with dwindling resource requirements in clouds,” in *IEEE INFOCOM*, 2014, pp. 601–609.

[17] Y. Zhu, Y. Jiang, W. Wu, L. Ding, A. Teredesai, D. Li, and W. Lee, “Minimizing makespan and total completion time in mapreduce-like systems,” in *IEEE INFOCOM*, 2014, pp. 2166–2174.

[18] Y. Zhao, K. Chen, W. Bai, C. Tian, Y. Geng, Y. Zhang, D. Li, and S. Wang, “Rapier: Integrating routing and scheduling for coflow-aware data center networks,” in *IEEE INFOCOM*, 2015.

[19] L. Wang, F. Zhang, K. Zheng, A. V. Vasilakos, S. Ren, and Z. Liu, “Energy-efficient flow scheduling and routing with hard deadlines in data center networks,” in *IEEE ICDCS*, 2014, pp. 248–257.

[20] M. Xu, Y. Shang, D. Li, and X. Wang, “Greening data center networks with throughput-guaranteed power-aware routing,” *Computer Networks*, vol. 57, no. 15, pp. 2880–2899, 2013.

[21] L. Wang, F. Zhang, A. V. Vasilakos, C. Hou, and Z. Liu, “Joint virtual machine assignment and traffic engineering for green data center networks,” in *ACM SIGMETRICS*, 2014, pp. 107–112.

[22] K. Han, Z. Hu, J. Luo, and L. Xiang, “Rush: Routing and scheduling for hybrid data center networks,” in *IEEE INFOCOM*, 2015.

[23] D. Gale and L. S. Shapley, “College admissions and the stability of marriage,” *American Mathematical Monthly*, pp. 9–15, 1962.

[24] P. Brucker and S. Knust, “Complexity results for scheduling problems,” http://www2.informatik.uni-osnabrueck.de/knust/class/, 2009, [Online; accessed 1-July-2015].

[25] J. Q. Xing W, Zhang J, “Capacitated single flexible manufacturing cell with setups: model, complexity and lagrangian relaxation,” *Operations Research and Its Applications*, pp. 162–170, 1995.

[26] M. L. Fisher, “An applications oriented guide to lagrangian relaxation,” *Interfaces*, vol. 15, no. 2, pp. 10–21, 1985.

[27] A. Dogramaci and J. Surkis, “Evaluation of a heuristic for scheduling independent jobs on parallel identical processors,” *Management Science*, vol. 25, no. 12, pp. 1208–1216, 1979.

[28] W. Wang, B. Liang, and B. Li, “Low complexity multi-resource fair queueing with bounded delay,” in *IEEE INFOCOM*, 2014, pp. 1914–1922.

[29] Y. Yuan, D. Wang, and J. Liu, “Joint scheduling of mapreduce jobs with servers: Performance bounds and experiments,” in *IEEE INFOCOM*, 2014, pp. 2175–2183.