Minimizing Tardiness for Data-Intensive Applications in Heterogeneous Systems: A Matching Theory Perspective

Ke Xu, *Senior Member, IEEE*, Liang Lv, Tong Li, Meng Shen, *Member, IEEE*, Haiyang Wang, and Kun Yang, *Senior Member, IEEE*

Abstract—The increasing data requirements of Internet applications have 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 (IPNs) before being sent to servers. However, the intermediate processing constraint is rarely considered in existing flow computing models. This paper aims to minimize the tardiness of data-intensive applications in the presence of intermediate processing constraint. Motivating cases show that the tardiness is affected by both IPN locations and flow dispatching strategies. Based on the observation that dispatching flows to IPNs is essentially building a matching between flows and IPNs, a novel solution is proposed based on matching theory. In the deployment phase, a tardiness-aware deferred acceptance algorithm is developed to optimize IPN locations. In the operation phase, the Power-of-D paradigm and matching theory are combined together to dispatch flows efficiently. Evaluation results show that our solution effectively minimizes the total tardiness of data-intensive applications in heterogeneous systems.

Index Terms—Heterogeneous system, data-intensive application, matching theory, power-of-D

1 Introduction

Real-time big data applications in heterogeneous systems have received considerable attention in recent years [1], [2]. In these applications, indiscriminately sending all raw data to the server puts pressure on the system infrastructure as well as the server. Recent researches preprocess the raw data, such as size changing, format conversion, and noise handling, on intermediate nodes to alleviate this problem [3], [4]. Intermediate Processing Nodes (IPNs) are usually high-performance machines with relative sufficient resources (e.g., CPU, memory). Besides, the practical needs of hierarchical processing and load balancing in real-world systems also promote processing data-intensive workloads on the intermediate nodes [5], [6], [7], which is called *intermediate processing constraint*.

As elaborated in Fig. 1, the completion time of a *flow*[1] from soft real-time data-intensive applications is the sum of handling time and routing delay. Handling time is composed of the queueing delay (i.e., the time a flow waits in the *queue on IPN* until it can be processed[2]) and the processing time for a flow to be processed on an IPN. Routing delay equals to the propagation time on the path from source to destination via an IPN. The expected completion time and deadlines of the flows might be different [8] and it is difficult to guarantee all deadlines are met in such distributed environments [9]. To be consistent with the previous researches [10], [11], [12], we focus on minimizing the *total tardiness* of all flows.

In the real-world distributed systems, there may be multiple generations or brands of hardware so that the IPNs may have different processing capacity, memory space and I/O bandwidth. The configurational heterogeneity makes it non-trivial to minimize the total tardiness in the presence of the intermediate processing constraint. In such heterogeneous systems, the processing time and routing delay of a flow depends on the routing strategies (i.e., selection of

- K. Xu is with the Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China, and also with the Beijing National Research Center for Information Science and Technology (BNRist), Beijing 100084, China. E-mail: xuke@tsinghua.edu.cn.
- L. Lv is with the Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China. E-mail: lvl16@mails.tsinghua.edu.cn.
- T. Li is with the Protocol Research Lab, Huawei, Shenzhen, China. E-mail: litong12@tsinghua.org.cn.
- M. Shen is with the School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081, China. E-mail: shenmeng@bit.edu.cn.
- H. Wang is with the Department of Computer Science and Engineering, University of Minnesota at Duluth, Duluth, MN 55812. E-mail: haiyang@d.umn.edu.
- K. Yang is with the School of Computer Science and Electronic Engineering, University of Essex, Colchester CO4 3SQ, United Kingdom. E-mail: kunyang@essex.ac.uk.

Manuscript received 25 May 2018; revised 26 June 2019; accepted 17 July 2019. Date of publication 25 July 2019; date of current version 18 Dec. 2019.
(Corresponding author: Tong Li.)
Recommended for acceptance by H. Childs.
Digital Object Identifier no. 10.1109/TPDS.2019.2930992

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

[2] In this paper, the time a flow waits for forwarding on the *network switches* is included in the *routing delay* instead of the queueing delay.

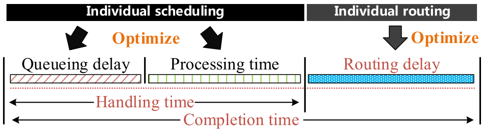


Fig. 1. Components of flow completion time.

IPNs), and the queueing delay depends on the scheduling strategy of IPNs (i.e., processing sequence of the flows).

However, routing strategies usually focus on routing delay and do not care about the handling time of flows on IPNs, while scheduling strategies are just the opposite. The interaction of routing and scheduling strategies makes it challenging to simultaneously reduce the above elements. For instance, as to a flow, being processed at a high-performance IPN gains shorter processing time, but the relay path might be long. On the other hand, selecting the IPN on the shortest path from source to destination may induce long queueing delay if the IPN is overloaded. Therefore, under these circumstances, to minimize the total tardiness, it is essential to take into account the collaborative strategies of routing and scheduling (see Section 2).

Much prior work has focused on individual routing[13], [14], [15], [16], [17], [18], [19], [20], [21] and individual scheduling [22], [23], [24], [25], [26], [27], [28], [29], [30]. Recent years, more researches pay attention to the interaction between routing and scheduling strategies [31], [32], [33], [34], [35], [36], [37], [38]. However, the prior solutions seldom consider the intermediate processing constraint, which is new and challenging to minimize the total tardiness in heterogeneous systems.

It should be mentioned that the intermediate processing constraint is becoming popular nowadays, especially with the emerging smart cities. Three-tier architecture is widely used by both academia [39], [40] and industrial [41], [42], [43] smart city solutions, where relay nodes process and transmit data from the cameras and sensors to the command center. In such scenarios, the relay nodes act as IPNs, while the cameras and the command center respectively act as sources and destination.

In this paper, we try to address the following question: *how to minimize the total tardiness of flows in heterogeneous systems with the existence of the intermediate processing constraint?* We perform a comprehensive analysis of motivating cases and have several findings. First, dispatching flows to IPNs is essential to build a matching between flows and IPNs. Second, both IPN locations and flow dispatching strategies affect the total tardiness. Based on these observations, we apply matching theory and minimize the total tardiness from two aspects, i.e., IPN locations optimization and flow dispatching strategy.

In the *deployment phase*, we propose the Tardiness-aware Deferred Acceptance algorithm (TDA) to evaluate the service capacity of IPN deployment schemes and optimize IPN locations. In the *operation phase*, we combine the Power-of-D [18] paradigm and matching theory, and design the Tardiness-aware Power-of-D algorithm (TPD) for online flow dispatching. Evaluations in real-world scenarios show that both TDA and TPD outperform baseline algorithms significantly and our solution can effectively minimize the total tardiness of applications in heterogeneous systems.

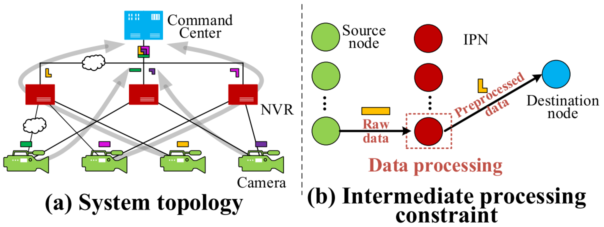


Fig. 2. An example of intermediate processing constraint.

The rest of this paper is organized as follows. Based on the analysis of motivating cases in Section 2, we perform problem formulation in Section 3. The algorithms TDA and TPD are proposed in Sections 4 and 5, respectively. Section 6 performs extensive evaluations based on real-world systems. Section 7 lists the related work and Section 8 concludes the paper.

2 Example and Motivating Cases

In this section, we give an example of intermediate processing constraint, which works as a background of this paper, followed by the motivating cases.

2.1 Example

Fig. 2 shows a smart city [4]. A typical application in such a system is that when capturing traffic violations at some point, distributed cameras transmit videos and snapshots to the command center. Due to the limited processing capacity of the camera embedded system [44], image processing needs to be migrated to one of the IPNs (e.g., network video recorders, NVR). A dispatcher collects the system status (e.g., transmission latency, load on IPNs) and chooses the best IPNs to process and relay the videos and snapshots to the command center.

2.2 Look at Flow Dispatching from the Matching Perspective

Matching theory [45] tries to build a stable matching between two sets of elements. Most researches on this topic trace their history to the work of Gale and Shapley [46]. According to the number of elements in each stable matching pair, the matching models could be divided into three categories: one-to-one matching (e.g., stable-marriage [46], [47]), many-to-one matching (e.g., college admission [45], [46]) and many-to-many matching (e.g., job matching [48]).

A representative many-to-one matching model is College Admission Problem (CAP) [46], where a set of students apply for a set of colleges and each college only accepts a specific number (*quota*) of students. Each student has a strict order of preference over the colleges and each college also has a strict order of preference over the students. CAP aims to build a stable matching without unstable pairs[3] so that every student is matched to at most one college and no college has more assignees than its quota.

We observe that flow dispatching with intermediate processing constraint is similar to CAP. In flow dispatching, each flow is relayed to exactly one IPN and one IPN can

[3] College x and student y form an unstable pair if: x prefers y to one of its admitted students and y prefers x to assigned college.

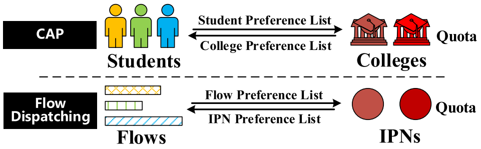


Fig. 3. From CAP to flow dispatching.

process multiple flows. Flows prefer IPNs with higher processing capacity and lower transmission latency to minimize processing time and routing delay, while IPNs prefer flows with shorter sizes and earlier deadlines to minimize the expected queueing delay and tardiness. Regarding flows as students and IPNs as colleges, we can heuristically map flow dispatching into the CAP as shown in Fig. 3. Particularly, we can think of the stable matching in CAP as a tradeoff between the students' and the colleges' preferences, i.e., joint optimization of both sides' wellbeing. Thus, from the perspective of matching theory, a trade-off between processing time, routing delay and queueing delay is naturally achieved by building a matching between IPNs and flows.

2.3 Why Consider Both Flow Routing and Scheduling?

As illustrated in Fig. 4a, two flows are generated at time 0 (f_1 from S_1 to D with deadline $4t$, and f_2 from S_2 to D with deadline $6t$) and we dispatch them with intermediate processing constraint. Resource requirements of f_1 and f_2 are 3 units and 4 units, respectively. We define the length of a flow as the processing time needed when it is processed at an IPN with processing speed 1. The length of f_1 and f_2 are $3t$ and $4t$, respectively. Transmission latency on link $\langle S_1, K_1\rangle$ and $\langle K_1, D\rangle$ are both $3t$, with $1t$ on the other links. The resources on IPNs K_1 and K_2 are 6 units and 5 units, respectively. The processing speed of K_1 and K_2 are respectively 2 and 1.

We assume that the flow dispatched to a full-loaded IPN has to wait for the IPN's idle state and its processing time is equal to the flow length divided by the processing speed of IPN. We define the total tardiness [12] of the application as $\sum_{m\in\mathcal{M}} T_m$, where $\mathcal{M}$ is the set of flows, $T_m = \max\{0, C_m - e_m\}$ is the tardiness of flow m, C_m is the flow completion time, and e_m is the deadline.

Consider Scheduling Only. Scheduling aims to achieve the minimal handling time of flows. Since K_1 owns faster processing speed than K_2, only considering scheduling results in dispatching both f_1 and f_2 to K_1, as shown in Fig. 4b. Hence, routing delay of f_1 and f_2 are $6t$ and $4t$, respectively. Since both flows are completed without queueing delay, the completion time of f_1 and f_2 are $6t + 3t/2 = 7.5t$ and $4t + 4t/2 = 6t$, respectively. Thus, the total tardiness is $(7.5t - 4t) + (6t - 6t) = 3.5t$.

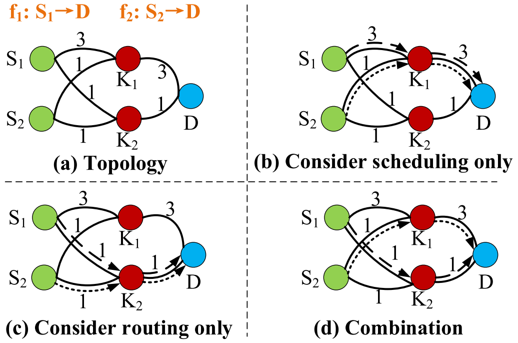


Fig. 4. Case study for joint consideration.

Consider Routing Only. Tardiness-aware routing [14] aims to seek an IPN on the shortest path. As shown in Fig. 4c, only considering routing results in dispatching f_1 to K_2 with routing delay $2t$, and thus its completion time is $1t + 3t/1 + 1t = 5t$. When f_1 is processed on K_2, f_2 has to wait for $3t$. As a result, the queueing delay of f_2 is $3t$. Note the routing delay of f_2 is $2t$, hence its completion time becomes $2t + 3t + 4t/1 = 9t$. The total tardiness is $(5t - 4t) + (9t - 6t) = 4t$.

Combination. As illustrated in Fig. 4d, the global optimal strategy is to consider both routing and scheduling by dispatching f_1 to K_2 with routing delay $2t$ and completion time $2t + 3t/1 = 5t$ while dispatching f_2 to K_1 with routing delay $4t$ and completion time $4t + 4t/2 = 6t$. The total tardiness is $(5t - 4t) + (6t - 6t) = 1t$.

The above example reveals that the total tardiness is hardly minimized within a single dimension. It shows the potential to achieve the global optimal results by considering both routing and scheduling in flow dispatching.

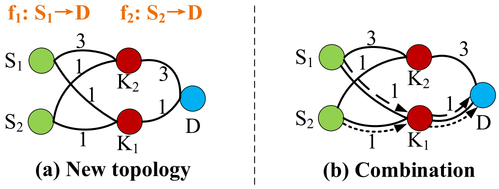


Fig. 5. Case study for the IPN deployment optimization.

2.4 Why Optimize IPN Locations?

As analyzed above, the joint consideration of routing and scheduling in flow dispatching is critical for total tardiness minimization. However, we note that the locations of IPNs lay a foundation for flow dispatching.

Swapping K_1 and K_2 in Fig. 4, we get the instance shown in Fig. 5a. Again, we dispatch f_1 and f_2 with the presence of intermediate processing constraint. Fig. 5b shows the result when we dispatch the flows with the *combination* strategy. The optimal choice is dispatching both f_1 and f_2 to K_1. The routing delay of f_1 and f_2 are both $2t$. The completion time of f_1 is $2t + 3t/2 = 3.5t$. Since f_2 has to wait for $1.5t$ in the queue, the completion time of f_2 is $2t + 1.5t + 4t/2 = 5.5t$. The total tardiness is 0.

It shows that, even if we dispatch flows considering both routing and scheduling, different IPN locations may result in different tardinesses. It motivates us to minimize the total tardiness from two aspects, i.e., IPN locations optimization and flow dispatching strategy.

3 Problem Formulation

We formulate the matching problem between flows and IPNs in this section. Table 1 summarizes the notations in the formulation.

We express the system in the triad of vertices, edges, and IPNs, i.e., $\mathcal{G}(\mathcal{N}, \mathcal{E}, \mathcal{K})$. $\mathcal{K}$ denotes the set of K IPNs equipped with computing capacity, and $\mathcal{M}$ denotes the set of M flows integrated to finish a soft real-time application. For a given IPN k, we use Q_k to denote its amount of resources, and v_k to denote its processing speed. For a given flow m, we use h_m, q_m and e_m to denote its flow length, resource requirement and deadline, respectively.

Once generated, each flow $m \in \mathcal{M}$ is relayed from source to the destination via exactly one IPN $k \in \mathcal{K}$ that satisfies $Q_k \geq q_m$. The flow joins a priority queue after arriving at an IPN. We assume that the priority queues are of infinite sizes, i.e., IPNs can theoretically buffer infinite flows. Multiple flows could be processed in parallel if the total resource requirement of the flows is less than the total resource of the IPN. We focus on flow-level solutions and suppose that flow processing cannot be interrupted.

We use g_m, p_m and u_m to denote the generation time, processing time and queueing delay of flow m, respectively. Particularly, p_{mk} denotes the processing time of flow m on IPN k, which is equal to $\frac{h_m}{v_k}$. We use l_{mk} to denote the routing delay of flow m when it is dispatched to IPN k. Let ω_{mk} be the transmission latency between the source of flow m and IPN k, and ω'_{mk} be the transmission latency between IPN k and the destination. Thus, $l_{mk} = \omega_{mk} + \omega'_{mk}$. We assume that transmission latency is proportional to path length for convenience.

According to Fig. 1, we define the completion time C_m of flow m as $C_m = u_m + p_m + l_m$, i.e., the sum of queueing delay, processing time and routing delay of m. Our objective is to build a matching between all flows and IPNs to minimize the total tardiness, i.e., $\min \sum_{m \in \mathcal{M}} T_m$, where the tardiness of flow m is $T_m = \max\{0, C_m - e_m\}$. For convenience, we denote T_m as $\{C_m - e_m\}^+$, where $\{x\}^+ = \{\, x, 0,\ x \geq 0 \text{ otherwise}$.

Based on the above descriptions, we present the problem as follows:

$$\min \sum_{m \in \mathcal{M}} T_m \tag{1}$$

$$\text{s.t.} \quad T_m = \{C_m - e_m\}^+ \qquad \forall m \in \mathcal{M} \tag{2}$$

$$C_m = u_m + p_m + l_m \qquad \forall m \in \mathcal{M} \tag{3}$$

$$u_m = u_{mk^*_m} \qquad \forall m \in \mathcal{M} \tag{4}$$

$$p_m = \frac{h_m}{v_{k^*_m}} \qquad \forall m \in \mathcal{M} \tag{5}$$

$$l_m = \omega_{mk^*_m} + \omega'_{mk^*_m} \qquad \forall m \in \mathcal{M} \tag{6}$$

$$\tau_m = g_m + \omega_{mk^*_m} + u_m \qquad \forall m \in \mathcal{M} \tag{7}$$

$$x^t_{mk^*_m} = \begin{cases} 1, & 0, \\ t \in [\tau_m, \tau_m + p_m] & \text{otherwise}, \end{cases} \qquad \forall m \in \mathcal{M} \tag{8}$$

$$\sum_{m \in \mathcal{M}} x^t_{mk} \cdot q_m \leq Q_k, \qquad \forall k \in \mathcal{K}, t \in [0, +\infty), \tag{9}$$

where k^*_m denotes the IPN that flow m is dispatched to, u_{mk} is the queueing delay of flow m on IPN k, x^t_{mk} is the decision variable that is 1 if and only if flow m is being processed at IPN k at moment t, and τ_m denotes the time m starts to be processed. Formula (8) refers to that each flow is assigned one and only one IPN during processing, and the processing is uninterruptible. Formula (9) is the capability constraint, ensuring that the cumulative resource usage on IPN k at any given time does not exceed its capacity.

TABLE 1
Notations in Problem Formulation

Notation	Description
C_m	Completion time of flow m;
e_m	Deadline of flow m;
g_m	Generation time of flow m;
h_m	Length of flow m;
$\mathcal{K}$	Set of IPNs;
K	Number of IPNs in $\mathcal{K}$;
l_m	Routing delay of flow m;
$\mathcal{M}$	Set of flows requiring intermediate processing;
M	Number of flows in $\mathcal{M}$;
p_m	Processing time of flow m;
Q_k	Total resource of IPN k;
q_m	Resource requirement of flow m;
T_m	Tardiness of flow m;
u_m	Queueing delay of flow m;
v_k	Processing speed of IPN k;
x^t_{mk}	1 if flow m is processed at IPN k at discrete time t.
τ_m	Time for flow m that starts to be processed;
ω_{mk}	Transmission latency from source of flow m to IPN k;
ω'_{mk}	Transmission latency from IPN k to destination;

Since there are discrete variables (e.g., M, K and x^t_{mk}) and continuous variables (e.g., e_m, q_m, h_m, g_m), this problem is a non-linear mixed-integer programming problem. Brucker et al. summarized the complexity for scheduling problems [49] and the multi-processer scheduling problem for minimizing the total tardiness is NP-hard. Since our problem has higher complexity than multi-processer scheduling, we focus on the near-optimal value of the total tardiness.

It is easy to see that the processing time p_m and routing delay l_m depends on the routing strategy (i.e., the choice of IPN), and the queueing delay u_m depends on the scheduling strategy of the IPN. In other words, to minimize the total tardiness, we should consider both routing and scheduling decisions. Especially, k^*_m is essentially the matching between flows and IPNs. In other words, matching theory solves the problem by determining proper values of k^*_m.

According to Section 2.3, in addition to the flow dispatching strategy, the IPN locations affect the tardiness in an implicit way. Hence, in the following sections, we minimize the total tardiness from two aspects, i.e., optimize IPN locations in the deployment phase and optimize the flow dispatching strategy in the operation phase.

4 Deployment Phase: IPN Location Optimization

IPN locations lays a foundation for flow dispatching. However, it is non-trivial to directly compute the optimal IPN deployment scheme. We note that, due to factors such as cost, electricity and terrain, the number of feasible deployment schemes in real-world systems is usually limited. Even in large-scale scenarios, we can use pruning or divide-

and-conquer strategies to easily reduce the number of candidate deployment schemes. Thus, in this section, we attempt to optimize IPN locations in an indirect way: we evaluate the service capacity of all feasible IPN deployment schemes and select the optimal one that with the strongest service capacity. The question is: how to evaluate the service capacity of a given IPN deployment scheme?

To answer this question, we use stress test to evaluate the real service capacity of the scheme, since performance degradation usually happens when distributed systems are under pressure. It is a common practice to stress the distributed systems with concurrent flows [50], [51]. Hence, we consider that an application generates concurrent flows from every source to the destination in this phase. When we dispatch these flows considering both routing and scheduling, the total tardiness would indicate the service capacity of the IPN deployment scheme under stress. The smaller the total tardiness, the better the IPN deployment scheme.

4.1 CAP and The Deferred Acceptance Algorithm

As demonstrated in Section 2.1, we can map flow dispatching to a matching process, i.e., CAP. Regarding to CAP, the deferred acceptance algorithm is a classic solution, which is described as follows.

Iteration 1: Each student applies to the first choice of college in his preference list. Each student can apply to only one college that has never rejected him. Suppose college k has x_k applicants now. Each college k puts top φ_k ($x_k > \varphi_k$) students in its preference list into its prospective admission list $\mathcal{Y}_k$ and then rejects the other $x_k - \varphi_k$ applicants. If $x_k \leq \varphi_k$, put the x_k applicants into $\mathcal{Y}_k$.

Iteration i: All the students eliminated in the last iteration continue to apply to their ith choice in their preference lists, and then each college k takes both new applicants and the ones already in $\mathcal{Y}_k$ into account. Suppose college k has x_k applicants. Every college k puts top φ_k ($x_k > \varphi_k$) students in its preference list into $\mathcal{Y}_k$ and rejects the other $x_k - \varphi_k$ applicants. If $x_k \leq \varphi_k$, then put the x_k applicants into $\mathcal{Y}_k$.

Repeat *Iteration i* until every student is either in a prospective admission list of a college, or rejected by all the colleges in his preference list. It is worth mentioning that the preference lists of students and colleges are both constant during each iteration. A student cannot apply to other colleges if he is rejected by all the colleges in his preference list.

4.2 Tardiness-Aware Deferred Acceptance Algorithm

Although the deferred acceptance algorithm performs well in CAP, there are two minor differences between flow dispatching and CAP, which makes the classic deferred acceptance algorithm inefficient in our problem. First, both flows and IPNs own diverse properties, and their preferences are not as intuitive as colleges' and students' preferences. Second, each college in CAP can admit a quota of φ students, while IPNs can theoretically admit (buffer and process) infinite flows in our problem. Therefore, we face two challenges to map flow dispatching to CAP: a) how to define the preference lists reasonably and efficiently, and b) how to set an appropriate quota for each IPN.

Preference List Generation. Every flow m has a preference list $\mathcal{A}_m$ of IPNs (IPNs in $\mathcal{A}_m$ are also in $\mathcal{K}$). Also, every IPN k has a preference list $\mathcal{B}_k$ of flows (flows in $\mathcal{B}_k$ are included in $\mathcal{M}$). We regard the top of the preference list as the first choice. Note that tardiness is tightly coupled with both routing delay and handling time on IPNs. Flows and IPNs try to minimize the total tardiness by generating proper preference lists, however, from different perspectives.

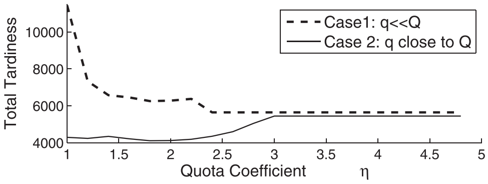


Fig. 6. Total tardiness versus quota coefficient η.

- $\mathcal{A}_m$: For flow m and IPN k, we define standardized variable θ_l in inverse proportion to routing delay l_{mk}, and θ_{pu} in inverse proportion to the flow's expected handling time on the IPN, i.e., the sum of processing time p_{mk} and queueing delay u_{mk}. The preference $\mathcal{A}_m$ of flow m for IPNs is obtained based on $\theta_l + \theta_{pu}$, which is essentially a trade-off between the routing delay and the expected handling time.
- $\mathcal{B}_k$: Similarly, we define standardized variable θ_e in proportion to flow deadline e_m, and θ_h in proportion to the flow length h_m. IPN k generates its preference list $\mathcal{B}_k$ depending on $\theta_e + \theta_h$, which is essentially a trade-off between the due dates and flow length.

The preference lists in CAP are constant during all iterations, which ensures the matching stability of the classic deferred acceptance algorithm. However, in our problem, the handling time depends on the processing sequence according to preference list $\mathcal{B}$. To calculate the expected handling time for a flow, we just temporarily add it into IPN's prospective admission list $\mathcal{Y}$ and schedule it with the other flows on the IPN. Since there exists a dependency between the flow's expected handling time and the interaction of all flows in the queue, we update the preference lists of flows dynamically during each iteration.

Quota Setting. In our problem, an IPN can theoretically admit infinite flows (although may induce long tardiness). Nevertheless, the definition of quota is reasonable since we seldom process all the flows on the same IPN. It contributes to load balance by avoiding overloading the popular IPNs. However, setting a proper quota for each IPN is non-trivial, because we find that the total tardiness has a dependency on the quota. Assume there are $K = 10$ IPNs in the system, and $M = 50$ flows are generated. For simplicity, we set quota of IPN k as $\varphi_k = \lceil \eta \cdot \frac{M}{K} \rceil$ $(k = 1, 2, \ldots, 10)$. Fig. 6 shows the total tardiness varies with the quota coefficient η. In the case that IPNs equipped with sufficient resources, i.e., one IPN can serve several flows at a time ($q_m \ll Q_k$), the total tardiness decreases with the increase of η. However, when one IPN can only serve one or two flows at a time (q_m close to Q_k), the total tardiness increases with η.

To mitigate the challenge of quota dependency, we propose a technical solution that set an initial quota for each IPN and gradually increase the preferred IPNs' quota in each iteration. We define the preferred IPNs as the IPNs having more applicants than its quota φ_k. For each IPN k ($k \in \mathcal{K}$), we define a variable η_k in proportion to its processing speed v_k. To ensure $\varphi_k \in [1, M]$, the initial quota of each IPN is denoted as

$$\varphi_k = \left\lceil \eta_k \cdot \frac{M}{K} \right\rceil, (\eta_k \in (0, K], \forall k \in \mathcal{K}). \tag{10}$$

By overcoming the challenges of preference list generation and quota setting, we finally propose the Tardiness-aware Deferred Acceptance algorithm (TDA) detailed in Algorithm 1.

Algorithm 1. Tardiness-Aware Deferred Acceptance

```
Input:
  $\mathcal{G}(\mathcal{N}, \mathcal{E}, \mathcal{K})$, $\mathcal{M}$, $\mathcal{A}$, $\mathcal{B}$, $\varphi_k(k \in \mathcal{K})$;
Output:
  Total tardiness of flows;
for all $m, k$ that $m \in \mathcal{M}$, $k \in \mathcal{K}$ do
  Define standardized variable $\theta_v$ in proportion to $v_k$;
  $\mathcal{A}_m = GetPreferenceListA(\theta_l, \theta_v)$;
  $\mathcal{B}_k = GetPreferenceListB(\theta_e, \theta_h)$;
end for;
$i \leftarrow 0, \mathcal{Y}_k \leftarrow \emptyset(k \in \mathcal{K})$;
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
    $H_{mk} = GetExpectedHandleTime(\mathcal{Y}_k)$;
    $\mathcal{A}_m = GetPreferenceListA(\theta_l, \theta_{pu}(H_{mk}))$;
    $\mathcal{Y}_k = \mathcal{Y}_k \bigcup \{m | GetTopItem(\mathcal{A}_m) = k\}$;
  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;
for all $k, y$ that $k \in \mathcal{K}$, $y \in \mathcal{Y}_k$ do
  Dispatch flow $y$ to IPN $k$;
end for;
Each IPN $k$ processes flows on it according to $\mathcal{B}_k$;
return Total tardiness $T_{\mathcal{M}}$;
```

The iterations of TDA are outlined as below.

Iteration i: Every flow m that is not assigned to an IPN applies to its first choice of IPN in preference list $\mathcal{A}_m$, so each IPN k has x_k applicants. According to preference list $\mathcal{B}_k$, every IPN puts top φ_k ($x_k > \varphi_k$) flows into its prospective admission list $\mathcal{Y}_k$ before rejecting the other $x_k - \varphi_k$ applicants. After that, $\triangle\varphi_k$ ($1 \leq \triangle\varphi_k \leq \lfloor \frac{M}{K} \rfloor$) is added to the φ_k of the preferred IPNs. It is worth noting that a larger $\triangle\varphi_k$ gives a faster convergence but also a larger range of sub-optimality in the end. If $x_k \leq \varphi_k$, put the x_k applicants into $\mathcal{Y}_k$.

Repeat *Iteration* i until every flow is in a prospective admission list of a certain IPN. After that, each IPN processes flows assigned to it according to its preference list.

Note that no matter whether an IPN has rejected a flow ever before, the flow can always apply to the IPN in a new iteration. Besides, because the preference lists of flows change dynamically, in each iteration, flows could apply to the IPNs based on $\theta_l + \theta_{pu}$ to minimize their expected completion time, which guarantees that no flow suffers infinite tardiness. Since the quota of IPNs increases gradually, no flow will be dropped in TDA, which is different from the classic deferred acceptance algorithm.

5 Operation Phase: Online Flow Dispatching

In the operation phase, according to the prior work [52], we assume the flows are generated dynamically based on a Poisson process, which is a common practice to model online activities. Flow dispatching runs in an event-driven style as shown in Algorithm 2. Flows are dispatched based on regular updates of the system status. For each IPN k, the dispatcher regularly collects the transmission latency (i.e., ω_{ik} and ω_{kj}) in *Step 4* and the current workload of IPNs in *Step 5*. If a new flow m is generated, we select an IPN k (*Step 7*) and route m to k (*Step 8*). After that, m is scheduled and processed on IPN k (*Step 9*).

Algorithm 2. High Level of Flow Dispatching

```
Input:
  $\mathcal{G}(\mathcal{N}, \mathcal{E}, \mathcal{K})$;
Output:
  Flow dispatching scheme;
1:  repeat
2:    $WaitForEvent(\&event)$;
3:    if $event == collect\_information$ then
4:      $CollectLatency()$;
5:      $CollectWorkload()$;
6:    else if $event == flow\_generation$ then
7:      Select an IPN $k$ to process the new flow $m$;
8:      Dispatch flow $m$ to IPN $k$;
9:      IPN $k$ schedules and processes flow $m$;
10:   end if;
11: until no more flows are generated;
```

In Section 4, we show that the TDA is good at handling concurrent flows in static scenarios. Since sequential flow is a special case of concurrent flows whose degree of concurrency is 1 (i.e., M = 1), TDA can also handle sequence flows. However, we argue that TDA is insufficient in online scenarios. In a real-world system, the system status are collected regularly, and moving a flow from the source to the IPN induces time delay. The system status collected may be out-of-date, approximate or even erroneous. It is unrealistic to assume the information is up to date. Researches show that having jobs go to the apparently least loaded server may hurt performance significantly [53]. Hence, it is not a good idea to make decisions entirely based on these information. In this section, we combine the Power-of-D in the Supermarket Model [18] with matching theory for online flow dispatching.

5.1 Supermarket Model and the Power-of-D Paradigm

The Supermarket Model [18] considers the following scenario: customers arriving as a Poisson stream of rate $n\lambda(0 < \lambda < 1)$, where n is the number of homogeneous

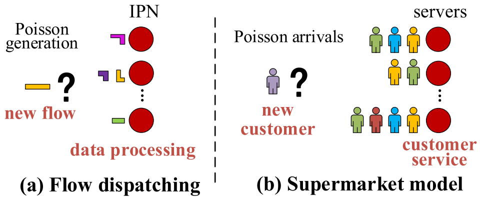


Fig. 7. Flow dispatching versus Supermarket model.

servers. Customers are served in a FIFO manner at the servers. The Supermarket Model aims to dispatch customers to the “right” server to minimize the expected sojourn time of each customer. Online flow dispatching and the Supermarket Model are compared in Fig. 7. We can see that, if we regard flows as customers and IPNs as servers, flow dispatching is similar to the Supermarket Model.

A classic solution to Supermarket Model is the Power-of-D paradigm [18], [54], where the customer samples d servers uniformly and chooses the server with the shortest queue for service. Analysis [18] shows that $d = 2$ leads to exponential improvements over $d = 1$, whereas $d = 3$ results in only a constant factor better than $d = 2$. Hence, Power-of-D is also referred as Power-of-Two. Randomness in sampling the candidate servers makes Power-of-D extremely effective in real-world systems. The randomness sheds light on our solution, and we propose the Tardiness-aware Power-of-D for online dispatching.

5.2 Challenges

Although flow dispatching is similar to the Supermarket Model, there are also differences between them, which pose challenges for applying Power-of-D in flow dispatching.

Unbalanced Load. The servers in the Supermarket Model are homogeneous, while the IPNs are heterogeneous. In the heterogeneous system, Power-of-D performs worse than in the homogeneous situation [19]. This is because sampling servers uniformly leads to congestion at slow servers and increases the sojourn time of customers. Besides, in heterogeneous systems, the performance of Power-of-D is sensitive to the value of d, and inappropriate values of d may result in unbalanced workload among servers [21], which also results in increment of sojourn time.

Optimality Standard. Power-of-D chooses the server with shorter queue to serve the arrival customer to reduce the expected sojourn time (handling time in Fig. 1). However, shorter queue does not necessarily mean shorter queueing delay, since few long flows may result in a much longer processing time than many short ones. Besides, we focus on not only handling time but also routing delay, which is not considered by the Supermarket Model.

Schedule Strategy. Power-of-D aims to assign a proper server to the customer, and customers are then served according to the FIFO rule. Since no prioritization occurs in FIFO, Power-of-D is inefficient to meet the deadlines of flows (consider that a flow has to wait even if it already misses its deadline, which induces increment of tardiness).

5.3 Challenge Mitigation

To mitigate the challenge of *unbalanced load*, instead of sampling IPNs uniformly, we determine the sampling probability distribution $\mathcal{F}$ according to the system status. Mukhopadhyay et al. [19] showed that in the heterogeneous Supermarket Model, the mean sojourn time is minimized when the sampling probability of server k is proportional to its processing speed, i.e., $\mathcal{F}(k) = \frac{v_k}{\sum_{i\in\mathcal{K}} v_i}$. However, we consider not only handling time but also routing delay. Thus, we treat the routing delay l_{mk} as a part of the ”real” processing time and define the apparent processing speed v'_k of IPN k as $v'_k = v_k \cdot \frac{p_{mk}}{(p_{mk}+l_{mk})}$. In other words, it seems that the processing speed of IPNs is slower than v_k because of the routing delay. Note that transmission latency may be constantly changing in operation, hence v'_k is also correspondingly updated. With the definition of apparent processing speed, we compute the sampling probability of IPN k by the following equation:

$$\mathcal{F}(k) = \frac{v'_k}{\sum_{i\in\mathcal{K}} v'_i}, \tag{11}$$

where the sampling probability of IPN k is proportional to its apparent processing speed v'_k. By this way, the IPN with higher processing speed and shorter transmission latency tends to serve more flows, avoiding long relay paths and potential congestions at slow IPNs.

Moreover, we decide the value of d carefully to guarantee balanced workload distribution. Wieder et al. [20] proved the tight lower and upper bounds of d to maintain a balanced workload in a heterogeneous Supermarket Model, where the lower bound is expressed as

$$d_{lower} = (1-\varepsilon) f(\alpha, \beta), \tag{12}$$

and the upper bound is expressed as

$$d_{upper} = (1+\varepsilon) f(\alpha, \beta). \tag{13}$$

In the equations, $\varepsilon \in [0, 1]$, $f(\alpha, \beta) = (\ln\frac{\alpha\beta-1}{\alpha-1})/(\ln\frac{\alpha\beta-1}{\alpha\beta-\beta})$, α and β are real numbers to make sure the sampling probability of IPN $\mathcal{F}(k) \in \left[\frac{1}{\alpha K}, \frac{\beta}{K}\right]$, where K is the number of IPNs. We set d as the minimum integer in $[d_{lower}, d_{upper}]$ in PDT for computation efficiency.

In terms of *optimal standard*, among the d candidates, we choose the IPN k that

$$\underset{1\le k\le K}{\arg\min} \left(\sum\nolimits_{i\in Queue_k} p_{ik} + p_{mk} + l_{mk}\right), \tag{14}$$

where $Queue_k$ stands for the set of flows waiting in the queue on IPN k. By this way, IPNs with lighter workload, faster processing speed and shorter transmission latency are preferred in flow routing, which is consistent with the flows’ preferences for IPNs in the deployment phase. The expected completion time tends to be minimum by choosing IPN k.

Regarding to *schedule strategy*, IPN picks and processes the flow i that satisfies

$$\underset{i\in Queue_k}{\arg\min} (\theta^i_e + \theta^i_h), \tag{15}$$

where θ^i_e and θ^i_h are the standardized variable related to the deadline and length of flow i, respectively. As a result, flows with smaller sizes and earlier deadlines are given higher priorities, which are in line with IPNs’ preferences for flows in the deployment phase. Particularly, to avoid starvation of long flows, flows that miss their deadlines are given the highest priority.

5.4 Tardiness-Aware Power-of-D

Finally, combining the Power-of-D paradigm and matching theory, we propose the Tardiness-aware Power-of-D algorithm for online dispatching (Algorithm 3).

Algorithm 3. Tardiness-Aware Power-of-D

```
Input:
    $\mathcal{G}(\mathcal{N},\mathcal{E},\mathcal{K})$, new flow $m$, $v_k$ $(k \in \mathcal{K})$, $l_{mk}$ $(k \in \mathcal{K})$;
 1: for all $k$ that $k \in \mathcal{K}$ do
 2:     $p_{mk} = \frac{h_m}{v_k}$;
 3:     $v'_k = v_k \cdot \frac{p_{mk}}{(p_{mk}+l_{mk})}$;
 4: end for;
 5: for all $k$ that $k \in \mathcal{K}$ do
 6:     $\mathcal{F}(k) = \frac{v'_k}{\sum_{i \in \mathcal{K}} v'_i}$;
 7: end for;
 8: Calculate $d$ according to Formulae (12) and (13);
 9: Sample $d$ candidates with probability distribution $\mathcal{F}$;
10: Route $m$ to the IPN $k$ satisfying Formula (14);
11: IPN $k$ schedules and processes flow $m$ according to
    Formula (15);
```

For a new flow m, *Step* 2 calculates the processing time if m is processed by k. *Step* 3 updates the apparent processing speed v'_k of each IPN based on p_{mk}. *Step* 6 computes the sampling probability $\mathcal{F}(k)$ of IPN k according to v'_k. *Step* 6 calculates the value of d according to Formulae (12) and (13). After sampling d candidates with distribution $\mathcal{F}$ in *Step* 9, *Step* 10 selects IPN k to process flow m where m tends to be completed earliest. After that, IPN k inserts m into its priority queue (*Step* 11), where the flow with the highest priority is processed first.

Algorithm 3 is a linear time algorithm, and the time complexity is proportional to the number of IPNs, i.e., $O(|K|)$. The number of IPNs is usually relatively small in real-world scenarios. Thus, TPD is expected to work efficiently in operation.

6 Performance Evaluation

In this section, we evaluate the proposed solution in two real-world scenarios, namely *smart campus* and *wind farm inspection*. The smart campus system is deployed in a university, where the transmission latency is relatively low and stable. Thus, handling time contributes the most part of flow completion time in the *smart campus* scenario. The wind farm inspection system is distributed in several provinces and thus the transmission latency is higher and the jitters are severer. As a result, handling time and routing delay contribute almost equally to flow completion time in the *wind farm inspection* scenario.

In the evaluation, we seek to understand: 1) What are the performance of our solution to measure the service capacity of an IPN deployment scheme? 2) With different traffic patterns, how will be our solution compared to standard heuristics for online flow dispatching? 3) How much performance improvement can our solution bring to the applications in real-world systems?

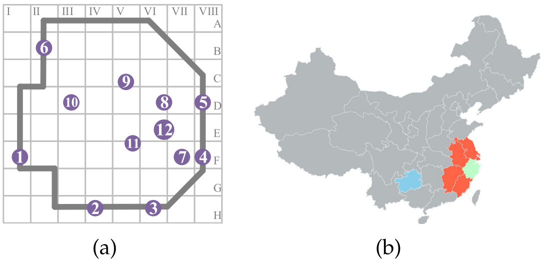


Fig. 8. (a) *Smart Campus*: The campus outline with key locations. (b) *Wind Farm Inspection*: Locations of the wind farm (green), the DCs (red) and the command center (blue).

Summary of results:

- *Service capacity measurement:* We use the Tardiness-aware Deferred Acceptance (TDA) algorithm and four other algorithms to measure the service capacity of a given IPN deployment scheme. Evaluation shows that TDA estimates the service capacity accurately, which helps optimizing the IPN locations in the deployment phase.
- *Online flow dispatching:* We use the Tardiness-aware Power-of-D (TPD) algorithm and three standard heuristics to dispatch flows with different flow patterns. With both stable and flash crowd flows, TPD outperforms the others significantly in terms of total tardiness. It shows that TPD can effectively handle both stable and unexpected surge of traffic.
- *Case study:* We validate the effectiveness of the proposed solution with case studies based on real-world systems. We use TDA to measure the service capacity of candidate IPN deployment schemes and use TPD to dispatch flows under these schemes. The results show that our solution effectively improves the performance of data-intensive applications in heterogeneous systems.

6.1 Experimental Setup

6.1.1 *Smart Campus*

The smart campus, which is a part of a large-scale smart city security system, is deployed in a university that spans over 3,000,000 square meters. The smart campus consists of more than 600 cameras, 12 surveillance points and a command center. As shown in Fig. 8a, there are 12 key locations in the university, such as library, teaching building and museum. To better monitor the key locations, every surveillance point is located at one of the key locations. In particular, the location numbered 12 is the main building, which is also the location of the command center. Each surveillance point deploys a Network Video Recorder (NVR). Cameras send video clips and snapshots to one of the NVRs, which transcodes the videos and then sends them to the command center. In the command center, the videos are displayed on large screens and further analyzed (e.g., fire detection and tripwire detection).

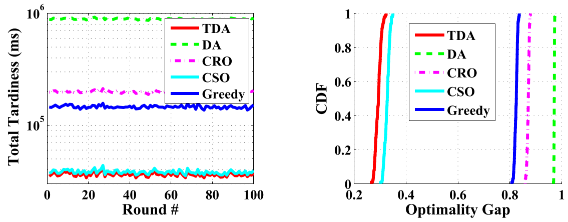
We consider a tripwire detection application where the cameras capture moving objects at 30 fps (frames per second) and upload video clips to the command center via NVRs. An analysis based on a large video analytics dataset [55] shows that video clips between 5 and 15 seconds are sufficient for tripwire detection. Thus, in the experiment, we assume the video clips captured by the cameras range from 5 to 15 seconds (i.e., 150 frames to 450 frames) in duration. Due to the incremental upgrade strategy, there are two generations of NVRs deployed on campus, including 8 higher performance NVRs and 4 lower performance NVRs. The higher performance NVRs can transcode up to 64 video channels at the same time and the average transcoding rate for each channel is 1,200 fps. The lower performance NVRs can transcode up to 32 video channels at the same time and the average transcoding rate for each channel is 600 fps. We perform field measurements of the network transmission latency at multiple locations on the campus and found that the routing delay between any two points ranges from 0.005 to 0.015 seconds. Thus, we set the routing delay $l_{mk} \in [0.005, 0.03]$ s and assume the transmission latency is in proportion to the relay path length.

Based on the above real-world system configuration, $\forall k \in \mathcal{K}$, $\forall m \in \mathcal{M}$, we set $Q_k = \{64, 32\}$ channels, $v_k = \{1200, 600\}$ fps for each channel, $h_m \in [150, 450]$ frames, $q_m = 1$ channel, $p_{mk} = h_m/v_k$. To guarantee quick responses to tripwires (e.g., trigger the alarm), the total tardiness of the flows should be minimized. In the experiment, we empirically set the deadline of each flow $e_m \in [0.2, 0.3]$ s.

6.1.2 Wind Farm Inspection

The wind farm, which contains more than 30 wind turbines, is located in Zhejiang Province. The wind turbines are up to 100 meters and are easily damaged by lightening strikes. To eliminate the need for dangerous manual inspection, a wind farm inspection system is deployed to provide automatic inspection service [56]. The system consists of 10 Unmanned Aircraft Systems (UASs), 5 media servers and a command center. The UASs capture the videos of the wind turbines and upload the video clips to the media servers via 4G cellular networks. At each inspection, the UASs can take a total of more than 15,000 video clips of the turbines. The media servers process the video clips, such as splicing and tagging, and send the processed videos to the command center. For the sake of robustness, the media servers are rented from 5 Data Centers (DCs) belonged to different providers. As shown in Fig. 8b, the DCs are located in the provinces adjacent to Zhejiang, namely Shanghai, Anhui, Jiangsu, Jiangxi and Fujian. In the command center located in Guizhou, videos from tens of wind farms in South China are analyzed and displayed on the screen wall.

We consider a turbine failure detection application where a machine-learning-based diagnostic model detects turbine failures via video analysis. Commercial solutions show that a typical video clip in a wind farm inspection system is about 10 seconds long and has a frame rate of 30 fps [57]. Due to the financial constraints, there are two types of media servers, including 2 higher performance servers and 3 lower performance ones. The average processing speed of the higher performance media servers is 1000 fps, while that of the lower performance media servers is 500 fps. We suppose both types of media servers process up to one video clip at a time. The transmission latency between the wind farm and the media servers ranges from 0.1 to 0.5 seconds, and that between the media servers and the command center ranges from 0.03 to 0.06 seconds. We also suppose the transmission latency is in proportion to the relay path length.



(a) Total tardiness (b) Optimality gap

Fig. 9. *Smart Campus*: Service capacity measurement.

Finally, $\forall k \in \mathcal{K}$, $\forall m \in \mathcal{M}$, we set $Q_k = 1$ piece, $v_k = \{1000, 500\}$ fps, $h_m \in [30, 300]$ frames, $q_m = 1$ piece, $p_{mk} = h_m/v_k$ and deadline of each flow $e_m \in [0.8, 1]$ s.

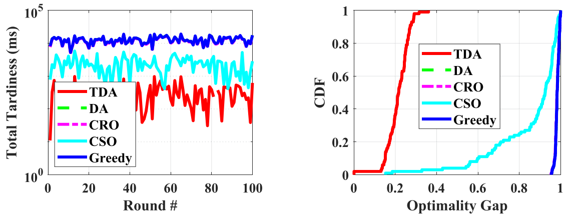
6.2 Service Capacity Measurement

In the deployment phase, we use TDA and the other four algorithms to measure the service capacity of a specific IPN deployment scheme. More specifically, in *smart campus*, we consider the scheme where the 4 lower performance NVRs are deployed at surveillance points 1 to 4 in Fig. 8a and the 8 higher performance NVRs are deployed at the other 8 surveillance points; In the *wind farm inspection*, we consider the scheme where the 3 lower performance media servers are located in the DCs of Shanghai, Jiangsu and Anhui, respectively, and the 2 higher performance media servers are located in the DCs of Jiangxi and Fujian. To stress the systems, we consider applications where every source node generates a flow at the same time (i.e., $M = 600$ in *smart campus* and $M = 10$ in the *wind farm inspection*) and the flows are relayed to the destination node (i.e., command center in both scenarios) by the IPNs (i.e., NVRs in *smart campus* and media servers in *wind farm inspection*). For both scenarios, we run the application for 100 rounds.

The algorithms compared are listed below.

- *TDA*: The Tardiness-aware Deferred Acceptance algorithm designed in Section 4.2. We set $\triangle\varphi = 1$ to yield results that are close to the optimal solutions.
- *DA*: The classic deferred acceptance algorithm described in Section 4.1, where IPNs prefer flows with smaller sizes and earlier deadlines, and flows prefer IPNs with faster processing speed and shorter transmission latency. We set the quota of IPN $\varphi_k = \lceil \frac{M}{K} \rceil$ for simplicity.
- *Consider Routing Only (CRO)*: The routing-only strategy, in which flows are dispatched to the IPN with the shortest routing delay and then processed by the FIFO (First In First Out) rule.
- *Consider Scheduling Only (CSO)*: A modified TDA algorithm that considers flow scheduling only, without taking into account θ_l during updating preference list $\mathcal{A}$.
- *Greedy*: The flows are dispatched to the IPN with the smallest $l_{mk} + \frac{p_{mk}}{Q_k}$, and then processed according to the Earliest Due Date (EDD) priority rule [58].

Total Tardiness. Figs. 9a and 10a respectively compares the total tardiness when applying the above algorithms in



(a) Total tardiness (b) Optimality gap

Fig. 10. *Wind Farm Inspection*: Service capacity measurement.

the 100 rounds of simulations in *smart campus* and *wind farm inspection*. In both scenarios, TDA significantly outperforms other algorithms in all the cases. We can see that due to inefficient stability and quota dependency, DA performs even worse than the CRO or Greedy strategy.

Optimality Gap. To quantitatively analyze the algorithm optimality, for each round of simulation I, we introduce the optimality gap

$$Gap(I) = \frac{Z(I) - Z_{LB}(I)}{Z(I)}, \tag{16}$$

where $Z(I)$ is the total tardiness, and $Z_{LB}(I)$ is the lower bound of the total tardiness. We apply the Lagrangian Relaxation Theory to the problem in Appendix A and obtain the lower bound $Z_{LB}(I)$ by a subgradient optimization algorithm detailed in Appendix B. Note that the lower bound is 0 if the application completes without tardiness.

Figs. 9b and 10b shows the Cumulative Distribution Function (CDF) of the gap on the total tardiness between the algorithms and the lower bound in *smart campus* and *wind farm inspection*, respectively. In both scenarios, the optimality gap of TDA is less than 0.3 in almost 100 percent of the cases. TDA outperforms any other algorithms in optimality and estimates the service capacity well.

The above results show that the proposed solution can accurately estimate the total tardiness of concurrent flows, which reflects the service capacity of a specific IPN deployment scheme. System operators can use TDA to measure the service capacity of candidate IPN deployment schemes and choose the optimal one, which lays a foundation for the total tardiness minimization.

6.3 Online Flow Dispatching

We next validate the effectiveness of our solution in the operation phase. For both scenarios, we consider the same IPN deployment scheme as given in Section 6.2. To simulate the dynamics of flow generation, according to the prior work [19], [52], we consider an application generating sequential flows according to a Poisson Process with mean $K\lambda$, where K is the number of IPNs (i.e., $K = 12$ in *smart campus* and $K = 5$ in *wind farm inspection*).

In *smart campus*, we consider two flow patterns: *Stable* and *Flash Crowd*, [59], [60]. *Stable* describes the general state of the traffic, i.e., the flows are generated at a stable and relatively low rate. *Flash Crowd* describes a surge traffic, i.e., the arrival rate of the flows increases sharply but lasts for a short time. For the stable pattern, we set flow number $M = 10000$ and arrival rate $\lambda = 0.7$. For the flash crowd pattern, we set flow number $M = 600$ and arrival rate $\lambda = 5$. With both patterns, we run the application for 100 rounds.

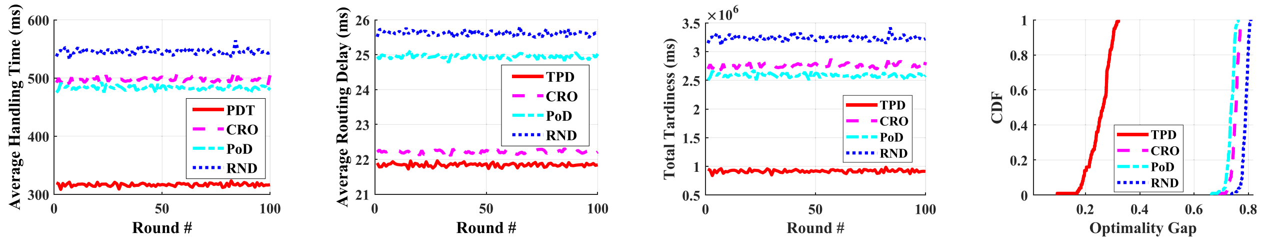
In *wind farm inspection*, as described in Section 6.1.2, the UASs totally take more than 15,000 video clips for each inspection, thus we set flow number $M = 15000$. Since there are only 10 UASs in the inspection system, we only consider the *Stable* pattern for simplicity and set arrival rate $\lambda = 0.1$. We also run the application for 100 rounds in the evaluation.

The algorithms compared are listed below. For PoD, CRO and RND, we assume flows are processed by the EDD priority rule on the IPNs.

- *TPD*: The Tardiness-aware Power-of-D algorithm proposed in Section 5.4.
- *Power-of-D (PoD)*: The Power-of-Two strategy, where two IPNs are sampled uniformly and the one with shorter queue (i.e., fewer flows queued at the IPN) is selected, is a typical consider scheduling only strategy that focuses on handling time minimization.
- *Consider Routing Only (CRO)*: The online-version routing-only strategy, where two IPNs are sampled uniformly and the one with shorter transmission latency is selected.
- *Random (RND)*: Random strategy serves as a baseline, where one IPN is randomly chosen.

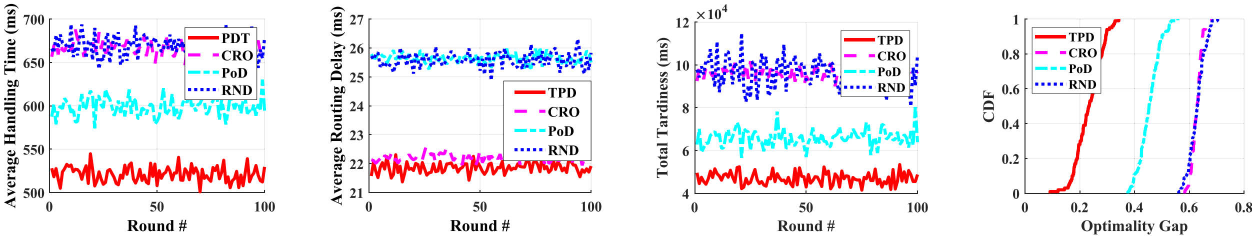
Handling Time. Figs. 11a and 12a compare the average handling time with stable and flash crowd flow patterns in *smart campus*, respectively. Fig. 13a shows the average handling time in *wind farm inspection*. In both scenarios, TPD performs obviously better than the other algorithms, followed by PoD, because both TPD and PoD consider handling time optimization. However, as illustrated in Section 5.2, the classic Power-of-D paradigm is insufficient for heterogeneous systems.

Routing Delay. Figs. 11b and 12b show the average routing delay with different flow patterns in *smart campus*. TPD and CRO gain better performance because both of them take transmission latency into account. Particularly, as



(a) Average handling time (b) Average routing delay (c) Total tardiness (d) Optimality gap

Fig. 11. *Smart Campus*: A comparison of different algorithms for flow dispatching (Stable).



(a) Average handling time (b) Average routing delay (c) Total tardiness (d) Optimality gap

Fig. 12. *Smart Campus*: A comparison of different algorithms for flow dispatching (Flash Crowd).

shown in Fig. 13b, the advantages of TPD and CRO are more obvious in *wind farm inspection*, where the routing delay is more unstable. Moreover, when determining the optimal IPN, TPD usually samples more than two candidates due to Formulae (12) and (13). Thus, it has more opportunities to find an IPN on a shorter path and yield shorter routing delay than CRO.

Total Tardiness and Optimality Gap. As shown in Figs. 11c, 12c and 13c, TPD outperforms the other algorithms in both scenarios. This reveals that TPD is good at handling traffic with different patterns. In particular, we use the optimality gap defined in Formula (16) to analyze the optimality of the algorithms. To compute the lower bound $Z_{LB}(I)$ in online scenarios, for each new flow, we traverse all the IPNs and determine the global optimal IPN and the optimal processing sequence that minimizes the total tardiness of the flows in the system. As demonstrated in Figs. 11d, 12d and 13d, TPD outperforms the other strategies in all the cases in terms of optimality gap. The above results confirm the effectiveness of our solution for online flow dispatching.

6.4 Case Studies

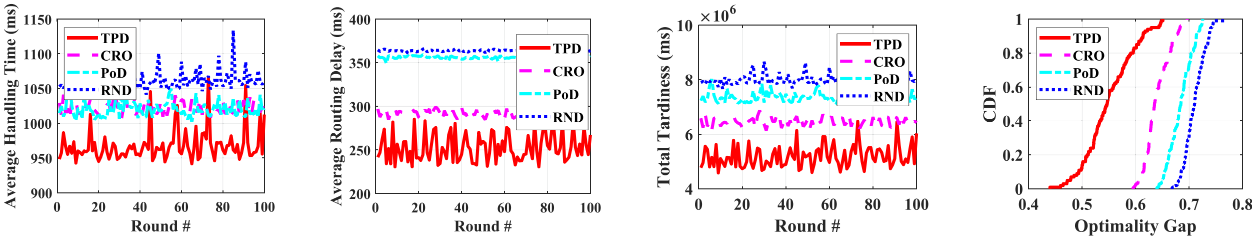
We finally perform case studies to validate the performance of our solution in the real-world scenarios. In *smart campus*, there are $C_{12}^{4} = 495$ possible schemes to deploy the two generations of NVRs at the 12 surveillance points. In *wind farm inspection*, there are $C_{5}^{3} = 10$ possible schemes to rent the two types of media servers from the 5 DCs. For both scenarios, in the deployment phase, we traverse all these deployment schemes and measure their service capacities with TDA; In the operation phase, we use TPD to dispatch flows of different patterns under these schemes. The effectiveness of our solution will be confirmed if: the total tardiness is reduced in the operation phase under the optimal IPN deployment scheme selected by TDA.

Deployment Phase. For both scenarios, under each IPN deployment scheme, we stress the system with the same concurrent flow settings as Section 6.2. Figs. 14a and 15b displays the total tardiness of the flows under every IPN deployment scheme in the two scenarios. In *smart campus*, the total tardiness reaches the minimum value of 34,938 ms under the 324th scheme and maximum value of 38,322 ms under the 495th scheme. It implies that the service capacity of the 324th IPN deployment scheme is the highest and that of the 495th is the lowest. In *wind farm inspection*, the total tardiness is minimized under the 6th scheme and maximized under the 1st scheme. In other words, the 6th deployment scheme outperforms the others in terms of service capacity. The details of the above four schemes are given in Table 2.

Operation Phase. In *smart campus*, under the 324th and 495th IPN deployment schemes, we use TPD to dispatch flows with the same settings as given in Section 6.3. Fig. 14b demonstrates the difference of total tardiness between the two schemes with different flow patterns. With the stable pattern, the total tardiness under the 324th scheme is always shorter than under the 495th scheme, and the maximum difference is up to 41,010 ms. With the flash crowd pattern, in 97 percent of the cases, the total tardiness under the 324th scheme is shorter than under the 495th scheme, and the maximum difference is up to 1,422 ms. Both stable and surge heavy traffic gain better performance under the 324th scheme. In *wind farm inspection*, compared to the 1^{st} scheme, the total tardiness is reduced by up to 5,000,000 ms under the 6th scheme.

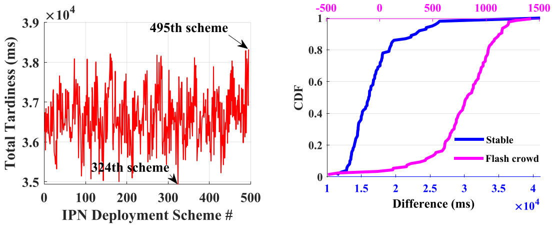
6.5 Summary

The evaluation above confirms the effectiveness of our solution. As shown in Section 6.2, TDA can evaluate the service capacity of IPN deployment schemes accurately in the deployment phase. Section 6.3 shows that TPD outperforms the other dispatching algorithms in terms of total tardiness in the operation phase. In Section 6.4, we show that the performance of applications in both real-world scenarios are obviously improved. In short, the proposed solution can



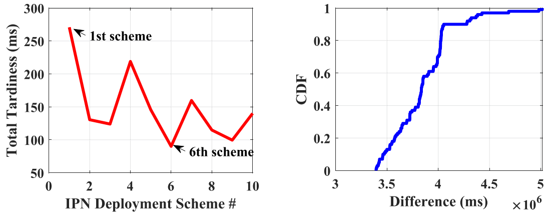
(a) Average handling time (b) Average routing delay (c) Total tardiness (d) Optimality gap

Fig. 13. *Wind Farm Inspection*: A comparison of different algorithms for flow dispatching.



(a) (b)

Fig. 14. *Smart Campus*: (a) Service capacity measurement of candidate IPN deployment schemes. (b) $(T_{\mathcal{M}})_{495th} - (T_{\mathcal{M}})_{324th}$ in the operation phase.



(a) (b)

Fig. 15. *Wind Farm Inspection*: (a) Service capacity measurement of candidate IPN deployment schemes. (b) $(T_{\mathcal{M}})_{1st} - (T_{\mathcal{M}})_{6th}$ in the operation phase.

effectively minimize the total tardiness of data-intensive applications in heterogeneous systems.

7 Related Work

Routing generally focuses on optimizing network resource utilization while not harming the average completion time. Traditional routing strategies usually fall into the general category of Multi-Constrained Path (MCP) allocation problems and a large amount of heuristics and approximation algorithms are proposed to select the path based on statically configured metrics (e.g., hop) and QoS metrics [61] (e.g., bandwidth [16], delay [13] and jitter [15]). Al-Fares et al. proposed Hedera [52], which leverages Simulated Annealing schedulers to distribute flows to balance the traffic load. However, these solutions emphasize on the quality of transmission, which leads to routing delay reduction but not handling time. Power-of-D could significantly improve the expected handling time of the customers in homogeneous systems [18]. Researches extended Power-of-D to heterogeneous systems by considering sampling probability distribution [19], the value of d [20], and the definition of the "best" choice [21]. However, they do not take routing delay (i.e., the cost for a customer to join a specific queue) into account. In a word, these solutions seldom consider the coupling of communication and computation, and thus are inefficient to meet the intermediate processing constraint.

Scheduling aims to optimize one or more evaluation criteria in job processing, e.g., minimizing the average completion time [62]. Lampson et al. [22] first considered state vectors and the properties of process in multi-processing system scheduling. Even et al. [23] pointed out that the multi-commodity flow problem is NP complete. Dogramaci et al. [58] proposed two scheduling rules, namely the Earliest Due Dates first (EDD) and the Shortest Processing Time first (SPT). Bai et al. [63] presented PIAS, which leverages multiple level feedback queues to minimize flow completion time. Zhang et al. [64] proposed a novel metric expansion ratio for flow scheduling without starvation. Recently, there are researches focusing on coflow scheduling [65], where a collection of parallel flows are transferred between two groups of machines. Chowdhury et al. [66] proposed Minimum-Allocation-for-Desired-Duration and Smallest-Effective-Bottleneck-First heuristics for intra- and inter-coflow scheduling, respectively. CODA [67] could automatically collect coflow information and schedule coflows without modifying cluster applications. Zhang et al. [68] proved that coflow scheduling is equivalent to the concurrent open shop problem and proposed a decentralized coflow scheduling solution named D-CAS. However, these solutions do not take into account the collaborative routing strategies and thus are insufficient in scenarios characterized by the intermediate processing constraint. Specifically, in our solution, the priority of a flow is decided by both its length and deadline, which is essentially the trade-off between EDD and SPT.

TABLE 2
Details of IPN Deployment Schemes

Scenario	Scheme No.	Location of Lower Performance IPNs
Smart Campus	324th	3, 5, 7, 12
	495th	9, 10, 11, 12
Wind Farm Inspection	1st	Shanghai, Jiangsu, Anhui
	6th	Shanghai, Nanchang, Fujian

Joint optimization of routing and scheduling is attracting more attention in recent years. Researchers in [32] focused on the power-down strategies for network devices and proposed the energy-efficient flow scheduling and routing strategies. Han et al. [35] proposed a RUSH framework dealing with both the batch-arrival and the sequential-arrival of flow demands in hybrid data centers. Zhao et al. [31] proposed RAPIER, which for the first time demonstrates that routing and scheduling must be jointly considered to optimize the average Coflow Completion Time. Li et al. [36] proposed an online multiple coflow routing and scheduling algorithm with theoretical performance guarantees. Jahanjou et al. [69] presented the first approximation algorithms for scheduling coflows over general network topologies to minimize the total weighted completion time. However, the researches above mainly focus on the bandwidth allocation, which leads to the optimization of routing delay only, while our solution optimizes both routing delay and handling time of flows. Besides, existing studies seldom focus on the intermediate processing constraint in heterogeneous systems and thus are inapplicable to our problem.

8 Conclusion

This paper is devoted to minimizing the total tardiness of data-intensive applications in heterogeneous systems with the intermediate processing constraint.

In the paper, we apply matching theory for the joint optimization of routing delay and handling time, which we found is the key to achieve our goal. Particularly, our solution minimizes the total tardiness from two aspects. In the deployment phase, we evaluate the service capacity of feasible IPN deployment schemes with the Tardiness-aware Deferred Acceptance algorithm, which helps to optimize IPN locations and lays a foundation for flow dispatching. In the operation phase, we dispatch flows with the Tardiness-aware Power-of-D algorithm, which is a combination of Power-of-D and matching theory. We perform extensive evaluations based on real-world systems. The results show that our solution can effectively minimize the total tardiness of data-intensive applications in heterogeneous systems.

Acknowledgments

Ke Xu's work was supported in part by the National Key R&D Program of China under Grant 2018YFB0803405, in part by the China National Funds for Distinguished Young Scientists under Grant 61825204 and in part by the Beijing Outstanding Young Scientist Project. Meng Shen's work was supported in part by the National Natural Science Foundation of China under Grant 61602039, in part by the Beijing Natural Science Foundation under Grant 4192050, and in part by the CCF-Tencent Open Fund WeBank Special Funding. Kun Yang's work was supported by the UK EPSRC Project NIRVANA (EP/L026031/1).

References

[1] M. M. Moazzami, D. E. Phillips, R. Tan, and G. Xing, "Orbit: A platform for smartphone-based data-intensive sensing applications," *IEEE Trans. Mobile Comput.*, vol. 16, no. 3, pp. 801–815, Mar. 2017.

[2] D. Andrei, M. Tornatore, M. Batayneh, C. U. Martel, and B. Mukherjee, "Provisioning of deadline-driven requests with flexible transmission rates in WDM mesh networks," *IEEE/ACM Trans. Netw.*, vol. 18, no. 2, pp. 353–366, Apr. 2010.

[3] C. P. Chen and C.-Y. Zhang, "Data-intensive applications, challenges, techniques and technologies: A survey on big data," *Inf. Sci.*, vol. 275, pp. 314–347, 2014.

[4] R. Kitchin, "The real-time city? big data and smart urbanism," *GeoJournal*, vol. 79, no. 1, pp. 1–14, 2014.

[5] S. Wang, J. Zhang, T. Huang, J. Liu, T. Pan, and Y. Liu, "A survey of coflow scheduling schemes for data center networks," *IEEE Commun. Mag.*, vol. 56, no. 6, pp. 179–185, Jun. 2018.

[6] T. Li, C. S. Magurawalage, K. Wang, K. Xu, K. Yang, and H. Wang, "On efficient offloading control in cloud radio access network with mobile edge computing," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst.*, 2017, pp. 2258–2263.

[7] H. Wang, R. Shea, X. Ma, F. Wang, and J. Liu, "On design and performance of cloud-based distributed interactive applications," in *Proc. IEEE 22nd Int. Conf. Netw. Protocols*, 2014, pp. 37–46.

[8] K. Gardner, M. Harchol-Balter , and S. Borst, "Optimal scheduling for jobs with progressive deadlines," in *Proc. IEEE Conf. Comput. Commun.*, 2015, pp. 1113–1121.

[9] B. Kao and H. GarciaMolina, "Deadline assignment in a distributed soft real-time system," *IEEE Trans. Parallel Distrib. Syst.*, vol. 8, no. 12, pp. 1268–1274, Dec. 1993.

[10] R. McNaughton, "Scheduling with deadlines and loss functions," *Manage. Sci.*, vol. 6, no. 1, pp. 1–12, 1959.

[11] F. Yalaoui and C. Chu, "Parallel machine scheduling to minimize total tardiness," *Int. J. Prod. Econ.*, vol. 76, no. 3, pp. 265–279, 2002.

[12] M. A. Sharaf, P. K. Chrysanthis, A. Labrinidis, and C. Amza, "Optimizing I/O-intensive transactions in highly interactive applications," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2009, pp. 785–798.

[13] S. Chen, M. Song, and S. Sahni, "Two techniques for fast computation of constrained shortest paths," *IEEE/ACM Trans. Netw.*, vol. 16, no. 1, pp. 105–115, Feb. 2008.

[14] T. Korkmaz and M. Krunz, "Multi-constrained optimal path selection," in *Proc. IEEE Conf. Comput. Commun. 20th Annu. Joint Conf. IEEE Comput. Commun. Society*, 2001, pp. 834–843.

[15] 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 *Proc. IEEE 34th Real-Time Syst. Symp.*, 2013, pp. 140–149.

[16] Q. Ma and P. Steenkiste, *Quality-of-Service Routing for Traffic with Performance Guarantees*. New York, NY, USA: Springer, 1997.

[17] F. A. Kuipers and P. F. A. V. Mieghem, "Conditions that impact the complexity of QoS routing," *IEEE/ACM Trans. Netw.*, vol. 13, no. 4, pp. 717–730, Aug. 2017.

[18] M. Mitzenmacher, "The power of two choices in randomized load balancing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 12, no. 10, pp. 1094–1104, Oct. 2001.

[19] A. Mukhopadhyay and R. R. Mazumdar, "Analysis of randomized join-the-shortest-queue (JSQ) schemes in large heterogeneous processor-sharing systems," *IEEE Trans. Control Netw. Syst.*, vol. 3, no. 2, pp. 116–126, Jun. 2016.

[20] U. Wieder, "Balanced allocations with heterogenous bins," in *Proc. 19th Annu. ACM Symp. Parallel Algorithms Architectures*, 2007, pp. 188–193.

[21] A. Mukhopadhyay, A. Karthik, R. R. Mazumdar, et al., "Randomized assignment of jobs to servers in heterogeneous clusters of shared servers for low delay," *Stochastic Syst.*, vol. 6, no. 1, pp. 90–131, 2016.

[22] B. W. Lampson, "A scheduling philosophy for multi-processing systems," *ACM Symp. Operating Syst. Principles*, vol. 11, pp. 347–360, 1968.

[23] S. Even, A. Itai, and A. Shamir, "On the complexity of time table and multi-commodity flow problems," in *Proc. Symp. Foundations Comput. Sci.*, 1975, pp. 184–193.

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

[25] B. Vamanan, J. Hasan, and T. Vijaykumar, "Deadline-aware datacenter TCP (D2TCP)," in *Proc. ACM SIGCOMM Conf. Appl. Technol. Architectures Protocols Comput. Commun.*, 2012, pp. 115–126.

[26] F. Ciucu, F. Poloczek, and J. Schmitt, "Sharp per-flow delay bounds for bursty arrivals: The case of FIFO, SP, and EDF scheduling," in *Proc. IEEE Conf. Comput. Commun.*, 2014, pp. 1896–1904.

[27] C. Y. Hong, M. Caesar, and P. B. Godfrey, "Finishing flows quickly with preemptive scheduling," in *Proc. ACM SIGCOMM Conf. Appl. Technol. Architectures Protocols Comput. Commun.*, 2012, pp. 127–138.

[28] H. Wu, X. Lin, X. Liu, and Y. Zhang, "Application-level scheduling with deadline constraints," in *Proc. IEEE Conf. Comput. Commun.*, 2014, pp. 2436–2444.

[29] S. Albagli-Kim , H. Shachnai, and T. Tamir, "Scheduling jobs with dwindling resource requirements in clouds," in *Proc. IEEE Conf. Comput. Commun.*, 2014, pp. 601–609.

[30] 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 *Proc. IEEE Conf. Comput. Commun.*, 2014, pp. 2166–2174.

[31] 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 *Proc. Conf. Comput. Commun.*, 2015, pp. 424–432.

[32] 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 *Proc. IEEE 34th Int. Conf. Distrib. Comput. Syst.*, 2014, pp. 248–257.

[33] M. Xu, Y. Shang, D. Li, and X. Wang, "Greening data center networks with throughput-guaranteed power-aware routing," *Comput. Netw.*, vol. 57, no. 15, pp. 2880–2899, 2013.

[34] L. Wang, F. Zhang, A. V. Vasilakos, C. Hou, and Z. Liu, "Joint virtual machine assignment and traffic engineering for green data center networks," *ACM SIGMETRICS Performance Eval. Rev.*, vol. 41, pp. 107–112, 2014.

[35] K. Han, Z. Hu, J. Luo, and L. Xiang, "Rush: Routing and scheduling for hybrid data center networks," in *Proc. IEEE Conf. Comput. Commun.*, 2015, pp. 415–423.

[36] Y. Li, H. C. Jiang, H. Tan, C. Zhang, G. Chen, J. Zhou, and F. C. M. Lau, "Efficient online coflow routing and scheduling," in *Proc. ACM Int. Symp. Mobile Ad Hoc Netw. Comput.*, 2016, pp. 161–170.

[37] S. Moharir and S. Shakkottai, "Maxweight versus backpressure: Routing and scheduling in multichannel relay networks," *IEEE/ACM Trans. Netw.*, vol. 23, no. 5, pp. 1584–1598, Oct. 2015.

[38] J. Luo, C. Rosenberg, and A. Girard, "Engineering wireless mesh networks: Joint scheduling, routing, power control, and rate adaptation," *IEEE/ACM Trans. Netw.*, vol. 18, no. 5, pp. 1387–1400, Oct. 2010.

[39] S. O. Ajiboye, P. Birch, C. Chatwin, and R. Young, "Hierarchical video surveillance architecture: A chassis for video big data analytics and exploration," pp. 9407–9407, 2015. [Online]. Available: https://doi.org/10.1117/12.2083937

[40] X. Tang, X. Xu, T. Svensson, and X. Tao, "Coverage performance of joint transmission for moving relay enabled cellular networks in dense urban scenarios," *IEEE Access*, vol. 5, pp. 3001–13009, 2017.

[41] Huawei, "Huawei espace intelligent video surveillance solution," 2018. [Online]. Available: https://e.huawei.com/en/solutions/business-needs/collaboration/video-surveillance/Intelligent-Surveillance, Accessed on: Dec. 2018

[42] Hikvision, "Smart traffic solutions," 2019. [Online]. Available: http://www.hikvision.com/en/Solutions/Traffic, Accessed on: Jan. 2019

[43] Quantum, "Intellegent storage: The foundation of the new surveillance & security infrastructure," 2019. [Online]. Available: https://iq.quantum.com/exLink.asp?34709140OE56Q88I649172 80&ST01870A&view=1, Accessed on: Jan. 2019

[44] I. F. Akyildiz and X. Wang, "A survey on wireless mesh networks," *IEEE Commun. Mag.*, vol. 43, no. 9, pp. S23–S30, Sep. 2005.

[45] A. E. Roth and A. Oliveira, "Two-sided matching: A study in game-theoretic modeling and analysis," in *Econometric Society Monograph Series*, Cambridge, U.K.: Cambridge Univ. Press, 1990.

[46] D. Gale and L. S. Shapley, "College admissions and the stability of marriage," *Amer. Math. Monthly*, vol. 69, pp. 9–15, 1962.

[47] A. E. Roth, *The Economics of Matching: Stability and Incentives.* Catonsville, MD, USA: INFORMS, 1982.

[48] J. Oviedo, "A theory of stability in many-to-many matching markets," in *Proc. Meet. Papers*, 2006, pp. 233–273.

[49] P. Brucker and S. Knust, "Complexity results for scheduling problems," 2019. [Online]. Available: http://www2.informatik.uni-osnabrueck.de/knust/class/, Accessed on: Jul. 1, 2015

[50] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley, "Improving datacenter performance and robustness with multipath TCP," in *Proc. ACM SIGCOMM Conf. Appl. Technol. Architectures Protocols Comput. Commun.*, 2011, pp. 266–277.

[51] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "Vl2: A scalable and flexible data center network," in *Proc. ACM SIGCOMM Conf. Data Commun.*, 2009, pp. 51–62.

[52] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *Proc. Usenix Symp. Netw. Syst. Des. Implementation*, 2010, pp. 281–296.

[53] M. Mitzenmacher, "How useful is old information?" *IEEE Trans. Parallel Distrib. Syst.*, vol. 11, no. 1, pp. 6–20, Jan. 2000.

[54] A. W. Richa, M. Mitzenmacher, and R. Sitaraman, "The power of two random choices: A survey of techniques and results," *Combinatorial Optimization*, vol. 9, pp. 255–304, 2001.

[55] A. Rasouli, I. Kotseruba, and J. K. Tsotsos, "It is not all about size: On the role of data properties in pedestrian detection," in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 210–225.

[56] Rectrix, "Wind farms inspection," 2019. [Online]. Available: https://rectrixdrones.com/sectors/utilities-renewable-energy/wind-farms-inspection/, Accessed on: Jun. 2019

[57] DJI, "Wind turbine inspection," 2019. [Online]. Available: https://www.dji.com/zenmuse-z30/, Accessed on: Jun. 2019.

[58] A. Dogramaci and J. Surkis, "Evaluation of a heuristic for scheduling independent jobs on parallel identical processors," *Manage. Sci.*, vol. 25, no. 12, pp. 1208–1216, 1979.

[59] I. Ari, B. Hong, E. L. Miller, S. A. Brandt, and D. D. Long, "Managing flash crowds on the internet," in *Proc. 11th IEEE/ACM Int. Symp. Model. Anal. Simul. Comput. Telecommun. Syst.*, 2003, pp. 246–249.

[60] M. Mao and M. Humphrey, "Auto-scaling to minimize cost and meet application deadlines in cloud workflows," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2011, pp. 1–12.

[61] E. Crawley, R. Nair, B. Rajagopalan, and H. Sandick, "A framework for QoS-based routing in the internet," RFC 2386, Aug. 1998.

[62] M. J. Gonzalez, "Deterministic processor scheduling," *ACM Comput. Surveys*, vol. 9, no. 3, pp. 173–204, 1977.

[63] W. Bai, L. Chen, K. Chen, D. Han, C. Tian, and H. Wang, "Information-agnostic flow scheduling for commodity data centers," in *Proc. 12th USENIX Symp. Netw. Syst. Des. Implementation*, 2015, pp. 455–468.

[64] S. Zhang, Z. Qian, H. Wu, and S. Lu, "Efficient data center flow scheduling without starvation using expansion ratio," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 11, pp. 3157–3170, Nov. 2017.

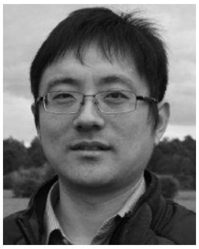
[65] M. Chowdhury and I. Stoica, "Coflow: A networking abstraction for cluster applications," in *Proc. 11th ACM Workshop Hot Topics Netw.*, 2012, pp. 31–36.

[66] M. Chowdhury, Y. Zhong, and I. Stoica, "Efficient coflow scheduling with varys," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 443–454, 2014.

[67] H. Zhang, L. Chen, B. Yi, K. Chen, M. Chowdhury, and Y. Geng, "Coda: Toward automatically identifying and scheduling coflows in the dark," in *Proc. ACM SIGCOMM Conf.*, 2016, pp. 160–173.

[68] S. Luo, H. Yu, Y. Zhao, S. Wang, S. Yu, and L. Li, "Towards practical and near-optimal coflow scheduling for data center networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 11, pp. 3366–3380, Nov. 2016.

[69] H. Jahanjou, E. Kantor, and R. Rajaraman, "Asymptotically optimal approximation algorithms for coflow scheduling," in *Proc. 29th ACM Symp. Parallelism Algorithms Architectures*, 2017, pp. 45–54.



Ke Xu (M'02-SM'09) received the PhD degree from the Department of Computer Science and Technology, Tsinghua University, where he serves as full professor. He serves as associate editor of *IEEE Internet of Things Journal* and has guest edited several special issues in IEEE and *Springer Journals*. His research interests include next generation Internet, P2P systems, Internet of Things, network virtualization, and network economics. He is a member of ACM. He is a senior member of the IEEE.

Liang Lv received the BS and MS degrees from the School of Computer, National University of Defense Technology, Hunan, China, in 2008 and 2010, respectively. He is working toward the the PhD degree at the Department of Computer Science and Technology, Tsinghua University. His research interests include next generation Internet, network virtualization and resource management in large-scale distributed systems.

Tong Li received the BS degree in computer science from Wuhan University, in 2012, and the PhD degree in computer science from Tsinghua University, in 2017. He held a visiting researcher position with the School of Computer Science & Electronic Engineering of University of Essex. He is now working as a senior researcher in Protocol Research Lab of Huawei. His research interests include cloud/edge computing, network protocol, measurement, and network science.

Meng Shen (M'14) received the BEng degree in computer science from Shandong University, Jinan, China, in 2009, and the PhD degree in computer science from Tsinghua University, Beijing, China, in 2014. He is currently an associate professor with the Beijing Institute of Technology, Beijing, China. His research interests include network congestion control, traffic engineering and network virtualization. He is a member of the IEEE.

Haiyang Wang received the PhD degree from Simon Fraser University, Burnaby (Metro-Vancouver), British Columbia, Canada, in 2013. He is currently an associate professor with the Department of Computer Science, University of Minnesota Duluth. His research interests include cloud computing, peer-to-peer networking, social networking, big data and multimedia communications.

Kun Yang received the PhD degree from the Department of Electronic & Electrical Engineering, University College London (UCL), United Kingdom. He is currently a chair professor with the School of Computer Science & Electronic Engineering, University of Essex, leading the Network Convergence Laboratory (NCL), United Kingdom. Before joining in University of Essex at 2003, he worked at UCL on several European Union (EU) research projects for several years. His main research interests include wireless networks and communications, data and energy integrated networks, computation-communication cooperation. He manages research projects funded by various sources such as UK EPSRC, EU FP7/H2020 and industries. He has published more than 150 journal papers. He serves on the editorial boards of both IEEE and non-IEEE journals. He is a senior member of the IEEE (since 2008) and a fellow of IET (since 2009).