The Value-of-Information in Matching with Queues

Longbo Huang
longbohuang@tsinghua.edu.cn
IIS, Tsinghua University

Abstract—We consider the problem of optimal matching with queues in dynamic systems and investigate the value-of-information. In such systems, operators match tasks and resources stored in queues, with the objective of maximizing the system utility of the matching reward profile, minus the average matching cost. This problem appears in many practical systems and the main challenges are the no-underflow constraints, and the lack of matching-reward information and system dynamics statistics. We develop two online matching algorithms: Learning-aided Reward optimAl Matching (LRAM) and Dual-LRAM (DRAM) to effectively resolve both challenges. Both algorithms are equipped with a learning module for estimating the matching-reward information, while DRAM incorporates an additional module for learning the system dynamics. We show that both algorithms achieve an $O(\epsilon + \delta_e)$ close-to-optimal utility performance for any $\epsilon > 0$, while DRAM achieves a faster convergence speed and a better delay compared to LRAM, i.e., $O(\delta_x / \epsilon + \log(1/\epsilon)^2)$ delay and $O(\delta_x / \epsilon)$ convergence under DRAM compared to $O(1/\epsilon)$ delay and convergence under LRAM ($\delta_x$ and $\delta_e$ are maximum estimation errors for reward and system dynamics). Our results show that information of different system components can play very different roles in algorithm performance and provide a novel way for designing joint learning-control algorithms.

I. INTRODUCTION

Matching is a fundamental problem that appears in resource allocation in various systems across different areas. For instance, network switch scheduling [1], online advertising [2], crowdsourcing [3], ride sharing [4], cloud computing [5], and inventory control [6]. As a result, efficient matching algorithms are of great importance to system control.

In this paper, we study the problem of optimal matching with queues in a dynamic environment with unknown matching reward statistics. Specifically, we consider a system consisting of a set of task queues and a set of resource queues, which store different types of workload and different types of resources that come into the system according to some random processes. At every time, the system operator decides how to match the resources to the pending workload. Each matching incurs a cost that depends on the resource allocated and random factors in the system, e.g., changing channel conditions in a downlink system, time-varying prices in inventory control, or fluctuating payment requirements in crowdsourcing. On the other hand, the matching also generates a reward, which is random with an unknown distribution determined by the amount of tasks resolved and the system condition. The objective is to design a matching strategy that carefully manages resources and tasks, so as to achieve optimal system utility, which is a function of the achieved reward profile, subject to the constraint that all tasks are fulfilled timely.

This is a general problem and models the aforementioned application scenarios. However, it is non-trivial to solve. First, the system utility is a function of the matching reward, which means that it is affected by when and how much resource is actually matched to the tasks and is only indirectly related to the traffic rates. This differs from traditional flow utility optimization problems [7], [8], and requires both careful admission control to avoid instability and appropriate matching to achieve good utility. Second, since each matching action is rewarded based on the actual amount of tasks resolved, the matching scheme must ensure that there are nonzero tasks and nonzero resources in the queues, i.e., no-underflow. This constraint is complex and is mostly tackled with dynamic programming, which can have high computational complexity.

Third, the system is dynamic and the statistics of system conditions and reward functions are unknown beforehand. This requires that the matching scheme can efficiently learn the sufficient statistics of the randomness and adapt to the changing environment.

In addition to resolving the above challenges, we also try to investigate the value-of-information in such matching-with-queue systems, by explicitly considering the impact of information on algorithm performance. Existing works on stochastic system control either focus on systems with perfect a-priori information, e.g., [9], [10], or rely on stochastic approximation techniques that do not require such information, e.g., [11], [12]. While the proposed solutions are effective, they do not capture how information affect algorithm design and performance, and do not provide interfaces for integrating the fast-developing “data science” tools, e.g., data collecting methods and machine learning algorithms, [13], [14], into system control.

To provide a rigorous quantification of the value-of-information, we first introduce an abstract notion of a learning module, which represents a general information learning algorithm and features a learning accuracy level $\delta$ (maximum error), a learning time $T_\delta$, and the probability of learning accuracy guarantee $P_\delta$. We then design two online matching algorithms: Learning-aided Reward optimAl Matching (LRAM) and Dual-LRAM (DRAM). LRAM utilizes a single $(T_\delta, \delta, P_\delta)$ learning module for estimating the reward statistics and achieves an $O(\epsilon + \delta_e)$ system utility, for any $\epsilon > 0$, while ensuring an $O(1/\epsilon)$ delay bound and an $O(1/\epsilon)$ algorithm con-

This work was supported in part by the National Basic Research Program of China Grant 2011CBA00300, 2011CBA00301, the National Natural Science Foundation of China Grant 61361136003, 61303195, Tsinghua Initiative Research Grant, Microsoft Research Asia Collaborative Research Award, and the China Youth 1000-talent Grant.
vergence time, defined to be the time taken for the algorithm to enter the optimal control state. DRAM incorporates an additional \((T_{\delta_x},\delta_{\pi},P_{\delta_x})\) learning module for the random system state distribution and guarantees a similar \(O(\epsilon + \delta_{\pi})\) system utility. Moreover, DRAM is able to achieve an \(O(\delta_{x}/\epsilon + \log(1/\epsilon)^2)\) delay bound and an \(O(\delta_{x}/\epsilon)\) algorithm convergence time, which can be significantly faster compared to LRAM. Our formulation and algorithms share similarity with those in the multi-armed bandit literature, e.g., [15], [16]. However, our results differ in that (i) queue underflow affects reward, and (ii) queuing delay is explicitly considered and analyzed.

Our results reveal an interesting fact that the reward information largely determines the utility performance, while the system dynamics information greatly affects delay and algorithm convergence. This indicates that information of different system components can have different impacts on algorithm performance, and may require different learning power for achieving a desired goal. We also explicitly quantify the values of different system information. Closest to our paper are recent works [17] and [18], which consider joint learning and control. However, [17] and [18] both assume prior knowledge of the action-reward functions and consider a specific learning approach, whereas our framework allows more general learning methods, and handles the no-underflow constraints and unknown rewards. Moreover, the analysis here is different from those in [17] and [18], due to the potential estimation errors in learning the reward functions.

We summarize the main contributions as follows:

1) We propose a matching queueing system model, which models general resource-task matching problems in stochastic systems. To explicitly quantify the value of information in such systems, we introduce an abstract notion of a \((T_{\delta_x},\delta_{\pi},P_{\delta_x})\)-learning module that captures key characteristics of general learning algorithms and provides interfaces for bringing the information learning aspect into system control.

2) We design two learning-aided matching algorithms LRAM and DRAM. We show that with a single \((T_{\delta_x},\delta_{\pi},P_{\delta_x})\)-module for learning reward statistics, LRAM achieves an \(O(\epsilon + \delta_{\pi})\) utility, while ensuring an \(O(1/\epsilon)\) delay bound and an \(O(1/\epsilon)\) algorithm convergence time. DRAM adopts an additional \((T_{\delta_x},\delta_{\pi},P_{\delta_x})\)-module for learning the system state distribution, and guarantees a similar \(O(\epsilon + \delta_{\pi})\) system utility, while achieving an \(O(\delta_{x}/\epsilon + \log(1/\epsilon)^2)\) delay and an \(O(\delta_{x}/\epsilon)\) convergence time. We also construct two \((O(1/c^2),O(c/\epsilon^2),1-O(c^{\log(1/\epsilon)})\) online learning modules based on sampling \((c > 0)\). Combining them with DRAM, one achieves a fast \(O(1/\epsilon^{1-1/c^2}+1/c^2)\) convergence time with \(c < 1\) (existing algorithms require \(\Theta(1/\epsilon)\)).

3) Our algorithm design approach provides a low-complexity way to tackle multiple simultaneous no-underflow constraints in systems and optimize utilities that are not purely defined on flow rates. The development of DRAM also demonstrates how general learning algorithms can be combined with queue-based control (stochastic approximation) to achieve superior delay and accelerate algorithm convergence speed.

The rest of the paper is organized as follows. We first list a few motivating examples in Section II. We then present the matching system model in Section III. The algorithm design approach and the two algorithms LRAM and DRAM are presented in Section IV. Analysis is carried out in Section VII and simulation results are presented in Section VIII. We then conclude the paper in Section IX.

**II. Motivating Examples**

**Crowdsourcing:** In a crowdsourcing application, e.g., crowdsourcing query search [3] or ride-sharing [4], tasks of different types (task) arrive at the server and are assigned to workers (resource). The workers then carry out the tasks. Depending on workers’ qualifications, types of jobs, and the instantaneous system condition (state), e.g., whether a query requestor is in a hurry due to weather, requestors receive certain rewards, e.g., satisfaction, and workers receive payments.

The objective of the system is to design a matching scheme, so as to maximize system utility, which is a function of the achieved requestor reward profile.

**Energy Harvesting Networks:** In an energy harvesting network, e.g., [19], [20], nodes are responsible for transmitting data (task) and can harvest energy (resource) from the environment. At every time, each node decides how much energy to allocate for transmission and determines traffic scheduling. Depending on the time-varying channel condition (state), amount of energy enables certain processing results.

The objective is to design a joint energy management and scheduling algorithm, so as to maximize traffic utility and ensure that no energy outage happens.

**Online Advertisement:** In an online advertising system, [2], [21], advertisers deposit money (task) into their accounts at the advertising platform. Queries (resource) for different keywords arrive in the system and the server decides which advertiser’s ads to show, based on their relevance to the keywords and the available budget of the advertisers. Depending on the chosen ad and the user’s condition (state), e.g., location or mood, a business transaction may take place. The goal of the system is to design an ad matching scheme, so as to maximize system utility, which is a function of the average income profile from advertisers.

**Cloud Computing:** In a cloud computing platform, e.g., [5], computing resources (resource), e.g., CPU and memory, are assigned to virtual machine instances (task) for processing arriving job requests. The quality of experience of a requestor depends on the job completion quality, which is affected by system conditions such as background task level (state) and user status. The objective here is to design a resource allocation policy, such that the overall quality of service is maximized.

In all these examples, the underlying problem is indeed matching with queues. Below, we present the general model.

**III. System Model**

We consider a discrete-time system shown in Fig. 1. In this system, there are two sets of queues, task queues and resource queues. The rest of the paper is organized as follows. We first list a few motivating examples in Section II. We then present the matching system model in Section III. The algorithm design approach and the two algorithms LRAM and DRAM are presented in Section IV. Analysis is carried out in Section VII and simulation results are presented in Section VIII. We then conclude the paper in Section IX.
The matching queueing system.

queues, and a central server (called operator below), which coordinates resource allocation and scheduling in the system. Time is divided into unit-size slots, i.e., \( t \in \{0, 1, \ldots\} \).

A. Tasks and Resources

The task queues store jobs that come into the system and are waiting to be served by the server. We assume there are \( N \) types of tasks and denote the set of task queues by \( Q = \{Q_1, \ldots, Q_N\} \). We use \( A_n(t) \) to denote the amount of new tasks arriving at \( Q_n \) at time \( t \) and assume that \( 0 \leq A_n(t) \leq A_{\text{max}} \). We then define the arrival vector \( A(t) = (A_1(t), \ldots, A_N(t)) \). In many systems, arrivals to the system may not always be admitted due to congestion control, e.g., when all servers are busy. We model this by using 0 \( \leq R_n(t) \leq A_n(t) \) to denote the actual admitted traffic to \( Q_n \) at time \( t \). We then use \( Q_n(t) \) to denote the amount of tasks stored at \( Q_n \) at time \( t \) and denote \( Q(t) = (Q_1(t), \ldots, Q_N(t)) \) the task queue vector.

The resource queues, on the other hand, hold the resources the system collects over time. There are \( M \) types of system resources and we denote the resource queues by \( H = \{H_1, \ldots, H_M\} \). We similarly let \( e_{mn}(t) \) be the amount of resources arriving at \( H_m \) with \( 0 \leq e_{mn}(t) \leq h_{\text{max}} \). We also use \( H_m(t) \) to denote the amount of resource \( m \) the system currently holds and denote \( H(t) = (H_1(t), \ldots, H_M(t)) \) the resource queue vector.

In many systems, it is feasible (and sometimes necessary) to control the amount of resources in the system, e.g., to avoid too many workers waiting in crowdsourcing. We model this decision by using \( h_{mn}(t) \in [0, e_{mn}(t)] \) to denote the actual amount of type \( m \) resource admitted. For now, it is also convenient to temporarily assume that the queues are all of unlimited sizes. We will later show that our algorithms ensure that finite buffer sizes are sufficient.

B. System State and Resource allocation

We assume that the system condition varies over time, e.g., channel conditions in a downlink system, or the expected happiness measures of human users in a crowdsourcing system. We call this condition the system state and model it by a random state variable \( \omega(t) \). Note that \( \omega(t) \) represents the aggregate system condition. For instance, in an \( M \)-user downlink system, \( \omega(t) \) can represent a vector \( (CH_1(t), \ldots, CH_M(t)) \) where \( CH_m(t) \) denotes the channel condition of user \( m \) at time \( t \).

Denote \( z(t) = \omega(t) \). In this paper, we assume that \( z(t) \) is i.i.d. and takes values in \( Z = \{z_1, \ldots, z_K\} \). We then denote \( \pi_k = \Pr(z(t) = z_k) \). Note that this allows arbitrary dependency among \( A(t), e(t), \) and \( \omega(t) \).

At every time \( t \), the system operator determines the amount of resource to allocate to serving each queue. We denote this decision by a matching matrix \( b(t) = (b_{mn}(t), m, n) \), where \( b_{mn}(t) \) denotes the type \( m \) resource allocated to queue \( n \). When \( z(t) = z_k \), \( b(t) \) takes values from a finite discrete set \( B_k \subset \mathbb{R}_{+}^N \). We define \( b_{\text{max}} = \max_{b_k \in B_k} \|b_k\|_{\infty} \) the maximum amount of resource allocated to any queue at any time. Here we define \( \|x\|_{\infty} = \max_{j} |x_{ij}| \) for any matrix \( x \). It is clear that at any time \( t \), we must have:

\[
\sum_{n} b_{mn}(t) \leq H_{mn}(t), \quad \forall m. \tag{1}
\]

This is because one cannot spend more resource than what is available. In the following, we call (1) the no-underflow constraint. Depending on the system state and resource allocation decision, each task queue gets a service rate \( \mu_{n}(t) = \mu_{n}(z(t), b(t)) \). We assume \( \mu_{n}(z, b) \in [0, \mu_{\text{max}}] \) for all \( z \) and \( b \) and that \( \{\mu_{n}(z, b)\}_{n=1}^{N} \) are known to the operator. Also, they satisfy that \( \mu_{n}(z, 0) = 0 \) for all \( z \), and if \( \mu_{n}(z, b) > 0 \),

\[
\mu_{n}(z, b) \geq \beta_{\mu_{n}^{\text{min}}} \min_{b_{mn} > 0} b_{mn},
\]

for some \( \beta_{\mu_{n}^{\text{min}}} > 0 \). Moreover, if two vectors \( b \) and \( b' \) are such that \( b' \) is obtained by setting \( b_{mn} \) in \( b \) to zero, then,

\[
\mu_{n}(z, b) \leq \mu_{n}(z, b') + \beta_{\mu_{n}^{\text{min}}} b_{mn}, \quad \forall n. \tag{3}
\]

Note that (2) and (3) are not restrictive. They simply require that nonzero resource is needed for getting a positive service rate, and that a positive rate is upper and lower bounded by linear functions of the resources allocated.

C. Matching Cost and Reward

In every time slot, due to resource expenditure, there is a matching cost associated with the resource allocation decision. We model this by denoting \( c(t) = c(z(t), b(t)) \) the cost for choosing the resource vector \( b(t) \). This cost can represent, e.g., cost for purchasing raw materials in inventory control, or payments to workers in a crowdsourcing application. One example is \( c(z(t), b(t)) = \sum_{mn} e_{mn}(z(t)) b_{mn}(t) \), where \( e_{mn}(z(t)) \) denotes the per-unit resource price for type \( m \) resource under state \( z(t) \). We assume that \( c(z, b) \in [0, c_{\text{max}}] \) for all time and it is known to the system operator. Also, if \( 0 \leq b_1 \leq b_2 \) (entrywise-less), then \( c(z, b_1) \leq c(z, b_2) \).

Every time a matching is completed, the operator collects a matching reward, e.g., a customer conversion due to an ad, or user satisfaction due to job completion. We model this by denoting the reward collected at time \( t \) from type \( n \) tasks by \( \kappa_{n}(t) \). We assume that for any time sequence \( \{t_j, j = 1, 2, \ldots\} \) such that \( z(t_j) = z \), \( b(t_j) = b \) and \( Q(t) \), the sequence \( \{\kappa_{n}(t_j) \in [0, r_{\max}], j = 1, 2, \ldots\} \) consists of i.i.d. random variables, whose mean is determined by the reward function \( r_{\kappa}(z(t), \kappa_{n}(t)) = \mathbb{E}[\kappa_{n}(t) | z(t), b(t), Q(t)] \). \(^4\) Here \( \hat{\mu}_{n}(t) = \min\{Q_{n}(t), \mu_{n}(t)\} \) denotes the actual amount of tasks completed. We assume that \( r_{\kappa}(z(t), \mu_{n}(t)) \) satisfies:

\[
r_{\kappa}(z(t), \mu) \leq r_{\kappa}(z(t), \mu'), \quad \text{if } \mu \leq \mu', \tag{4}
\]

\(^2\)This assumption is made to simplify the learning algorithm description (Section VI). Our results can likely be extended to the case when \( \{B_k, k\} \) are general compact sets in \( \mathbb{R}_{+}^{N} \).

\(^3\)\( \|x\| \) is a generic symbol that represents a matrix.

\(^4\)The i.i.d. assumption is introduced to facilitate analysis. More general cases can also be considered but will be more involved.
and denote \( r = (r_1, \ldots, r_N) \) the reward matrix, where \( r_n = (r_n(z, \mu_n(z, b_2)), z \in Z, b_2 \in B_2) \) is a \( K \sum |B_k| \)-dimension vector denoting the action-reward correspondence for queue \( n \). Since each \( B_2 \) is finite, \( r \) is also finite.

Different from existing works, e.g., [21, 22], we do not assume any prior knowledge of the function \( r_n(z(t), \mu) \). This is common in practice. For example, in crowdsourcing applications, it is often unknown a-priori how qualified a worker is for a certain type of tasks; or in online advertising, one often does not know the conversion probabilities beforehand.

D. Queueing

From the above, we see that the queue vectors \( Q(t) \) and \( H(t) \) evolve according to:
\[
Q_n(t + 1) = \max\{Q_n(t) - \mu_n(t, 0) + R_n(t), \forall n, \quad (5)\]
\[
H_m(t + 1) = H_m(t) - \sum b_m(t) + h_m(t), \forall m, \quad (6)
\]
with \( Q(t) = 0 \) and \( H(t) = 0 \). Notice that there is no \( \max[\cdot, \cdot] \) operator in (6). This is due to the no-underflow constraint (1). In our paper, we say that a queue vector process \( x(t) \in \mathbb{R}^d_+ \) is stable if it satisfies:
\[
x_{av} \triangleq \lim_{t \to \infty} \frac{1}{t} \sum_{\tau=0}^{t-1} \sum_{n=1}^{d} \mathbb{E}\{x_n(\tau)\} < \infty. \quad (7)
\]

E. Utility Optimization

The system’s utility is determined by a function of the average matching reward profile. Specifically, define \( \tau_n \triangleq \lim_{t \to \infty} \frac{1}{t} \sum_{\tau=0}^{t-1} \mathbb{E}\{r_n(\tau)\} \). The system utility is given by:
\[
U_{\text{total}}(\Pi) \triangleq \sum_{n} U_n(\tau_n). \quad (8)
\]
Here each \( U_n(r) \) is an increasing concave function with \( U_n(0) = 0 \) and \( U'(0) < \infty \). We denote \( \beta \triangleq \max_{n,r} U_n'(r) \) the maximum first derivative of the utility functions. We also define the following system cost due to resource expenditure:
\[
C_{\text{total}} \triangleq \lim_{t \to \infty} \frac{1}{t} \sum_{\tau=0}^{t-1} \mathbb{E}\{c(\tau)\}. \quad (9)
\]

We say that making a matching algorithm \( \Pi \) is feasible if for all time \( t \), it selects \( 0 \leq R(t) \leq A(t), 0 \leq h(t) \leq e(t) \), and \( b(t) \in B_2(t) \), and it ensures constraint (1) for all time. Our objective is to design a feasible policy \( \Pi \), so as to:
\[
\max_{\Pi} f_{av} \triangleq U_{\text{total}}(\Pi) - C_{\text{total}} \quad (10)
\]
\[
\text{s.t.} \quad Q_{av} < \infty, \quad H_{av} < \infty. \quad (11)
\]
Here the superscript \( II \) indicates that the utility and cost are functions of the control policy \( II \). We denote the optimal solution value as \( f_{av}^{\Pi} \). Here the queue stability constraints are to ensure that the tasks and resources do not stay in the queue forever. This is important in many cases. For instance, in an energy harvesting network, it is important to ensure timely packet delivery, or in a crowdsourcing system, it is desirable to keep the worker waiting time short.

F. Discussion on the Model

Due to the general matching reward function, our problem is different from a flow utility maximization problem, e.g., [7], which is a special case when \( r_n(t) = \mu_n(t) \). By tuning parameters of the model, our model can model all the examples in Section II. For example, by choosing \( U_{\text{total}} = \sum_{n} \alpha_n R_n \), the system models the revenue maximization problem in online advertisement systems. By choosing \( \mu_n(t) = 1_{[b_1(t) > b_\text{min}]} 1_{[b_2(t) > b_\text{min}]} \), our model can represent a cloud computing system, where a computing task requires two types of resources. Note that in our model we also require the resource queues to be stable in our model for the following two reasons. (i) In certain applications, e.g., crowdsourcing and taxi ride-sharing, the available resource consists of human users who will perform the tasks. In this case, it is important to ensure that workers do not wait indefinitely in the system for tasks, (ii) Imposing the stability constraint helps develop algorithms that only require finite size buffers for holding resources. This is in fact an important feature of our framework. In practice, although it is always beneficial to accumulate more resources, the amount is naturally limited by the finite buffers in the system, and it is often challenging to design algorithms for such systems.

Problem (10) is very challenging. First of all, the no-underflow constraint (1) requires a careful selection of control actions, because actions in a slot can affect action feasibility in later slots. Problems of this kind are often tackled with dynamic programming, whose computational complexity can be extremely high when the action space is large. Secondly, the reward function \( r_n(z(t), \mu(t)) \) is unknown and is dependent on \( Q_n(t) \). This makes the problem different from existing utility maximization works, e.g., [7], [8]. Thirdly, due to the more and more stringent user requirements on service quality, it is more desirable to ensure small queuing delay.

Below, we first present an algorithm that assumes full prior knowledge of all system components. This ideal algorithm and its derivation serve as the foundation for the general algorithms. Then, we introduce a general learning model, and present two learning-aided algorithms that explicitly learn and incorporate system information into control. For convenience, we summarize the notations in the paper in Table I.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>( Q_n, Q_n(t) )</td>
<td>Task queue ( n ) and its size at time ( t )</td>
</tr>
<tr>
<td>( H_m, H_m(t) )</td>
<td>Resource queue ( n ) and its size at time ( t )</td>
</tr>
<tr>
<td>( \omega(t) )</td>
<td>System state</td>
</tr>
<tr>
<td>( \mu(t) )</td>
<td>Aggregate system condition</td>
</tr>
<tr>
<td>( \beta(t) )</td>
<td>System priority</td>
</tr>
<tr>
<td>( R(t) )</td>
<td>Matching reward function</td>
</tr>
<tr>
<td>( b(t) )</td>
<td>Matched task reward</td>
</tr>
<tr>
<td>( y(t) )</td>
<td>System configuration</td>
</tr>
<tr>
<td>( \pi_n(t) )</td>
<td>Expected matching reward for type ( n ) tasks</td>
</tr>
<tr>
<td>( \delta_n(t) )</td>
<td>Expected matching reward for type ( n ) tasks</td>
</tr>
<tr>
<td>( \delta(t) )</td>
<td>Matrix estimation error</td>
</tr>
</tbody>
</table>

TABLE I

IV. MATCHING WITH FULL REWARD INFORMATION

In this section, we will first present an ideal algorithm, which assumes full knowledge of the reward functions.
r_n(z(t), \mu_t) and will serve as a basic building block. Even in this case, we will see that the problem is highly nontrivial due to the existence of the no-underflow constraint (1) and the dependency of r_n on Q(t).

To start, we first introduce an auxiliary variable \( \gamma_n(t) \in [0, r_{\text{max}}] \) and create for each n a deficit queue \( d_n(t) \) that evolves as follows:

\[
d_n(t + 1) = \max[d_n(t) - \kappa_n(t), 0] + \gamma_n(t),
\]

with \( d(0) = 0 \). Note that the input into \( d_n(t) \) is \( \kappa_n(t) \) instead of \( r_n(t) \). The deficit queue \( d_n(t) \) measures how much the actual reward profile is currently lagging behind the target value (due to randomness). Hence, stabilizing the deficit queues implies that the average reward collected from type \( n \) tasks is no smaller than the target value.

We first present our algorithm assuming full knowledge of the expected reward vector \( r(t) \).

**Reward Optimal Matching (RAM):** At every time t, observe \( z(t) \) and \( y(t) \). Do:

1. **Quota:** For each \( n \), choose \( \gamma_n(t) \) by solving:
   \[
   \max : VU_n(\gamma_n(t)) - d_n(t)\gamma_n(t), \text{ s.t. } 0 \leq \gamma_n(t) \leq r_{\text{max}}. \tag{13}
   \]
   2. **Admission:** For each \( n \), if \( Q_n(t) < \theta_1 \), let \( R_n(t) = A_n(t) \); otherwise \( R_n(t) = 0 \). Similarly, for each \( m \), if \( H_m(t) < \theta_2 \), let \( h_m(t) = e_m(t) \); otherwise \( h_m(t) = 0 \).
   3. **Resource:** Choose the resource allocation vector \( b(t) \) by solving:
   \[
   \min : \Psi_r(b) \triangleq Vc(z(t), b) - \sum_m (H_m(t) - \theta_2) \sum_n b_{mn} \tag{14}
   - \sum_n (Q_n(t) - \theta_1) \mu_n(z(t), b)
   - \sum_n d_n(t) r_n(z(t), \mu_n(z(t), b))
   \text{ s.t. } b \in B_{z(t)} \text{ (Constraint (1))}
   \]
   4. **Queueing:** The queues \( Q(t) \), \( H(t) \), and \( d(t) \) evolve according to (5), (6), and (12), respectively. \( \square \)

RAM takes as input the state \( z(t) \) and the queue vector \( y(t) \), and outputs corresponding actions \( \gamma(t), R(t), \) and \( b(t) \). In the algorithm, \( V \) is a parameter introduced for controlling the tradeoff between system utility and service delay (explained later). Note that in (14) we have used \( r_n(z(t), \mu_n(t)) \) instead of \( r_n(z(t), \mu_n(t)) \). We will see in our later analysis that our algorithm automatically guarantees \( \tilde{\mu}_n(t) = \mu_n(t) \). This is very useful, otherwise the algorithm performance will be very hard to analyze. We also emphasize here that the introduction of \( \theta_1 \) and \( \theta_2 \) are important. It can be seen in the admission step that if \( \theta_1 = \theta_2 = 0 \), no task or resource will be admitted at the first place and the algorithm will not even proceed! From the algorithm description, we see how the \( r_n(z(t), \mu_n(t)) \) functions come into the picture. The explicit form of (14) also enables an exact analysis about how estimation errors in \( r_n(z(t), \mu_n(t)) \) affect performance.

We now provide the mathematical derivation behind RAM. To do so, we denote

\[
f_s(t) \triangleq \sum_n U_n(\gamma_n(t)) - c(t)\tag{16}
\]

the target instantaneous system utility minus cost and denote \( y(t) \triangleq (Q(t), H(t), d(t)) \). We then define a Lyapunov function as follows:

\[
L(t) = \frac{1}{2}\|Q(t) - \theta_1\|^2 + \frac{1}{2}\|H(t) - \theta_2\|^2 + \frac{1}{2}\|d(t)\|^2, \tag{17}
\]

where \( \| \cdot \| \) is the euclidean norm and \( \theta_1 = \theta_1 \cdot 1_N \) and \( \theta_2 = \theta_2 \cdot 1_M \) with \( 1_k \in \mathbb{R}^k \) being the vector with all components being 1, and \( \theta_1 \) and \( \theta_2 \) are constants that will be specified later. We define the one-slot utility-based conditional Lyapunov drift

\[
\Delta V(t) \triangleq \mathbb{E}(L(t+1) - L(t) - V_f(t) \mid y(t)).
\]

Using queueing dynamics (5), (6), and (12), we obtain the following lemma, in which \( V \geq 1 \) is the tunable parameter as in the RAM algorithm.

**Lemma 1.** Under any feasible policy, we have:

\[
\Delta V(t) \leq G - V \sum_n \mathbb{E}\{U_n(\gamma_n(t)) - d_n(t)\gamma_n(t) \mid y(t)\} \tag{18}
+ \sum_n (Q_n(t) - \theta_1) \mathbb{E}\{R_n(t) \mid y(t)\}
+ \sum_m (H_m(t) - \theta_2) \mathbb{E}\{h_m(t) \mid y(t)\}
+ \mathbb{E}\{Vc(t) - \sum_m (H_m(t) - \theta_2) \sum_n b_{mn} \}
- \sum_n (Q_n(t) - \theta_1) \mu_n(t) - \sum_n d_n(t) r_n(t) \mid y(t)\}.
\]

Here \( G \triangleq N(\Lambda_{\text{max}}^2 + \mu_{\text{max}}^2 + 2r_{\text{max}}^2) + Mh_{\text{max}}^2 + MN^2b_{\text{max}}^2 \) does not depend on \( V \), and the expectations are taken over the randomness in the system as well as in the policy. \( \square \)

**Proof.** See Appendix A.

**V. LEARNING MODULE**

From the previous section, we see that if full reward information is available beforehand, RAM can be used to achieve optimal utility. In the following, we turn to consider the case when such information is not given and must be learned. To investigate the impact of learning on algorithm design and performance, we first define the notion of learning capability. Specifically, for any general matrix \( W \) and a learning algorithm \( \Gamma \) that outputs an estimation \( \bar{W} \), we denote its maximum estimation error by:

\[
\delta_m \triangleq \|\bar{W} - W\|_{\infty}. \tag{19}
\]

Recall that \( \|x\|_{\infty} \triangleq \max_{ij} |x_{ij}| \). Then, the formal definition of a learning module is as follows.

**Definition 1.** An algorithm \( \Gamma \) is called a \( (T_3, P_3, \delta) \)-learning module, if (i) it completes learning in \( T_3 \) time, (ii) it guarantees \( P_3 \{\delta_m < \delta\} \geq P_3 \), and (iii) for any \( T \geq 0 \), \( P_3 \) does not decrease if the algorithm is run for \( T_3 + T \) time. \( \square \)

Here \( \delta \) specifies the estimation accuracy guarantee, and \( T_3 \) can be both random or deterministic depending on the termination rules. This definition is general and captures key features of learning algorithms. With this definition, having perfect knowledge at the beginning can be viewed as having an \((0, 1, 0)\)-learning module. The introduction of a learning module is to enable an explicit separation between the learning component and the control component. Doing so provides new guidelines for developing a modular design procedure for learning-aided control algorithms. We will also show how parameters of the learning module can be incorporated into system control and how they affect system performance.
As concrete examples, we now describe two sampling-based learning modules for estimating $r$ and $\pi$. We first describe a threshold-based sampling module for estimating $r$. In the module, we use $s(z, b, t)$ to denote the number of times the pair $(z, b)$ is sampled, i.e., adopt $b$ when $z(t) = z$, up to time $t$. We also denote $s_{\min}(t) = \min_{z, b} s(z, b, t)$.

**Threshold-Based Sampling TBS($s_{th}$):** Every time $t$, sample the resource allocation vector $b^* \in \arg \min_b s(z(t), b, t)$ until $s_{\min}(t) \geq s_{th}$. If terminate at $T_{bs}$, output $\hat{r}_n(z, b) = \sum_{t=1}^{T_{bs}} 1[z(t) = b(t)]K_n(t)/s(z, b, T_{bs})$.

**Proof.** See Appendix B.

By choosing $s_{th} = T_{th} = V^c$, we can guarantee $\delta_r = \delta_\pi = c \log(V)/V^{c/2}$ with $P_\delta$ and $P_{\delta_\pi}$ being $1 - O(V^{-c/2})$.

**VI. OPTIMAL MATCHING WITH LEARNING**

We now consider the case when one does not have full reward information and present two algorithms that integrate learning methods for estimating $r$ and $\pi$.

**A. With Reward Information Learning**

We first present an optimal matching algorithm for general systems that do not possess perfect knowledge of $r$ and need to rely on some learning algorithms for estimation. In the algorithm, we use $\beta_r$ to denote the maximum “derivative” of the estimated $\hat{r}$ with respect to any $b_{mn}$. Specifically, we assume that if $b$ and $b'$ are such that $b'$ is obtained by setting one $b_{mn}$ in $b$ to zero. Then,

$$\hat{r}_n(z, z_n(b, b')) \leq \hat{r}_n(z, z_n(b, b')) + \beta_r b_{mn}, \forall n. \quad (20)$$

Since both $\mathcal{Z}$ and $\mathcal{B}_n$ are finite, we see that $\beta_r$ exists and is $\Theta(1)$ (possibly depends on $r$).

**Learning-aided Reward OptimAl Matching (LRAM):**

1) (Learning) Apply any $(T_{bs}, P_{\delta}, \delta_r)$-learning module $\Gamma_r$, Terminate at $t = T_{bs}$, and output $\hat{r}$.

2) (Matching) Set $Q(T_{sb} + 1) = 0$, $H(T_{sb} + 1) = 0$, and $d(T_{sb} + 1) = 0$. Choose $\theta_1$ and $\theta_2$ according to:

$$\theta_1 = (h_{max} + (V\beta + r_{max})\beta_r)/\beta_\mu + \mu_{max} \quad (21)$$

and

$$\theta_2 = (V\beta + r_{max})\beta_r + r_{max}\beta_\mu + Nb_{max}. \quad (22)$$

Run RAM with $\hat{r}$, i.e., replace $r_n(z(t), z_n(t), b)$ in RAM by $\hat{r}_n(z(t), z_n(t), b)$.

In LRAM, we explicitly separate the algorithm into two disjoint phases. This is chosen to facilitate presentation and analysis. Doing so does not change the order of the overall algorithm convergence time and performance. We can also transform LRAM and DRAM below into continuous-learning versions, e.g., [17], and update estimations from time to time, e.g., using sliding-window estimation or frame-based estimation. Lastly, it is worth noting that $\theta_1$ and $\theta_2$ can be computed beforehand easily. This is a feature useful for implementation.

**B. With System State Information Learning**

In the previous section, we describe how the estimated reward information $\hat{r}$ can be naturally integrated into a matching algorithm. Here we consider the case when a learning module is also applied to learning the statistics of the system state $z(t)$. Our result here generalizes the dual-learning approach proposed in [17] to handle underflow and to allow general learning methods.

To start, we define the following optimization problem:

$$\max : \Phi = V \sum_n U_n(\gamma_n) - Cost \quad (23)$$

s.t. $\gamma_n \leq r_n(\sum_k \pi_k r_n(z_k, \mu_n(z_k, b_k)))$ (24)

$$Cost = \sum_k \pi_k c(z_k, b_k) \quad (25)$$

$$\sum_k \pi_k P_n^k = \sum_k \pi_k b_k, \forall n \quad (26)$$

$$\sum_k \pi_k b_{mn}^k = \sum_k \pi_k n^b_{mh}, \forall m \quad (27)$$

$$0 \leq \gamma_n \leq r_{max}, b_k \in B_k \quad (28)$$

$$0 \leq P_k \leq A_k, 0 \leq h^k \leq e^k. \quad (29)$$

Problem (23) can intuitively be viewed as a way to solve our matching problem. The introduction of the nonlinear objective function. The equalities in (26) and (27) are due to fact that only tasks that are actually served generate reward and the no-underflow constraint (1). However, a scheme obtained by solving (23) may not be implementable due to (i) it ignores the no-underflow constraint, and (ii) it assumes that all resources allocated are fully utilized, i.e., using $\mu_n(z_k, b_k)$ in (24). We will also see later that, it may require a much larger learning time for such a scheme to have the right statistics for achieving a performance comparable to our schemes.

We now obtain the dual problem for (23) as follows.

$$\min : g(\alpha, \alpha^d, \alpha^h) \text{ s.t. } \alpha^d \geq 0, \alpha^d \in \mathbb{R}^N, \alpha^h \in \mathbb{R}^M, \quad (30)$$

where $g(\alpha, \alpha^d, \alpha^h) = \sum_k \pi_k g_k(\alpha, \alpha^d, \alpha^h)$ is the dual function, and $g_k(\alpha, \alpha^d, \alpha^h)$ is defined as:

$$g_k(\alpha, \alpha^d, \alpha^h) = \sup_{R, \gamma, b, h} \left\{ V \left[ \sum_n U_n(\gamma_n) - c(z_k, b_k) \right] \right\} \quad (31)$$

$$- \sum_n \alpha_n^d r_n(z_k, \mu_n(z_k, b)) - \gamma_n \quad (32)$$

Technically speaking, one has to solve a convexified version of the problem in order to find an optimal control policy for our problem.
Note here that the actual queues $Q(t)$, $H(t)$, and $d(t)$ still evolve according to (5), (6), and (12), respectively, and DRAM defines $Q(t)$, $H(t)$, and $d(t)$ on top of their values. The dropping step is introduced to ensure zero underflow, by giving up the service rates and reward at that particular slot. We will see in later analysis and simulation that such an event almost never happens and hence does not affect performance.

We note that both LRAM and DRAM are two-stage algorithms, which commonly appear in the multi-armed bandit literature, e.g., [15], [16]. However, our formulation and solutions are different in that (i) queue underflow affects reward, and (ii) queuing delay is explicitly considered and analyzed.

VII. PERFORMANCE ANALYSIS

In this section, we present the performance results of LRAM and DRAM. We start with some definitions and assumptions. For notation simplicity, we denote $\alpha = (\alpha^d, \alpha^q, \alpha^h)$ and write $\alpha - \theta = (\alpha^d - \theta^1, \alpha^q - \theta^1, \alpha^h - \theta^2)$. Also, to indicate the different distributions and reward functions used, we use $\hat{g}(\alpha)$ to denote the dual function when $r$ is replaced by $\hat{r}$ in (31) and the distribution is given by $\pi$. Then, we use $g^\pi(\alpha)$ and $\hat{g}^\pi(\alpha)$ to denote the dual function with distribution $\pi$ and $r$, and with $\hat{\pi}$ and $\hat{r}$, respectively.

A. Preliminaries

We define the following polyhedral system structure:

**Definition 2.** A system is polyhedral with parameter $\rho > 0$ if the dual function $g(\alpha)$ satisfies:

$$g(\alpha^*) \leq g(\alpha) - \rho \| \alpha^* - \alpha \|.$$  

Here $\alpha^*$ is an optimal solution of (30).  

The polyhedral structure typically appears in practical systems, especially when system action sets are discrete (see [23] for more discussions). Note that (38) holds for all $\mathbf{V}$ if it holds under any $\mathbf{V}$, in particular $\mathbf{V} = 1$.

In our analysis, we make the following assumptions.

**Assumption 1.** There exist constants $\epsilon_r, \epsilon_z = \Theta(1) > 0$ such that for any valid state distribution $\pi$ and reward statistics $\hat{r}$ with $\| \hat{\pi} - \pi \| \leq \epsilon_z$ and $\| \hat{r} - r \| \leq \epsilon_r$, there exist a set of actions $\{q^k \}_{k=1,\ldots,|\mathbf{Z}|}$, $\{R^k \}_{k=1,\ldots,\infty}$, $\{b^k \}_{k=1,\ldots,\infty}$, and $\{h^k \}_{k=1,\ldots,\infty}$, and variables $\lambda_k^i \geq 0$ with $\sum_i \lambda_k^i = 1$ for all $k$ (possibly depending on $\hat{\pi}$ and $\hat{r}$), such that:

$$\gamma_n - \sum_k \hat{\pi}_k \sum_i \lambda_k^i \hat{r}_n(z_k, \mu_n(z_k, b_k^i)) \leq -\eta_0,$$  

where $\eta_0 = \Theta(1) > 0$ is independent of $\hat{\pi}$ and $\hat{r}$, and that

$$\sum_k \hat{\pi}_k \lambda_k^i \hat{R}_n^{k,i} = \sum_k \hat{\pi}_k \lambda_k^i \hat{\mu}_n(z_k, b_k^i)$$  

$$\sum_k \hat{\pi}_k \lambda_k^i h_{n,m}^{k,i} < \sum_k \hat{\pi}_k \lambda_k^i h_{n,m}^{k,i}.$$  

Assumption 2. For any $\pi$ and $\hat{r}$ with $\| \pi - \pi \| \leq \epsilon_z$ and $\| \hat{r} - r \| \leq \epsilon_r$, if $g(\alpha)$ is polyhedral with parameter $\rho$, $\hat{g}(\alpha)$ is also polyhedral with parameter $\rho$.  

In actual implementation, one can still serve the tasks with the actual allocated resource.
Assumption 3. For any \( \tilde{\pi} \) and \( \tilde{r} \) with \( \| \tilde{\pi} - \pi \| \leq \epsilon_z \) and \( \| \tilde{r} - r \| \leq \epsilon_r \), \( \tilde{g}^*(\alpha) \) has a unique optimal solution over \( \mathbb{R}^M + 2N \). \( \diamond \)

In the network optimization literature, e.g., [12], [24]. Assumption 1 is commonly assumed with \( \epsilon_r = \epsilon_z = 0 \). By allowing \( \epsilon_r, \epsilon_z > 0 \), we assume that systems with similar statistics have similar stability regions. Assumption 2 assumes that systems with similar statistics share a similar dual structure. This is not restrictive. In fact, when action sets are discrete, it is often the case that \( g_k(\alpha) \) is polyhedral, which usually leads to a polyhedral structure of \( \tilde{g}^*(\alpha) \). The uniqueness assumption is also often guaranteed by the utility maximization structure, e.g., [25].

Below, we adopt the following big-O notations taken from Chapter 3 in [26]. Specifically, we say that \( f(x) = O(g(x)) \) if there exists a constant \( c_1, c_2 < \infty \) such that \( 0 \leq f(x) \leq c_1 g(x) \) for all \( x \geq x_1 \), and that \( f(x) = \Theta(g(x)) \) if \( f(x) = O(g(x)) \) and \( g(x) = O(f(x)) \). In this work, the notations are defined with respect to the parameter \( V \).

B. Queue and Utility Performance

To state our results, we define the following events about the correctness of the estimations of the reward and distribution functions, i.e.,

\[
E_\epsilon = \{ \| \tilde{r} - r \| \leq \delta_r \} , \quad E_\epsilon = \{ \| \tilde{\pi} - \pi \| \leq \delta_r \} .
\]

We first summarize the performance of LRAM.

Theorem 1. Suppose \( T_{\delta_r} < \infty \) with probability 1. Under LRAM, we have for all \( t \geq T_{\delta_r} + 1 \) that:

\[
d_n(t) \leq d_{\max} \triangleq V \beta + r_{\max}, \quad \forall n \tag{43}
\]

\[
Q_n(t) \leq Q_{\max} \triangleq \theta_1 + r_{\max}, \quad \forall n \tag{44}
\]

\[
H_m(t) \leq H_{\max} \triangleq \theta_2 + h_{\max}, \quad \forall m. \tag{45}
\]

Moreover, with probability \( P_{\delta_r}, E_\epsilon \) occurs, and

\[
f_{\text{DRAM}}^{\text{av}, E_\epsilon} \geq f_{\text{av}} - \frac{G + r_{\max} \delta_r}{V} - 2N \beta \delta_r. \tag{46}
\]

Here \( f_{\text{DRAM}}^{\text{av}, E_\epsilon} \) denotes the resulting system performance under LRAM, as defined in (10) conditioning on \( E_\epsilon \), i.e.,

\[
\tau_n = \lim_{t \to \infty} \frac{1}{\tau_{t_{\tau_n}}} \sum_{\tau = T_{\tau_n}+1}^{t-1} \mathbb{E} \{ r_n(\tau) | E_\epsilon \} \text{ and } C_{\text{total}} = \lim_{t \to \infty} \frac{1}{\tau} \sum_{\tau = T_{\tau_n}}^{t-1} \mathbb{E} \{ c(\tau) | E_\epsilon \}. \diamond
\]

Proof. See Appendix C. \( \square \)

Theorem 1 shows that LRAM guarantees deterministic queueing bounds of size \( O(V) \), and the utility loss (compared to the optimal value) is \( O(1/V + \delta_r) \), which can be viewed as the performance loss due to inaccurate reward information. We remark here that the deterministic bounds are important for both algorithm implementation and performance guarantees. This is because errors in reward estimation will be amplified by the queue sizes when used in decision making, i.e., (14). Note that if we have \( \delta_r = 0 \) with \( T_{\delta_r} = 0 \) and \( P_{\delta_r} = 1 \), then Theorem 1 recovers the known \( [O(1/V), O(V)] \) utility-delay tradeoff for stochastic network problems [12].

We now present the performance results for DRAM.

Theorem 2. Suppose \( \max(T_{\delta_r}, T_{\delta_r}) < \infty \) with probability 1. Suppose \( g(\alpha) \) is polyhedral with \( \rho = \Theta(1) > 0 \), and that \( \epsilon_r \leq \epsilon_z \) and \( \delta_r \leq \epsilon_r \), and \( \alpha^* + \theta > 0 \). Then, with a sufficiently large \( V \), we have with probability \( P_{\delta_r} \), that both learning modules succeed, i.e., \( E_\epsilon \cap E_\epsilon \) occurs, and that under DRAM,

\[
f_{\text{DRAM}}^{\text{av}, E_\epsilon \cap E_\epsilon} \geq f_{\text{av}} - \frac{G}{V} - O(1/V + \delta_r). \tag{47}
\]

Here \( f_{\text{DRAM}}^{\text{av}, E_\epsilon \cap E_\epsilon} \) is defined with \( \tau_n \) and \( C_{\text{total}} \) being the average expected value conditioning on \( E_\epsilon \cap E_\epsilon \). Also, the fraction of time dropping happens is \( O(V^{-\log(V)}) \). Moreover, for all queues, there exist \( \Theta(1) \) constants \( D, K, a \) such that

\[
\lim_{t \to \infty} \Pr \{ d_n(t) \geq \frac{3}{2} \rho \zeta + D + \nu \} \leq ae^{-K \nu} \tag{48}
\]

\[
\lim_{t \to \infty} \Pr \{ Q_n(t) \geq \frac{3}{2} \rho \zeta + D + \nu \} \leq ae^{-K \nu} \tag{49}
\]

\[
\lim_{t \to \infty} \Pr \{ H_m(t) \geq \frac{3}{2} \rho \zeta + D + \nu \} \leq ae^{-K \nu}. \tag{50}
\]

Thus, all queues are stable. \( \diamond \)

Proof. See Appendix D. \( \square \)

Theorem 2 shows that the utility performance of DRAM is similarly within \( O(1/V + \delta_r) \) of the optimal. However, instead of showing deterministic bounds as in Theorem 1, it shows that the queues will rarely go beyond \( \zeta = O(\delta_r V \log(V)^2 + \log(V)^2) \). This provides additional insight into how learning accuracy affects delay performance. Similar to the LRAM case, if we also have \( \delta_r = 0 \) with \( T_{\delta_r} = 0 \) and \( P_{\delta_r} = 1 \), then DRAM provides a new way for achieving the near-optimal \([O(1/V), O(\log(V)^2)]\) utility-delay tradeoff.

In both LRAM and DRAM, it is possible to continuously update the reward function estimations during control steps. However, this does not automatically guarantee that we can always eliminate the effect of \( \delta_r \). This is so because the initial estimation \( \tilde{r} \) may affect what options will be continuously updated later. On the other hand, the same performance results will hold if further updates do not increase \( \delta_r \).

C. Convergence time

Here we look at another important performance metric - algorithm convergence time, which characterizes the time it takes for the algorithm to enter the “steady state.” Faster convergence implies better robustness against system statistics changes and higher efficiency in resource allocation, and is particularly important when system statistics can change. The formal definition of convergence time is as follows [17].

Definition 3. For a given constant \( D \), the \( D \)-convergence time of a control algorithm \( \Pi \), denoted by \( T_D^\Pi \), is the time it takes for the queue vector \( (d(t), Q(t), H(t)) \) under DRAM to get to within \( D \) distance of \( \alpha^* + \theta, \) i.e.,

\[
T_D^\Pi = \inf \{ t | \| (d(t), Q(t), H(t)) - (\alpha^* + \theta) \| \leq D \}. \tag{51}
\]

This definition of convergence time concerns about when an algorithm enters its “optimal state,” i.e., when the queue vectors get very close to the optimal multiplier and the algorithms start making optimal action selections. It is different from the metrics considered in [27] and [28], which concern about the time it takes for the objective value and constraints to be within certain accuracy. The following theorem summarizes the convergence performance results.
Theorem 3. Suppose \( g(\alpha) \) is polyhedral with \( \rho = \Theta(1) > 0 \), \( \delta_x \leq \epsilon_x \) and \( \delta_x \leq \epsilon_x \), and \( \alpha^* + \theta > 0 \). Then, with a sufficiently large \( V \), we have: \( \delta_x \) 

\[
E\{T_{\text{DAM}}^T \mid \mathcal{E}_T\} = O(T_{\delta_x} + \Theta(V)) \tag{52}
\]

\[
E\{T_{\text{DAM}}^{\text{LRAM}} \mid \mathcal{E}_T \cap \mathcal{E}_T\} = O((T_1 + \Theta(\delta_x V)) \tag{53}
\]

Here \( T_1 \) denotes the total learning time in \( \text{DAM} \), \( D_1 \) denotes the convergence time for \( \text{LRAM} \), and \( D_2 \) denotes the convergence time for \( \text{DAM} \) and \( \text{LRAM} \) with probability \( \delta_x \), the convergence time for \( \text{DAM} \) is \( O((T_1 + \Theta(\delta_x V)) \).

Proof. See Appendix E.

Theorem 3 shows how performance guarantees of learning modules affect the convergence time. Interestingly, \( \text{DAM} \) has a convergence time of \( O(\delta_x V) \), which can be significantly smaller than that under \( \text{DAM} \) when \( \delta_x \) is small. Combining Theorems 1, 2, and 3, we see that \( \delta_x \) largely affects the utility performance (reflected by \( D_1 \) and \( D_2 \)), while \( \delta_x \) can greatly improve the convergence time and delay. This indicates that information of different system components can play very different roles in performance and learning accuracies should be carefully chosen for meeting a desired performance goal.

We remark here that though Theorems 2 and 3 appear to be similar to results in [17], the analysis is different due to allowing general learning methods, errors in estimating \( r_n(z, \mu_n(t)) \) and \( \gamma_n(t) \), integrating them into dual learning. Moreover, the explicit quantifications in Theorems 2 and 3 provide insight into how learning modules should be designed and how they can be integrated with control algorithms.

D. Necessity in Controlling \( \delta_x \)

We provide a simple example showing that it is necessary to control \( \delta_x \) for good utility performance. Hence, it is important to learn the reward value for each matching option. Consider the case when \( N = 2 \) and \( M = 1 \). Suppose \( \epsilon(t) = A_1(t) = A_2(t) = 0 \) for all \( t \). Also suppose \( b(t) \in \{0, 1\} \), that is, at every time \( t \), we can only allocate resource to one or zero queue. Suppose \( \epsilon(t) = 0 \), \( \mu_n(t) = b_n(t) \), and \( r_n(t) = \mu_n(t) \). Finally, assume that \( U_1(r_1) = \log(1 + r_1) \) and \( U_2(r_2) = \log(1 + 2r_2) \).

In this case, the true optimal takes place at \( r_1 = 1/4 \) and \( r_2 = 3/4 \) with \( U_{\text{total}} = 0.7828 \). Now suppose we incorrectly estimate the rewards to be \( r_n(t) = (1 + \delta_x)\mu_n(t) \). Then, one can show that the optimal rewards become:

\[
r_1 = \frac{2(1 + \delta_x)(1 + \delta_x) - 2(1 + \delta_x) + (1 + \delta_x)}{4(1 + \delta_x)(1 + \delta_x)} \tag{54}
\]

\[
r_2 = \frac{2(1 + \delta_x)(1 + \delta_x) + 2(1 + \delta_x) - (1 + \delta_x)}{4(1 + \delta_x)(1 + \delta_x)} \tag{55}
\]

which is roughly \( r_1 \approx \frac{1}{4} + \frac{\delta_x}{4} \) and \( r_2 \approx \frac{3}{4} - \frac{\delta_x}{4} \). Thus, the resulting optimal utility is given by:

\[
U_{\text{total}} \approx 0.7828 - \frac{\delta_x}{4} \tag{56}\]

Therefore, in order to obtain an \( O(1/V) \) close-to-optimal utility, it is necessary to ensure that \( \max(|\delta_1|, |\delta_2|) = O(1/V) \).

VIII. Simulation

In this section, we present simulation results for our algorithms. We consider a system that has \( N = 2 \) and \( M = 1 \). We assume that \( \epsilon(t) \) is 0 or 2 with equal probabilities. \( A_1(t) \) is 0 or 2 with equal probabilities and \( A_2(t) \) is 1 or 2 with equal probabilities. \( b(t) \in \{0, 1\} \), i.e., at every time \( t \), we allocate one unit resource to one or zero queue. We set \( \epsilon(t) = b_1(t) + b_2(t) \) and \( \mu_n(t) = b_n(t) \), and \( \gamma_n(t) \in \{0, 1, 2\} \).

There are two system states \( \omega(t) \in \{1, 2\} \). In each state, the reward functions are given by \( r_n(t) = w_n(\omega(t))\mu_n(t) \), where \( w_1(1) = 0.8 \) and \( w_2(1) = 1 \) and \( w_1(2) = 1 \) and \( w_2(2) = 0.8 \). Thus, the system state indicates which tasks are preferred under specific conditions. Every time the corresponding reward is either \( 0.5r_n(t) \) or \( 1.5r_n(t) \), with equal probabilities.

Finally, we assume that \( U_1(r_1) = 1.2 \log(1 + 2r_1) \) and that \( U_2(r_2) = 1.2 \log(1 + 4r_2) \).

From the definitions, we have that \( \beta = 5 \), \( \beta^u = 1 \) and \( \beta^d = \max_{\omega(n)}(w_n) \). We also set \( r_{\max} = 2 \), \( \mu_{\max} = 1 \) and \( b_{\max} = 1 \), \( h_{\max} = 2 \). We simulate both LRAM and \( \text{DAM} \) with \( V = \{10, 20, 50, 80, 100\} \). According to (21) and (22), \( \theta_1 = (5V + 2)\beta_1 + 4 \) and \( \theta_2 = (5V + 2)\beta_2 + 3 \). In \( \text{DAM} \), we set \( \zeta = \log(V)^2 \) to see if using this smaller offset can still achieve good performance. We use TBS(s_t) to estimate \( r \) and set \( s_{th} = \log(V)^2 \), and use TLS(T_{th}) to estimate \( \pi \) and set \( T_{th} = T_{th} \). In \( \text{DAM} \), we randomly add or subtract the estimation error \( \delta_r \) from the true values.

Fig. 3 then shows the utility performance and the task queue size behavior of \( \text{LRAM} \) and \( \text{DAM} \), where the number after \( \text{LRAM} \) denotes \( \delta_x \). We see from the left plot that except for \( \delta_x = 0.1 \), \( \text{LRAM} \) performs very well under all other error values, suggesting that estimation error indeed plays an important role in system utility. We also see that \( \text{DAM} \) performs very well starting from \( V \geq 50 \). The right plot shows the backlog (delay) performance under different schemes. It is evident that \( \text{DAM} \) achieves an \( O(\log(V)^2) \) delay in this case, while all other variants possess an \( O(V) \) delay. This demonstrates the importance of incorporating system dynamics information into algorithm design.

Fig. 4 then shows the resource queue \( H(t) \) and deficit queues \( d(t) \). We see that \( \text{DAM} \) ensures an \( O(\log(V)^2) \) average resource queue, while other algorithms result in an \( O(V) \) queue size. This implies that \( \text{DAM} \) ensures a short stay in the system for the resource items. This feature is particularly useful if the resource items are human users, e.g., in crowdsourcing.
Finally, Fig. 5 shows the convergence behavior of the algorithms for $V = 100$. Here we show the resource queue value as its convergence time dominates the others. We see that RAM takes an $O(V)$ time to converge, which is expected. We also observe that $H(t)$ under LRAM-0.05 and LRAM-0.1 converge to values slightly above those under RAM. This explains why their convergence is slightly worse. On the other hand, we also see that DRAM converges quickly. The reason its steady state value is slightly above that under RAM is due to the inaccuracy of $\hat{r}$. Even in this case, we see that there is a $2.5 \times$ convergence speedup (most of the learning time is due to sampling) and DRAM achieves very good performance. In the case where $r$ can be obtained from other data source beforehand, which can commonly be done in practice, e.g., in online advertising, we see that DRAM (called state-only DRAM in this case) achieves a $10 \times$ convergence speedup (50 slots vs. 500 slots).

We observe in all simulation instances that no dropping occurs. This demonstrates the effectiveness of the algorithms and validates Theorem 2.

IX. CONCLUSION

In this paper, we study the problem of optimal matching with queues in dynamic systems. We develop two online learning-aided algorithms LRAM and DRAM for resolving the challenging underflow problem and to achieve near-optimality. We show that LRAM achieves an $O(1 + \delta_v)$ system utility, for any $\epsilon > 0$, while ensuring an $O(1/\epsilon)$ delay bound and an $O(1/\epsilon)$ algorithm convergence time (Here $\epsilon = 1/V$). DRAM, on the other hand, guarantees a similar $O(1 + \delta_v)$ system utility, while achieving an $O(\delta_v/\epsilon + \log(1/\epsilon)^2)$ delay bound and an $O(\delta_v/\epsilon)$ algorithm convergence time, which can be significantly better compared to LRAM when $\delta_v$ is small. Our algorithms and results show that different system information can play very different roles in algorithm performance and provide insights into joint learning-control algorithm design for dynamic systems.

REFERENCES

Rearranging terms, we obtain:

\[ G \leq V\left( \sum_{n}U_n(\gamma_n(t)) - c(t) \right) - \sum_{n}(Q_n(t) - \theta_1)(\mu_n(t) - R_n(t)) \]

\[ - \sum_{n}d_n(t)(\kappa_n(t) - \gamma_n(t)) \]

\[ - \sum_{n}(H_n(t) - \theta_2)(\sum_{n}b_m(t) - h_n(t)) \]

Here \( G \triangleq N(A_{max}^2 + \mu_{max}^2 + 2r_{max}^2) + MH_{max}^2 + MN^2b_{max}^2 \).

Rearranging terms, we obtain:

\[ L(t+1) - L(t) - Vf_\gamma(t) \]

\[ \leq G - \sum_{n}[VU_n(\gamma_n(t)) - d_n(t)\gamma_n(t)] \]

\[ + \sum_{n}(Q_n(t) - \theta_1)R_n(t) + \sum_{n}(H_n(t) - \theta_2)h_n(t) \]

\[ + \left[ Vc(t) - \sum_{n}(H_n(t) - \theta_2)\sum_{n}b_m(t) \right] \]

\[ \sum_{n}(Q_n(t) - \theta_1)\mu_n(t) - \sum_{n}d_n(t)\kappa_n(t) \]

Taking an expectations on both sides conditioning on \( y(t) \) and using the fact that \( \kappa_n(t) \) is an i.i.d. random variable given \( z(t), b(t), Q(t) \), we see that the lemma follows.

**APPENDIX B – PROOF OF LEMMA 2**

We prove Lemma 2 here. In our proof, we use the following theorem from [29] (also known as the Bernstein inequality [30]).

**Theorem 4.** [29] Suppose \( X_i \) are independent random variables satisfying \( X_i \leq B \) for \( 1 \leq i \leq n \). Let \( X = \sum_{i}X_i \) and \( \|X\| = \sqrt{\sum_{i}E[X_i^2]} \). Then we have:

\[ \Pr\{X \leq \mathbb{E}\{X\} - b\} \leq e^{-\frac{b^2}{2\|X\|^2 + B^2/3}} \].

(57)

We now present the proof of Lemma 2.

**Proof.** (Lemma 2) First of all, we show that \( \mathbb{E}\{T_{th}\} = \Theta(\log(V)^2) \). To see this, notice that since each \( B_k \) is finite, if the state \( z(t) = k \) appears \( |B_k|s_{th} \) times, we must have sampled every \( b \in B_k \) \( s_{th} \) times. Thus,

\[ T_{ths} \leq \sum_{k}T\{\text{visit } z_k \mid B_k \} \]

(58)

Taking the expectations, we see that \( \mathbb{E}\{T_{ths}\} \leq s_{th}\sum_{k}^{|B_k|} \frac{|B_k|}{s_{th}} \).

Then, we see that in \( \hat{r} \), each single value has been sampled \( s_{th} \) times. Using Theorem 4 and that \( \kappa_n(t) \leq r_{max} \), we get:

\[ \Pr\left\{ \sum_{t=1}^{\hat{r}_n(z(t), b(t))} \kappa_n(t) \leq r_n(z, b)s(z, b, T_{ths}) - b \right\} \]

\[ \leq e^{-\frac{r_{max}^2}{2\|X\|^2 + B_{x}^2/3}} \].

Choosing \( b = \sqrt{s_{th}\log(s_{th})} \), dividing both sides of the inequality inside by \( s_{th}\), and using \( s(z, b, T_{ths}) \geq s_{th} \),

\[ \Pr\{\hat{r}_n(z, b) \leq r_n(z, b) - \log(s_{th})\} \leq e^{-\frac{s_{th}\log(s_{th})^2}{\sqrt{s_{th}\log(s_{th})^3}}} \].

Using Theorem 4 with \(-X\), we get a similar bound for the other side. Hence,

\[ \Pr\{\hat{r}_n(z, b) - r_n(z, b) \leq \log\left(s_{th} \right) \leq 2e^{-\frac{s_{th}\log(s_{th})^2}{\sqrt{s_{th}\log(s_{th})^3}}} \].

Using the union bound, we see that Part (a) follows. Part (b) can be proven similarly. □

**APPENDIX C – PROOF OF THEOREM 1**

We present the proof for Theorem 1 here. For our analysis, we use the following result, which is Theorem 1 in [31]. The result states that the optimal dual value \( g(\alpha^*) \) provides an upper bound for the optimal system utility.

**Theorem 5.** [31] Let \( \alpha^* \) be an optimal solution of (30). Then, \( g(\alpha^*) \geq Vf_{\alpha^*} \). □

**Proof.** (Theorem 1) (Queueing) First consider \( d_n(t) \). We see that if \( d_n(t) \leq V/\beta \), then \( d_n(t + 1) \leq V/\beta + r_{max} \). On the other hand, from (13), whenever \( d_n(t) > V/\beta \), \( \gamma_n(t) = 0 \). Hence, \( d_n(t) \) will not further increase. This proves the bound for \( d_n(t) \). The bounds for \( Q_n(t) \) and \( H_n(t) \) can be similarly proven by noticing that LRAM will not further admit tasks once \( Q_n(t) \geq \theta_1 \) and it does not admit resources once \( H_n(t) \geq \theta_2 \).

(Utility) We carry out the proof by comparing the RHS of (18) under LRAM with any other control policy, including the ones that do not respect the no-underflow constraint (1).

To this end, consider (14). We show that even without constraint (1), LRAM ensures that (i) whenever \( H_n(t) \leq Nb_{max} \), \( b_{mn}(t) = 0 \) for all \( n \), and (ii) whenever \( Q_n(t) \leq \mu_{max} \), \( \mu_n(t) = 0 \). This intermediate step allows us to compare the drift value under LRAM over all possible alternative policies, including ones that do not respect the no-underflow constraint. We first show (i). Suppose \( H_n(t) \leq Nb_{max} \). Using (22),

\[ H_n(t) - \theta_2 < -(V/\beta + r_{max})\beta_2 - r_{max}\beta_2 \mu_n(t) \]

(59)

In this case, denote the optimal resource allocation vector as \( b^* \) and suppose there is one \( n \) with \( b_{mn}^* \geq 0 \). Let \( b \) be the vector obtained from \( b^* \) by setting \( b_{mn}^* = 0 \). We have:

\[ \Psi_{\hat{r}}(b^*) - \Psi_{\hat{r}}(b) \]

\[ = Vc(z, b^*) - Vc(z, b) - (H_n(t) - \theta_2)b_{mn} \]

(60)

\[ - (Q_n(t) - \theta_1)(\mu_n(z(t), b^*) - \mu_n(z(t), b)) \]

\[-d_n(t)(\hat{r}_n(z(t), \mu_n(z(t), b^*)) - \hat{r}_n(z(t), \mu_n(z(t), b))) \]

\[ = -d_n(t)(\hat{r}_n(z(t), \mu_n(z(t), b^*)) - \hat{r}_n(z(t), \mu_n(z(t), b))) \]

\[ = -d_n(t)(\hat{r}_n(z(t), \mu_n(z(t), b^*)) - \hat{r}_n(z(t), \mu_n(z(t), b))) \]

\[ = -d_n(t)(\hat{r}_n(z(t), \mu_n(z(t), b^*)) - \hat{r}_n(z(t), \mu_n(z(t), b))) \]

\[ = -d_n(t)(\hat{r}_n(z(t), \mu_n(z(t), b^*)) - \hat{r}_n(z(t), \mu_n(z(t), b))) \]

\[ = -d_n(t)(\hat{r}_n(z(t), \mu_n(z(t), b^*)) - \hat{r}_n(z(t), \mu_n(z(t), b))) \]
In the inequality, we have used the fact that $Q_n(t) - \theta_1 \leq r_{\max} \mu_n(z(t), b^*) - \mu_n(z(t), b^*) \leq b_{\mu}^*, d_n(t) \leq V + r_{\max},$ and $r_n(z(t), \hat{\mu}_n(z(t), b^*)) - r_n(z(t), \hat{\mu}_n(z(t), b^*)) \leq b_{\mu}^*.$ However, $(66)$ contradicts with the fact that $b^*$ is the minimizer of $\Psi_r(b)$ and shows that we must have $b_{mn}^* = 0 \forall n$ whenever $H_m(t) < N_{\max}.$

Now we look at the case of $Q(t).$ Suppose $Q_n(t) < \mu_{\max}.$ Then, we have:

$$Q_n(t) - \theta_1 \leq -(h_{\max} + (V + r_{\max}) \beta_1) / \beta_1.$$  (61)

We similarly let $b^*$ be the optimal solution. Then, we construct $\hat{b}$ by setting $b_{mn}^* > 0$ to zero, where $m^* = \arg \min_n b_{mn}^*.$ In this case, if $\mu_n(z(t), b^*) = 0,$ we are done. Otherwise, $\Psi_r(b^*) - \Psi_r(\hat{b}) > -h_{\max} b_{mn}^* - (V + r_{\max}) \beta_1 b_{mn}^* + (h_{\max} + (V + r_{\max}) \beta_1) \beta_1 b_{mn}^* = 0.$

The inequality follows since $H_m(t) - \theta_2 \leq h_{\max}, d_n(t) \leq V + r_{\max},$ and that $\mu_n(z(t), b^*) > 0,$ which implies $\mu_n(z(t), b^*) \geq \beta_1 b_{mn}^*.$ This contradicts with the fact that $b^*$ is the minimizer and shows that $b_{mn}^* = 0.$

These two properties imply that LRAM automatically guarantees that the no-underflow constraints are satisfied for all $H_m(t)$ and that we always have $\hat{\mu}_n(t) = \min\{Q_n(t), \mu_n(t)\} = \mu_n(t).$ To compare our control policy with any other matching policies for the drift (18), we still need to show that the actions under LRAM, which is based on the estimated reward matrix $\hat{r},$ ensure that the RHS of (18), defined with the true reward $r,$ is approximately minimized.

To do so, first observe that $\gamma(t), R(t)$ and $h(t)$ are optimally chosen given $y(t)$ and $z(t).$ Hence, the only approximation comes from choosing $b(t).$ Let $b_{\hat{r}}^*$ be the chosen vector under LRAM and let $b_{\hat{r}}^*(t)$ be the vector chosen if $r$ is used. We have:

$$\Psi_r(b_{\hat{r}}^*(t)) \leq \Psi_r(b_{\hat{r}}^*(t)) + \sum_n d_n(t)[\hat{r}_n(b_{\hat{r}}^*(t)) - r_n(b_{\hat{r}}^*(t))].$$  (63)

On the other hand, we also have:

$$\Psi_r(b_{\hat{r}}^*(t)) = \Psi_r(b_{\hat{r}}^*(t)) + \sum_n d_n(t)[\hat{r}_n(b_{\hat{r}}^*(t)) - r_n(b_{\hat{r}}^*(t))].$$

Combining the above two equalities and using the fact that $\Gamma_r$ is a $(T_{\delta}, P_r, \delta_r)$-learning module, we see that with probability $P_{\delta_r},$ for all $t \geq T_0,$

$$\Psi_r(b_{\hat{r}}^*(t)) \leq \Psi_r(b_{\hat{r}}^*(t)) + 2 \sum_n d_n(t) \delta_r \leq \Psi_r(b_{\hat{r}}^*(t)) + 2N(V + r_{\max}) \delta_r.$$  (64)

This shows that the RHS of (18) under LRAM is minimized to within $2N(V + r_{\max}) \delta_r,$ over any other policies. Comparing this to $g(\alpha^x, \alpha^y, \alpha^z)$ in (31) and using the definition of $g(\alpha^x, \alpha^y, \alpha^z),$ we conclude that:

$$\Delta(t) = \mathbb{E}\{L(t+1) - L(t) - V f_r(t) | y(t)\}$$

$$\leq G + 2N(V + r_{\max}) \delta_r - g(d(t), Q(t), H(t))$$

$$\leq G + 2N(V + r_{\max}) \delta_r - V f_{av}.$$  (65)

Here (a) follows from Theorem 5. Taking an expectation over $y(t)$ on both sides and carrying out a telescoping sum from $t = 0, ..., T - 1,$ and dividing both sides by $VT,$ we obtain:

$$\frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}\{f_r(t)\} = \frac{1}{T} \sum_{t=0}^{T-1} \sum_n \mathbb{E}\{U_n(\gamma_n(t)) - c(t)\}$$

$$\geq f_{av} - \frac{G + r_{\max} \delta_r}{V}.$$  (66)

Finally, using the fact that $d(t)$ is bounded, which implies $\tau_n \leq \tau_n$ for all $n,$ and that $U_n(r)$ is increasing, we see that the tree follows.

**APPENDIX D – PROOF OF THEOREM 2**

Here we prove the performance of DRA. We carry out the analysis in the following two steps. First, we show that the estimated optimal multiplier $\alpha^* = (\alpha^x, \alpha^y, \alpha^z),$ the optimal solution of $\hat{g}(\alpha),$ is close to the true optimal. Then, we show via drift-augmentation that the definitions of $Q(t), H(t),$ and $d(t)$ ensure a near-optimal algorithmic performance.

We now have the following lemma for the first step, which characterizes the distance between the estimated multiplier and the true value as a function of $\delta_r.$ In the lemma, we denote $\alpha^*$ the optimal solution for $\hat{g}(\alpha),$ which is $g(\alpha)$ defined with $\pi$ and $\hat{r}.$

**Lemma 3.** Suppose $g(\alpha)$ is polyhedral with $\rho = \Theta(1) > 0,$ and that $\delta_\pi \leq \epsilon_2$ and $\delta_r \leq \epsilon_3.$ Then, with probability $P_{\delta_r},$ we have:

$$\|\alpha^* - \bar{\alpha}^x\| \leq \frac{2 \delta_{V} \mu_{\max} \bar{\alpha}}{\rho}$$  (67)

$$\|\alpha^* - \bar{\alpha}^x\| \leq \frac{2 \delta_{V} \mu_{\max} \bar{\alpha}}{\rho \eta\},$$

where $\eta \triangleq \frac{1}{\rho}(1 + r_{\max} + A_{\max} + \mu_{\max} + N b_{\max} + h_{\max}) / \eta$ and $\eta = \Theta(1).$  \(\square\)

**Proof.** See Appendix F.  \(\square\)

In our analysis, we make use of the following two results. In particular, Lemma 4 is a result linking queue underflow probability with the difference in the average service and arrival rate, whereas Theorem 6 shows that under LRAM, the queue values are exponentially attracted to the optimal multipliers.

**Lemma 4.** [17] Let $Q(t)$ be the size of a single queue with dynamics $Q(t+1) = Q(t) - (\mu(t)) + A(t).$ Suppose $0 \leq \mu(t), A(t) \leq \mu_{\max} = \Theta(1)$ for all $t$ and that the queue is stable. Then,

$$\mu(t) - A(t) \leq \mu_{\max} \lim_{t \to \infty} P_r(Q(t) < \mu_{\max}).$$

Here $\bar{x}(t) \triangleq \lim_{t \to \infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}\{x(t)\}.$

**Theorem 6.** Under LRAM with reward functions $\hat{r},$ there exist $\Theta(1)$ constants $a, K,$ and $D,$ such that,

$$P(D, \nu) \leq ae^{-K \nu},$$

where $P(D, \nu)$ is defined as:

$$P(D, \nu)$$
\[ \Delta_a(t) \triangleq - \sum_n (\hat{\alpha}_n^* - \zeta) \mathbb{E}\{\mu_n(t) - R_n(t) \mid y(t)\} \]

Using Lemma 3, we know that with probability \( P_{\theta, P_{\delta_{\nu}}} \), \( \|\hat{\alpha}^* - \alpha^*\| \leq \frac{2\delta_\nu V_f \max \delta_r}{\rho} \). Using \( \zeta \triangleq 2\max(\delta_r, V \log(V^2), \log(V^2)^2) \), we see that when \( V \) is large, we have:

\[ \frac{2\delta_\nu V_f \max \delta_r}{\rho} \leq \zeta/2. \]

Thus,

\[ \frac{3}{2} \zeta \leq \hat{\alpha}^* - \zeta \leq \hat{\alpha}^* - \zeta \leq \frac{1}{2} \zeta, \]

where the inequality is taken entry-wise. This implies that with probability \( P_{\theta, P_{\delta_{\nu}}} \), for each \( n \),

\[ \hat{d}_n(t) \leq \max(V\beta + r_{\max}, \hat{d}_n^* - \zeta/2) \]

\[ \leq \hat{d}_{\max} \triangleq \max(V\beta + r_{\max}, V \max(\delta_r, V \log(V^2), \log(V^2)^2) / \zeta/2). \]

The second inequality uses (87) in Appendix F. We now carry out a similar argument as in the proof of Theorem 1 and conclude that:

\[ \Delta_V(t) + \Delta_a(t) \leq G - 2N \delta_r \hat{d}_{\max} - g(\tilde{d}(t), \tilde{Q}(t), \tilde{H}(t)) \]

\[ \leq G - 2N \delta_r \hat{d}_{\max} - V f_{aw}. \]

Carrying out a telescoping sum and taking a limit as in the proof of Theorem 1, we obtain:

\[ \sum_n U_n(\tau_n) - \bar{c} \]

\[ \geq f_{aw} - \frac{G + 2N \hat{d}_{\max} \delta_r}{V} - \lim_{T \to \infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}\{\Delta_a(t)\}. \]

It remains to show that all queues are finite, and that \( \lim_{T \to \infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}\{\Delta_a(t)\} = O(1/V) \). Then, we can conclude \( \tau_n \geq \tau_n \) and complete the proof.

To this end, we first use (70) and the definition of \( \tilde{d}(t) \), \( \tilde{Q}(t) \), and \( \tilde{H}(t) \), to obtain that:

\[ \lim_{t \to \infty} \mathbb{P}\{d_n(t) \geq \frac{3}{2} \zeta + D + \nu\} \leq ae^{-K\nu} \]

\[ \lim_{t \to \infty} \mathbb{P}\{Q_n(t) \geq \frac{3}{2} \zeta + D + \nu\} \leq ae^{-K\nu} \]

\[ \lim_{t \to \infty} \mathbb{P}\{H_n(t) \geq \frac{3}{2} \zeta + D + \nu\} \leq ae^{-K\nu}, \]

which are exactly the queuing probability bounds (48), (49), and (50). Specifically, using the definition of \( \tilde{d}(t) \) and (74), we have:

\[ \{d_n(t) \geq \frac{3}{2} \zeta + D + \nu\} \subset \{\tilde{d}_n(t) - \tilde{d}_n^* \geq D + \nu\}. \]

This gives the bound for \( d_n(t) \). The bounds for \( Q_n(t) \) and \( H_n(t) \) can be reasoned similarly. Using (70) again, we see that for a large \( V \) such that \( \zeta \geq D + \mu_{\max} + N b_{\max} + r_{\max} + 2\log(V)/K \),

\[ \lim_{t \to \infty} \mathbb{P}\{d_n(t) \leq r_{\max}\} \leq \frac{a}{V^2} \]

\[ \lim_{t \to \infty} \mathbb{P}\{Q_n(t) \leq \mu_{\max}\} \leq \frac{a}{V^2} \]

\[ \lim_{t \to \infty} \mathbb{P}\{H_n(t) \leq N b_{\max}\} \leq \frac{a}{V^2}. \]

Combining the above bounds and Lemma 4, and that \( \hat{\alpha}^* = O(V) \) by (87), we conclude that \( \lim_{t \to \infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}\{\Delta_a(t)\} = O(1/V) \). Moreover, since \( d(t) \) is stable, \( \tau_n \geq \tau_n \). Finally, using (79) - (81), we see that the fraction of time dropping happens, i.e., when the claimed reward does not count, is \( O(1/V^2) \). Since \( \beta = \Theta(1) \), this results in an additional utility loss of \( O(1/V^2) \). Hence, we conclude that:

\[ \sum_n U_n(\tau_n) - \bar{c} \geq f_{aw} - \frac{G + 2N \hat{d}_{\max} \delta_r}{V} - O(1/V). \]

This completes the proof.

**APPENDIX E – PROOF OF THEOREM 3**

**Proof.** (Theorem 3) To prove the result, we define a different Lyapunov function as follows:

\[ L_0(t) = \frac{1}{2} \|d(t), Q(t), H(t) - \theta - \hat{\alpha}^*\|^2. \]

Then, we define a one-slot conditional Lyapunov drift as \( \Delta_0(t) = \mathbb{E}\{L_0(t + 1) - L_0(t) \mid y(t)\} \). Using the queuing dynamics, we obtain that:

\[ \Delta_0(t) \leq G - \sum_n (\hat{\alpha}_n^* - \zeta) \mathbb{E}\{\mu_n(t) - R_n(t) \mid y(t)\} \]

\[ - \sum_n (\hat{d}_n(t) - \hat{d}_n^*) \mathbb{E}\{r_n(t) - \gamma_n(t) \mid y(t)\} \]

\[ - \sum_m (\hat{h}_m(t) - \hat{h}_m^*) \mathbb{E}\{b_{mn}(t) - h_m(t) \mid y(t)\}. \]

Using that the last three components constitute the subgradient of \( \tilde{g}(\alpha) \) at \( \alpha = (d(t), Q(t), H(t)) \) [32], we obtain:

\[ \Delta_0(t) \leq G - \tilde{g}(\alpha) \mathbb{E}\{d(t), Q(t), H(t) - \theta - \hat{\alpha}^*\}. \]

Therefore, for any \( 0 < \epsilon_0 < \rho \), if \( \|d(t), Q(t), H(t) - \theta - \hat{\alpha}^*\|^2 \geq G \rho / \epsilon_0 \), the above implies that:

\[ \mathbb{E}\{\|d(t + 1), Q(t + 1), H(t + 1) - \theta - \hat{\alpha}^*\|^2 \mid y(t)\} \]

\[ \leq G \rho / \epsilon_0 \]

\[ \epsilon_0 \]

This is always possible as \( \zeta = \Omega(\log(V)^2) \).
which further implies that:
\[
\mathbb{E}\{||d(t), Q(t), H(t)|| - \theta - \alpha^*|| - \epsilon_0\}^2 \\
\leq \mathbb{E}\{||d(t + 1), Q(t + 1), H(t + 1)|| - \theta - \alpha^*|| - y(t)\}
\]
\[
\leq ||d(t), Q(t), H(t)|| - \theta - \alpha^*|| - \epsilon_0.
\]

Then, using the fact that \(\alpha^* = \Theta(V)\) \([23]\), \((d(T_\delta + 1), Q(T_\delta + 1), H(T_\delta + 1)) = 0\), and using Lemma 5 in \([17]\), we conclude then:
\[
\mathbb{E}\{T_{D|\alpha}^{\text{LRAM}}\} = \mathbb{E}\{T_\delta + \Theta(V)\}.
\]

Here \(D'_1 = G(p - \epsilon_0) = \Theta(1)\) and \(\mathbb{E}\{T_{D|\alpha}^{\text{LRAM}}\}\) denotes the expected time to get to \(D_1'\) of \(\alpha^*\). Using (68) in Lemma 3, and by defining \(D_1 \triangleq D'_1 + \frac{2V_{\text{max}}}{\rho_0}\), we conclude that:
\[
\mathbb{E}\{T_{D|\alpha}^{\text{DRAM}}\} = \mathbb{E}\{T_\delta + \Theta(V)\}.
\]

This proves (52). To prove (53), note that the main difference between DRAM and LRAM is that DRAM utilizes the system state information to “jump start” the algorithm. Using Lemma 3 again, we see that with probability \(P_\delta P_{\tilde{\delta}}\),
\[
||d(0), Q(0), H(0)|| - \theta - \alpha^*|| \leq 2\frac{V_{\text{max}}}{\rho_0} \epsilon_0.
\]
Combining this result and (85), we conclude that:
\[
\mathbb{E}\{T_{D|\alpha}^{\text{DRAM}}\} \leq \mathbb{E}\{T_\delta + \Theta\left(\frac{2\delta V_{\text{max}}}{\rho_0}\right)\}.
\]
This proves (53) and completes the proof of the theorem. □

APPENDIX F – PROOF OF LEMMA 3

We first show that the optimal multipliers are bounded. Then, we use this result to bound the differences among the optimal dual values. For notation simplification, we define \(f_{\text{max}} \triangleq Nbr_{\text{max}} + c_{\text{max}}\).

\textbf{Proof.} (Lemma 3) Since with probability \(P_\delta P_{\tilde{\delta}}\), \(\delta_1 \leq \epsilon_r\) and \(\delta_2 \leq \epsilon_z\), we have from Assumption 1 that there exists a set of actions and probabilities that guarantee (39), (40), and (41). Also, since \(0 < \sum_k \hat{\pi}_k \sum_i \lambda_i h_{im} < \mathbb{E}\{A_n(t)\}\) and \(0 < \sum_k \hat{\pi}_k \sum_i \lambda_i h_{im} < \mathbb{E}\{m(t)\}\), it can be shown that there exists \(\eta_1 = \Theta(1) > 0\), such that for any subset \(\mathcal{I}_n \subset \mathcal{N}\) and any subset \(\mathcal{I}_m \subset \mathcal{M}\), there exist a set of actions \(\{\hat{R}^k_{1m}\}_{k=1,\ldots,\infty}^{1,\ldots,\infty}\) and \(\{h^{1m}_{1m}\}_{k=1,\ldots,\infty}^{1,\ldots,\infty}\) such that:
\[
\sum_k \hat{\pi}_k \sum_i \lambda_i h_{im} = \sum_k \hat{\pi}_k \sum_i \lambda_i h_{im} - \sigma_{I_\eta_1},
\]
where \(\sigma_{I_\eta_1} = 1\) if \(n \in I_\eta\) and \(\sigma_{I_\eta_1} = -1\) otherwise. Similarly,
\[
\sum_k \hat{\pi}_k \sum_i \lambda_i b_{im} = \sum_k \hat{\pi}_k \sum_i \lambda_i h_{im} - \sigma_{I_\eta_1},
\]
where \(\sigma_{I_\eta_1} = 1\) if \(n \in I_\eta\) and \(\sigma_{I_\eta_1} = -1\) otherwise. Then, using Lemma 1 in \([17]\), we see that \(\alpha^*\) obtained by solving (34) satisfies that:
\[
\sum_n \hat{\alpha}_n^m \eta + \sum_n |\hat{\alpha}_n^m| \eta + \sum_m |\hat{\alpha}_m^h| \eta \leq V_{f_{\text{max}}},
\]
\[
\eta = \min(\eta_0, \eta_1).\]
Moreover, (87) also holds for \(\alpha^*\) and \(\hat{\alpha}^*\). Now consider \(\hat{g}(\alpha^*)\) and \(\hat{g}(\alpha^*)\). For explanation, we write \(\hat{g}(\alpha^*) = \hat{g}(\alpha^*, \gamma^*, b^*, \hat{R}^*, h^*)\), where \(\gamma^*, b^*, \hat{R}^*, h^*\) are the optimal actions corresponding to \(\alpha^*\) with \(r\) and the true distribution \(\pi\). From the definition, we know that:
\[
\hat{g}(\alpha^*, \gamma^*, b^*, \hat{R}^*, h^*)
\]
\[
\geq \tilde{g}(\alpha^*, \gamma^*, b^*, \hat{R}^*, h^*) + \sum_k \sum_n \hat{\alpha}_n^m [r_n(\hat{z}_k, \mu_n(\hat{z}_k, b^*_k)) - r_n(\hat{z}_k, \mu_n(\hat{z}_k, b^*_k))]
\]
\[
\geq \tilde{g}(\alpha^*, \gamma^*, b^*, \hat{R}^*, h^*) - V_{f_{\text{max}}} \delta_r/\eta.
\]
\[
\geq \hat{g}(\alpha^*, \gamma^*, b^*, \hat{R}^*, h^*) - V_{f_{\text{max}}} \delta_r/\eta.
\]
\[
\geq \hat{g}(\alpha^*, \gamma^*, b^*, \hat{R}^*, h^*) - 2V_{f_{\text{max}}} \delta_r/\eta.
\]
\[
\geq \hat{g}(\alpha^*, \gamma^*, b^*, \hat{R}^*, h^*) - 2V_{f_{\text{max}}} \delta_r/\eta.
\]
Here \(\gamma^*, b^*, \hat{R}^*, h^*\) are the actions corresponding to \(\alpha^*\) and \(\hat{g}(\alpha^*)\), and (a) follows from the definition of \(\hat{g}(\alpha^*)\) and the fact that \(\hat{g}(\alpha^*)\) achieves the supremum over all actions \(\gamma^*, b^*, \hat{R}^*, h^*\) for \(\alpha^*\) and \(\hat{g}(\alpha^*)\). In (b), we have used that \(\hat{g}(\alpha^*)\) achieves the minimum over all actions \(\alpha^*\), that the learning module guarantees that \(\|\hat{r} - r\|_{\text{max}} \leq \delta_r\), and (77). Finally, in (c), \(\gamma^*, b^*, \hat{R}^*, h^*\) are the actions corresponding to \(\alpha^*\) under \(\hat{g}(\alpha^*)\) and it follows again because \(\hat{g}(\alpha^*)\) achieves the supremum. The last inequality follows similarly to (b). Using the polyhedral structure of \(\hat{g}(\alpha^*)\), (89) implies that:
\[
\|\alpha^* - \tilde{\alpha}^*\| \leq 2\frac{V_{f_{\text{max}}}}{\rho_0} \delta_r.
\]
This proves (86). To prove (67), note that for any \(\alpha^*\),
\[
\hat{g}(\alpha^*) - \tilde{g}(\alpha^*) = \sum_k (\alpha_k - \tilde{\alpha}_k)g_k(\alpha^*),
\]
\[
\|\alpha^* - \tilde{\alpha}^*\| \leq 2\frac{V_{f_{\text{max}}}}{\rho_0} \delta_r.
\]
This implies that:
\[
\|\hat{g}(\alpha^*) - \tilde{g}(\alpha^*)\| \leq 2\frac{V_{f_{\text{max}}}}{\rho_0} \delta_r,
\]
for otherwise we have:
\[
\|\hat{g}(\alpha^*) - \tilde{g}(\alpha^*)\| \leq \|\hat{g}(\alpha^*)\| + \|\tilde{g}(\alpha^*)\| + \|\delta_r V_{f_{\text{max}}} \theta\| < 0,
\]
which contradicts with the fact that \(\alpha^*\) achieves the minimum of \(\hat{g}(\alpha^*)\). Using the polyhedral structure, we see that (67) follows. This completes the proof of the lemma. □

Longbo Huang

Longbo Huang received the Ph.D. degree in Electrical Engineering from the University of Southern California in August 2011. He then worked as a postdoctoral researcher in the Electrical Engineering and Computer Sciences department at University of California at Berkeley from July 2011 to August 2012. Since August 2012, Dr. Huang joined the Institute for Interdisciplinary Information Sciences (IIS) at Tsinghua University (Beijing, China) as an assistant professor. Dr. Huang was a visiting scholar at the LIDS lab at MIT in summer 2012 and summer 2014, at the EECES department at UC Berkeley in summer 2013. He was also a visiting professor at the Institute (88) of Network Coding at CUHK in winter 2012. Dr. Huang was selected into China’s Youth 1000-talent program in 2013. Dr. Huang’s current research interests are in the areas of learning and optimization, network control, data center networking, smart grid and mobile networks.