2019 IEEE Intelligent Transportation Systems Conference (ITSC)
Auckland, NZ, October 27-30, 2019

Load Balancing in Mobility-on-Demand Systems: Reallocation Via Parametric Control Using Concurrent Estimation

Rebecca M. A. Swaszek[1] and Christos G. Cassandras[1,2]
[1]Division of Systems Engineering, [2]Department of Electrical and Computer Engineering
Boston University, Boston, MA 02215, USA
E-mail:{swaszek,cgc}@bu.edu

Abstract— **Mobility-on-Demand (MoD) systems require load balancing to maintain consistent service across regions with uneven demand subject to time-varying traffic conditions. The load-balancing objective is to jointly minimize the fraction of lost user requests due to vehicle unavailability and the fraction of time when vehicles drive empty during load balancing operations. In order to bypass the intractability of a globally optimal solution to this stochastic dynamic optimization problem, we propose a parametric threshold-based control driven by the vehicles available in and en route to each region. This is still a difficult parametric optimization problem for which one often resorts to trial-and-error methods where multiple sample paths are generated through simulation. In contrast, this paper utilizes concurrent estimation methods to simultaneously construct multiple sample paths from a single nominal sample path. The performance of the parametric controller for intermediate size systems is compared to that of a simpler single-parameter controller, a state-blind static controller, a policy of no control, and a theoretically-derived lower bound. Simulation results show the value of state information in improving performance.**

I. Introduction

Autonomous Taxi (AT) systems are expected to make forays into the Mobility-on-Demand (MoD) transportation sector currently served by ride-sharing companies and traditional taxis. All these systems are comprised of a fleet of vehicles, service regions, and users with dynamic temporal-spatial demand patterns. Service providers of AT fleets will face myriad regulatory, safety, and management challenges. Fleet management of MoD systems literature considers how to efficiently route vehicles among regions to meet current and projected demand and determine the appropriate fleet size as to guarantee certain levels of system performance. In this paper, we focus on the crucial operational challenge of *proactive load balancing* which refers to the process of dynamically redistributing the fleet so as to maintain availability across service areas to meet future demand. There is a trade-off between satisfying customer requests and driving hours logged by empty vehicles performing load balancing operations.

Within *proactive* rebalancing there are two approaches: static controls based upon historical data and dynamic control based upon the current state of the system. A closed Jackson queueing network model with regards to the vehicles is used in [1] and [2]; the proposed controllers redistribute ATs by creating static predetermined "false" user demand rates to dispatch empty ATs from popular destination to popular origin regions regardless of the state of the system. To address this limitation, [1] proposes a time-driven controller to evenly redistribute the AT fleet among all regions at regular time intervals. The receding horizon controller introduced in [3] dynamically adjusts "false" user rates according to the state of the system. Greedy state-based controllers in [4] relocate available vehicles by considering the relative abundance of vehicles and waiting/expected customers in adjacent regions.

Other authors utilize a flow abstraction such that user demand rates induce vehicle "flows" between regions. In order to stop some regions from running out while others amass flow, a set of static controlled rates is proposed in [5] to redistribute the flow to ensure stability; [6] builds off the aforementioned work to minimize capital costs, operating costs, and passenger experience, i.e., wait time.

In this paper, we use a queueing network model akin to [1] with an objective to minimize the fraction of dropped user requests (due to AT unavailability) and the fraction of time ATs spend on load balancing operations driving empty between regions. The optimal control over an infinite horizon can be determined using dynamic programming, but the "curse of dimensionality" renders such solutions intractable for all but very small systems.

Our approach is to transform this intractable dynamic optimization problem into a manageable parametric optimization problem where the parameters are *thresholds* that direct redistribution based upon the relative quantity of available ATs in and en route to each region. These thresholds may be tuned to various demand patterns (rush-hour, high traffic, etc.). We propose two controllers: a time-driven single-parameter controller and an event-driven multi-parameter controller; the latter demonstrates superior performance but its parameters require more effort to tune. The contribution of this paper is to solve the proactive load balancing problem of a MoD system by formulating a threshold-based parametric optimization problem and using concurrent estimation methods [7],[8] to find well performing thresholds from a *single observed sample path* of the queueing network, thus bypassing the need for repeated trial-and-error. In addition, we derive a lower bound to assess the performance of our proposed threshold-based control.

*Supported in part by NSF under grants ECCS-1509084, DMS-1664644, CNS-1645681, by AFOSR under grant FA9550-19-1-0158, by ARPA-Es NEXTCAR program under grant DE-AR0000796 and by the MathWorks.

Section II introduces the MoD system model framework while Section III proposes two parametric threshold-based controllers. Section IV derives an average best possible performance lower bound and the parametric controllers' performance is demonstrated in Section V. Finally, Section VI concludes and highlights potential future work.

II. System Model

We model a MoD system as a closed Jackson queueing network of N nodes $\mathcal{N} = \{1, ..., N\}$ and m resources representing regions and vehicles, respectively, similar to the model in [1]. We focus on the load balancing of an urban autonomous taxi fleet and as such shall refer to vehicles and demands as ATs and requests, respectively.

Fig. 1 shows a region i which consists of a queue of available ATs. In order to capture time-varying demand, routing and service characteristics, we divide a finite time period $[0, T]$ into K intervals indexed by $k = 1, 2, ..., K$, each of length I. Thus, the user request rate $\lambda_{i,k}$ depends on interval $k = 1, 2, ..., K$. When a user request occurs, if there is an available idle AT, then the (user, AT) pair, denoted by $\times$ and $\square$, respectively, are joined and routed with probability $p_{i,j,k}$ (including intra-region trips $i = j$) to an infinite-capacity server $W_{i,j}$ with service rate $\mu_{i,j,k}$. This server captures the travel time from i to j. Upon arrival at region j, the pair is separated: the AT is routed with probability 1 to the idle AT queue in region j and the user exits the system. If there are no available ATs at the time of the user request event, the user exits the system and incurs a cost. A load balancing controller at each region, marked by $\diamondsuit$, routes (according to some control policy) an empty available AT to server $W_{i,j}$ destined for region j.

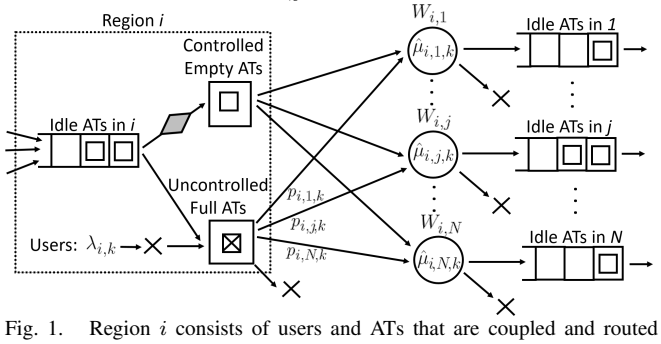

Fig. 1. Region i consists of users and ATs that are coupled and routed with probability $p_{i,j,k}$ to a infinite capacity server $W_{i,j}$, then uncoupled and the AT routed into region j. Idle ATs may also be forced by a decision $\diamondsuit$ to depart empty from the idle queue in i for another region j.

State Space: Let $x_i(t) \in \{0, 1, ..., m\}$ be the number of available idle ATs in region $i \in \mathcal{N}$ and $\boldsymbol{x}(t) = [x_1(t), ..., x_N(t)]$ be the idle AT vector. Let $y_{i,j}(t)$ be the number of full ATs (with passengers) en route from i to j and $\mathbf{Y}(t)$ be the corresponding $N \times N$ matrix populated by $y_{i,j}(t)$. Likewise, let $z_{i,j}(t)$ be the number of empty ATs in server $W_{i,j}$ and $\mathbf{Z}(t)$ be the corresponding $N \times N$ matrix populated by $z_{i,j}(t)$. Finally let $k(t) \in \{1, ..., K\}$ be the interval that specifies the user arrival rates $\lambda_{i,k}$, routing probabilities $p_{i,j,k}$, and service rates $\mu_{i,j,k}$ in effect at time t. Thus, the state of the system is $\mathscr{X}(t) = [\mathbf{x}(t), \mathbf{Y}(t), \mathbf{Z}(t), k(t)]$.

Events: The system dynamics are event-driven with the event set $E = E_U \cup E_C$ where E_U and E_C contain the uncontrollable and controllable events, respectively. We define the following uncontrollable event types within E_U:

- κ_k event: the start of the kth interval which prompts a change of $\lambda_{i,k}$, $p_{i,j,k}$, and $\mu_{i,j,k}$.
- $\delta_{i,j}$ event: a user request occurs for a trip departing from region i with destination j (with the possibility that $j = i$). Note that this request event does not necessitate an AT departure in the case that region i has no available ATs.
- $\alpha_{i,j}$ event: a full AT originating from i arrives at j.
- $\nu_{i,j}$ event: an empty AT originating from i arrives at $j \neq i$.

We also define the following controllable events in E_C:

- $\omega_{i,j}$ event: an empty AT (without passengers) departs from the idle AT queue in region i destined for region $j \neq i$.
- σ event: a timeout event used for time-driven control.

The control policy we select determines when controllable events are triggered. For example, a controllable $\omega_{i,j}$ event may be triggered by a timeout or the occurrence of an uncontrollable event resulting in the state of the system meeting certain criteria.

For a sample path of length T of the MoD system let $\mathbf{e} = \{e_1, ..., e_{Q_T}\}$ be the observed event sequence, $e_i \in E$, with corresponding event times $\boldsymbol{\tau} = \{\tau_1, ..., \tau_{Q_T}\}$ for a total of Q_T events in $[0, T]$. Only at these event times τ_q may the state of the system change. We may now write the state of the system at time τ_q as $\mathscr{X}(\tau_q) \equiv \mathscr{X}_q$, with $x_i(\tau_q) \equiv x_{i,q}$, $y_{i,j}(\tau_q) \equiv y_{i,j,q}$, $z_{i,j}(\tau_q) \equiv x_{i,j,q}$, for all $i \in \mathcal{N}$, where $q = 1, \ldots, Q_T$ is the asynchronous event counter.

Controls: A control action in this system consists of forcing an idle AT in some region i to travel empty to some other region $j \neq i$. This action depends upon the availability of idle ATs. Let $u_{i,j,q}(x_{i,q}) \in \{0, 1, ..., x_{i,q}\}$ be the number of empty ATs forced from i to j when the qth event occurs and let $\mathbf{U}_q(\mathbf{x}_q)$ be the $N \times N$ matrix populated by $u_{i,j,q}(x_{i,q})$. The following feasibility constraint is required for any control matrix $\mathbf{U}_q(\mathbf{x}_q)$:

$$\sum_{j=1}^{N} u_{i,j,q}(x_{i,q}) \leq x_{i,q} \quad \forall\, i \in \mathcal{N} \tag{1}$$

For simplicity of notation, let us drop the explicit control dependence on $x_{i,q}$ and write $u_{i,j,q}$. Note that such controls may be event-driven (deployed when the state of the system satisfies certain conditions) or time-driven via the controllable timeout event σ.

State Dynamics: The inventory $x_{i,q}$ of the idle AT queue in region i depends on both the uncontrollable and controllable events:

$$x_{i,q} = \begin{cases} x_{i,q-1} + 1 & e_q = \alpha_{j,i} \text{ or } e_q = \nu_{j,i} \\ \max\{x_{i,q-1} - 1, 0\} & e_q = \delta_{i,j} \\ x_{i,q-1} - u_{i,j,q} & e_q = \omega_{i,j} \\ x_{i,q-1} & \text{otherwise} \end{cases} \tag{2}$$

where $i, j \in \mathcal{N}$. Note that the $\max$ operation prevents the idle AT queue inventory from falling below 0 in the case that a $\delta_{i,j}$ user request event occurs and there are no idle ATs, i.e., when a user exits the system prematurely as in Fig. 1.

The number of full ATs $y_{i,j,q}$ en route from region i to region j evolves according to:

$$y_{i,j,q} = \begin{cases} y_{i,j,q-1} + 1 & e_q = \delta_{i,j} \text{ and } x_{i,q-1} > 0 \\ y_{i,j,q-1} - 1 & e_q = \alpha_{i,j} \\ y_{i,j,q-1} & \text{otherwise} \end{cases} \quad (3)$$

Likewise the number of empty ATs en route from i to j evolves according to:

$$z_{i,j,q} = \begin{cases} z_{i,j,q-1} + u_{i,j,q} & e_q = \omega_{i,j} \\ z_{i,j,q-1} - 1 & e_q = \nu_{i,j} \\ z_{i,j,q-1} & \text{otherwise} \end{cases} \quad (4)$$

At time instants $t = k\mathcal{I}$, $k \in \{1, \ldots, K\}$ the interval index k changes upon the occurrence of a κ_k event causing the time-varying parameters to change: $k_q = k_{q-1} + 1$

Objective Function: We formalize the trade-off in MoD fleet management of meeting user requests and minimize load balancing operations with a weighted sum of the probability that a user's request does not result in an AT departure (i.e., is rejected as in Fig. 1) and the probability that an AT is driving empty performing load balancing operations. The evaluation of these probabilities is generally infeasible because of the fast growth of the state space, rendering this task intracatable except for the simplest of systems. In order to assess the effect of controls $u_{i,j,q}$ on system performance, we replace these probabilities with estimates consisting of the fraction of rejected requests and total time spent driving empty in a sample path over $[0, T]$. Let $\boldsymbol{\rho}_T^i = \{\rho_{i,1}, ..., \rho_{i,D_T^i}\}$ be the event times of all $\delta_{i,j}$, $j \in \mathcal{N}$ user request events at region i where D_T^i is the total observed number of such events, and let $\mathbf{1}[\cdot]$ be the usual indicator function. As defined in **Events**, there are a total of Q_T events in $[0, T]$ and the qth event occurs at time τ_q.

The objective function we define is:

$$J(\mathscr{X}_0) = E\Bigg[w\frac{\sum_{i=1}^{N}\sum_{q=1}^{D_T^i}\mathbf{1}[x_i(\rho_{i,q}) = 0]}{\sum_{i=1}^{N} D_T^i} + (1-w)\frac{\sum_{q=1}^{Q_T}\sum_{i=1}^{N}\sum_{j=1}^{N}(\tau_q - \tau_{q-1})z_{i,j,q}}{Tm}\Bigg] \quad (5)$$

where $w \in (0, 1]$ is a weight coefficient. The first and second part of (5) refer to the fraction of rejected users unable to obtain an available AT (the numerator is the total number of all user request events) and the fraction of vehicle-hours ATs drive empty (the denominator is the total vehicle-hours driven by the m vehicles in the fleet over $[0, T]$). The weight coefficient w is used to quantify the trade-off between customer satisfaction and load balancing. We exclude $w = 0$ as the optimal control is trivial when customer satisfaction is irrelevant. If $w = 1$, the optimal control is still difficult to determine: although load balancing may not be a direct cost, the unavailability of ATs while performing load balancing operations creates an indirect cost. Note that the objective function in (5) is normalized so as to give values bounded by $[0, 1]$ and the weight w creates a convex combination of the two objective components.

The optimization problem we formulate based on (5) is to determine a control policy $\mathbf{U}(\mathscr{X})$ so as to minimize this objective:

$$J^*(\mathscr{X}_0) = \min_{\mathbf{U}(\mathscr{X})} J(\mathscr{X}_0) \quad (6)$$

III. Parametric Control Policies

The optimal control (6) may be found via dynamic programming using policy iteration (see [9]). However, such methods are limited to small systems as the state space grows combinatorially with the number of regions and ATs. For a system with N regions and m ATs, the cardinality of the state space is $\binom{m+2N^2-1}{2N^2-1}$ and the cardinality of states with at least $N-1$ controls is: $\binom{m+2N^2-1}{2N^2-1} - \binom{m+2N^2-N-1}{2N^2-N-1}$. An AT system with N=6 regions and m=50 taxis has over 3×10^{34} states. Utilizing sparse matrices and shared cloud computing facilities, the largest possible systems that we have been able to analyze are (N=2,m=10) and (N=3,m=5). The latter required only 31 GB of memory, while solving for N=3 regions and m=6 ATs required more than the 256 GB allotted to a full node of 28 cores. This provides motivation for seeking parametric control policies and assessing their performance using the fraction estimates in (5).

We introduce a parametric controller for larger MoDs expanding upon the framework of the "real-time" controller in [1]; this "real-time" controller rebalances ATs evenly among regions every half hour via an integer linear programming to minimize the expected load balancing time. Unlike our loss model in which user requests may be rejected, the model in [1] includes a queue for waiting users such that the number of ATs associated with a region is the sum of available ATs and ATs en route with the number of users queued up waiting for an AT subtracted from the latter.

Let $\Theta = [\theta_1, ..., \theta_N]$ be a parameter vector with $\theta_i \geq 0$, $i = 1, \ldots, N$ and $\sum_{i=1}^{N} \theta_i \leq m$ defining a "fill to" level for each of the N regions akin to (s, S) threshold policies in supply chain and inventory management [10] where s is a level that triggers a supply request and S triggers a request to stop the supply process. Note that each interval $k = 1, 2, ..., K$ may have its own set of parameters to account for different request patterns and traffic conditions. For simplicity of notation, let $a_i(t)$ be the total number of ATs available at or en route to region i at time t:

$$a_i(t) = x_i(t) + \sum_{j=1}^{N}\big(y_{j,i}(t) + z_{j,i}(t)\big) \quad (7)$$

Furthermore let us define a quantity $D_i(t)$ that is the *supply* of available excess ATs if positive or the *demand* for ATs if negative in region i : $D_i(t) = \min\{a_i(t) - \theta_i, x_i(t)\}$ Note that $D_i(t)$ is an integer quantity as $\theta_i, a_i(t)$, and $x_i(t)$ are all integers. In order for feasible AT redistribution actions to be triggered, it is a necessary condition that the overall supply must exceed the demand in the following inequality:

$$\sum_{k\in\{i\in\mathcal{N}|D_i(t)>0\}} D_k(t) \quad \geq \quad \sum_{j\in\{i\in\mathcal{N}|D_i(t)\leq 0\}} D_j(t) \quad (8)$$

This simply asserts that there is an adequate number of available ATs in regions which are above their "fill-to" levels specified in Θ which can be used to supply those regions whose queues of available ATs are below their "fill-to"levels.

Assuming for the moment that there exists a well-defined mechanism for triggering a process to redistribute ATs among regions (to be further discussed in this section), this

process consists of the following integer linear program with decision variables $u_{i,j} \in \{0, 1, 2, \ldots\} \forall i, j \in \mathcal{N}$:

$$\min_{u_{i,j},\ i,j\in\mathcal{N}} \sum_{i=1}^{N}\sum_{j=1}^{N} \frac{u_{i,j}}{\mu_{i,j}} \quad \text{s. t.} \quad \sum_{j=1}^{N} u_{i,j} \leq x_i(t) i \in \mathcal{N}$$
$$\theta_i \leq a_i(t) + \sum_{j=1}^{N} \left(u_{j,i} - u_{i,j}\right) i \in \mathcal{N} \tag{9}$$

The objective function of (9) minimizes empty vehicle driving time; the first constraint maintains feasibility and the second constraint requires the intended inventory for each region to meet or exceed "fill to" levels $\theta_1, ..., \theta_N$ and . In order to bypass the difficulty of integer programming, we rewrite (9) as a relaxed linear program in the form of a minimum cost flow problem in which regions indexed by i with positive *supply* $D_i(t)$ are sources and negative *demand* $D_i(t)$ are sinks with decision variable $u_{i,j} \geq 0 \quad \forall i, j \in \mathcal{N}$.

$$\min_{u_{i,j},\ i,j\in\mathcal{N}} \sum_{i=1}^{N}\sum_{j=1}^{N} \frac{u_{i,j}}{\mu_{i,j}} \tag{10}$$
$$\text{s. t.} \quad D_i(t) \geq \sum_{j=1}^{N} \left(u_{i,j} - u_{j,i}\right) \quad i \in \mathcal{N}$$

Note that the single constraint in (9) encompasses both constraints in (10): **1**, if $D_i(t) = a_i(t) - \theta_i$, the constraint in (10) is identical to the second constraint of (9); **2**, if $D_i(t) = x_i(t)$, then i is a source such that no flow is directed to it, i.e., $\sum_{j=1}^{N} u_{j,i} = 0$, therefore, the constraint in (10) becomes identical to the first constraint of (9). We recover the integer solution to (9) from this linear program since integer solutions are a property of minimum cost flow linear programs with integer sink and source quantities [11].

Depending upon the frequency at which these control actions are implemented, there could be empty ATs cycling inefficiently such as $z_{i,j}(t) > 0$ and $z_{j,i}(t) > 0$ simultaneously. To address this issue, we introduce an additional parameter Ω which acts as either a scalar "timeout" in a time-driven controller or as an integer threshold in an event-driven controller. In either case, Ω defines the mechanism that triggers solving (10) to determine the values of the control variables $u_{i,j}$, $i, j \in \mathcal{N}$.

Single Scalar Parameter Time-Driven Controller:

For a time-driven controller, let the scalar parameter $\Omega \in (0, \infty)$ be associated with the timeout event σ we defined in **Events** such that event σ triggers the control actions in (10) every Ω time units. The "real-time" controller proposed in [1] can be recovered from our controller by setting all $\theta_i = \lfloor \frac{m}{N} \rfloor$, $i \in \mathcal{N}$ in (9) and $\Omega = 30$ minutes. Note that this controller does not account for the system-specific demand rates; for example consider a heterogeneous two-region system – this control of distributing the fleet equally between them would perform badly if one region experienced far greater requests than the other.

N+1 Integer Parameter Event-Driven Controller:

To address the limitations of the aforementioned controller, we define an *event-driven* controller to trigger a solution of problem (10) whenever $a_i(t)$, the number of ATs in or en route to a region, drops below the threshold θ_i by some amount. In the simplest case, this occurs as soon as $a_i(t) < \theta_i$, corresponding to a greedy mechanism that pulls a single empty AT from the nearest region j with $a_j(t) > \theta_j$ and setting $u_{i,j} = 1$. However, since we have at our disposal a central controller with full information of the state space, we can do better than that as explained next.

Let us redefine $\Omega \in \{1, 2, .., m\}$ as an integer-valued threshold parameter used to trigger the control actions resulting from a solution of (10) whenever the condition $\theta_i - a_i(t) > \Omega$ is satisfied. This will effectively send a total of $\theta_i - a_i(t)$ empty ATs to region i. However, this is still a region-specific control that may be inefficient over the system as a whole. As such, we instead trigger the control when the sum-positive difference between the "fill-to" levels θ_i and available or en route ATs $a_i(t)$ surpasses threshold Ω *across all regions* for a global centralized control policy triggered by:

$$\sum_{j\in\{i\in\mathcal{N}|a_i(t)<\theta_i\}} \left(\theta_j - a_j(t)\right) \quad > \quad \Omega \tag{11}$$

For example, consider the state of a system with $N = 4$, $m = 20$ in Fig. 2 with control parameter vector $\Theta = [5, 3, 4, 5]$ is hashed in black and let $\Omega = 2$. After each event, the controller checks inequalities (8) and (11); if both hold, then $\omega_{i,j}$ events are induced as per (10). In this example, (8) holds with (0+1+0+2) $\leq$ (3+0+3+0) and (11) holds with (0+1+0+2) $<$ 2.

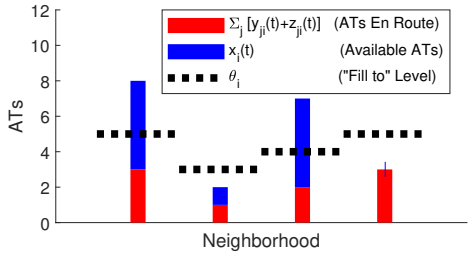

Fig. 2. Control parameters θ_i represent a "fill to" level for the number of ATs available at or en route to region i. Below this threshold represents need; above this threshold represents excess inventory to possibly send elsewhere.

This controller is triggered when one of the events defined in **Events** causes a change in the value of some $a_i(t)$ in (11) so that the sum crosses Ω either from below or from above. This may happen in two ways: **1** When event $\delta_{i,j}$ occurs, i.e., a user requests to go from region i to j, which alters $x_i(t)$ and $y_{i,j}(t)$ as per the state dynamics (2) and (3), respectively (note that this assumes (8) already holds). **2** Event $\alpha_{j,i}$ or $\nu_{j,i}$ occurs, i.e., a full (respectively, empty) AT arrives at region i and alters the value of $x_i(t)$ and $y_{j,i}(t)$ (respectively, $z_{j,i}(t)$). While the time-driven controller triggers control actions after a predetermined length of time regardless of the state of the system, the event-driven controller is only triggered when the inventory levels fall sufficiently low across all regions.

Concurrent Estimation:

Both the time and event-driven controllers require parameters be tuned to obtain the best performance attainable within their parametric family of load balancing controllers. The time-driven controller depends only on the scalar real-valued parameter Ω so that "brute-force" trial-and-error simulation can be used. However, such a "brute-force" method is

infeasible for the event-driven controller with its vector of $N+1$ tunable integer parameters, i.e., $\Theta = [\theta_1, ..., \theta_N]$ and $\Omega \in \{1, 2, .., m\}$ that gives rise to $\sum_{i=1}^{m} i\binom{i+N-1}{N-1}$ possible parameter vectors – a system of 6 regions and 50 ATs has over 1×10^9 parameter vectors. Therefore, we turn to Concurrent Estimation (CE) methods [8] which can construct multiple sample paths concurrently with an observed nominal sample path. Thus, we can simulate the system under a given setting (Θ_0, Ω_0) and, upon completion of this simulation run (or directly observed sample path in an actual MoD based on real data collected) over $[0, T]$, a large number $L \leq m \cdot \sum_{i=1}^{m} \binom{i+N-1}{N-1}$ of other feasible sample paths (Θ_l, Ω_l) are also available, $l = 1, \ldots, L$. The value of L (the number of concurrently estimated sample paths) is limited by memory storage as the computation time of constructing a single concurrent sample path is minimal (see [7],[8]).

The Markovian nature of the MoD model we have adopted allows us to make use of the Standard Clock (SC) CE method presented in [7],[8] with minor modifications. Details of the original Standard Clock method, as well as the variant we have adopted for our problem, are provided in [9]. Also included are the details of the optimization scheme to find the best control parameters. In short, we simulate a nominal sample path over $[0, T]$ with some starting parameter vector and use CE to concurrently simulate $2N$ sample paths, each of which uses a selected variation of the nominal parameter vector. At the end of the simulation we begin a new iteration by choosing the best performing sample path's parameter vector to use as the next nominal sample path's vector and create $2N$ more selected variations on this new nominal parameter vector. This greedy algorithm stops when the best performance of two successive iterations is less than some chosen ϵ (thus, there is no guarantee of global optimality).

N^2 Parameter Static Controller:

Both previously described controllers require tuning the parameters via simulation or through a data-driven on-line adaptation process. We shall compare in Section V these controllers to an alternative simpler parametric controller introduced in [5] whose parameters are determined by linear programming and rely solely on the model parameters $\lambda_{i,j}$ and $\mu_{i,j}$. This time-invariant and state-blind control sends empty ATs from i to j at a static rate $r_{i,j}$, $i, j \in \mathcal{N}$ such that there are N^2 rate parameters determined by the following linear program that minimizes empty travel time and seeks to equal the inflow and outflow of ATs at each region with $r_{i,j} \geq 0 \ \forall i, j \in \mathcal{N}$:

$$\min_{r_{i,j}} \sum_{i=1}^{N}\sum_{j=1}^{N} \frac{r_{i,j}}{\mu_{i,j}} \tag{12}$$

$$\text{s. t.} \sum_{j=1}^{N}(\lambda_{i,j} + r_{i,j}) = \sum_{j=1}^{N}(\lambda_{j,i} + r_{j,i}) \quad i \in \mathcal{N}$$

This linear program is always feasible; for example setting $r_{i,j} = \lambda_{j,i} \ \forall \ i, j \in \mathcal{N}$ satisfies the first constraint by sending back empty taxis at the same rate and satisfies the second constraint as $\lambda_{i,j} \geq 0$. Note that if this controller directs at time t an AT to leave region i for some j but $x_i(t) = 0$ (i.e., region i does not have any idle ATs) neither a penalty nor an event $\omega_{i,j}$ occur.

IV. System Performance Lower Bound

In order to assess the performance of the parametric controllers we establish a lower bound on best performance possible *on average*.As the objective function in (5) is an average over $[0, T]$, we seek only an average performance lower bound. Let us abstract the request arrival process into a continuous request flow process with rate $\lambda_{i,j}$. On average, region i will have an inflow of $\sum_{j=1}^{N} \lambda_{j,i}$ and an outflow of $\sum_{j=1}^{N} \lambda_{i,j}$ with an average difference in request flow: $d_i = \sum_{j=1}^{N}(\lambda_{i,j} - \lambda_{j,i})$ such that we may define the following two sets of regions depending on the sign of d_i:

$\mathcal{G} = \{i \in \mathcal{N} | d_i < 0\} \qquad \mathcal{B} = \{i \in \mathcal{N} | d_i \geq 0\}$

There are two sources of system costs in the objective function (5): unsated user requests for ATs and empty AT traveling time. Regions within $\mathcal{B}$ will run out of AT flow as they are a more popular origin than destination, thus they will reject a request flow d_i. For this abstracted flow AT system an amount p of flow lost by not being sated with available ATs costs the system $c(p)$ as defined by:

$$c(p) = p \cdot 1 / \sum_{i=1}^{N}\sum_{j=1}^{N} \lambda_{i,j} \tag{13}$$

Regions within $\mathcal{B}$ will build up excess AT flow and are candidates to send empty AT flow out. Suppose that p empty ATs are sent from i to j at the beginning of a time period, with an average trip time $\frac{1}{\mu_{i,j}}$ for a total mean empty AT driving time $\frac{p}{\mu_{i,j}}$. As there are a total of m AT-hours, the mean number of empty ATs driving is $\frac{p}{\mu_{i,j} m}$. Similarly, for the abstracted fluid AT system forcing an AT flow of p from i to j, the system cost incurred is as follows; note that both (13) and (14) are linear functions of the flow p.

$$C(p, i, j) = p \cdot 1 / \mu_{i,j} m \tag{14}$$

Consider two types of decision variables: $\beta_j, \ j \in \mathcal{B}$, as the fraction of positive request difference d_j that will be left unsated and $v_{i,j}$ as the forced empty AT flow from $i \in \mathcal{G}$ to $j \in \mathcal{B}$ that will sate the remaining $[1 - \beta_j]$ fractional difference in flow. The following linear program finds a lower bound with less than $N^2 + N$ decision variables. The first and second parts of the objective function are the cost of ignoring a fraction β_j and sating the fraction $[1 - \beta_j]$ of difference in request d_j from (13) and (14), respectively. The first constraint requires that the fraction $[1 - \beta_j]$ of difference in request is sated in "bad" regions. The second constraint places limitations on the empty AT flows from "good" regions.

$$\text{LB} = \min_{v_{i,j}, \beta_j} \sum_{j \in \mathcal{B}} \left(\frac{d_j \beta_j}{\sum_{i=1}^{N}\sum_{k=1}^{N} \lambda_{i,k}} + \sum_{i \in \mathcal{G}} \frac{v_{i,j}}{\mu_{i,j} m} \right) \tag{15}$$

$$\text{s. t.} \qquad d_j(1 - \beta_j) = \sum_{i \in \mathcal{G}} v_{i,j} \qquad j \in \mathcal{B}$$

$$- d_i \geq \sum_{j \in \mathcal{B}} v_{i,j} \qquad i \in \mathcal{G}$$

$$0 \leq v_{i,j} \quad i \in \mathcal{G}, j \in \mathcal{B} \qquad 0 \leq B_j \leq 1 \quad j \in \mathcal{B}$$

V. Simulation Example: A 6-Region System

Consider an N=6 MoD system with an objective function weight w=0.5 (whose request and travel rates may be found in [9]). In order to assess the performance of our event-driven parametric controller, we compare it to the time-driven controller, the static controller in [5], the lower bound derived in (15), and a policy of no control.

The event-driven and time-driven controllers require $N+1$ integer and 1 real-valued parameters respectively, which are determined using the concurrent estimation techniques described in detail in [9] applied to a simulated system. After running many iterations on a shared cloud computer cluster in MATLAB 2018b, the event and time-driven parameters in Table I were determined to perform the best for fleet sizes: []50, 75, 100, 125].

TABLE I

Event and Time-Driven Controller Parameters: N=6 System

Control	**Event-Driven**							**Time-Driven**
m	θ_1	θ_2	θ_3	θ_4	θ_5	θ_6	Ω	Ω
50	10	7	4	1	4	7	5	24
75	15	13	8	4	12	13	8	12
100	20	16	11	7	16	20	14	12
125	27	19	13	7	19	25	22	18

As these parameter vectors were determined to be the best via simulation, they are likely local minima of the objective function. Iterations stopped when slight deviations (i.e. $\theta_i' = \theta_i + 1$) had little effect on the objective function of a sufficiently long sample path (T=100,000 time units).

Fig. 3 shows the average simulated performance for all fleet sizes and controllers. The N^2 parameters of the static false rate controller introduced in [5] may be found in [9]. All systems performed about the same under no control with about 37% of user requests unsated – as the fleet size increased more vehicles sat idle in unpopular origin regions. As expected, the N+1 parameter event-driven controller with its state-dependent control and system-specific tuned parameters performs the best across all fleet sizes.

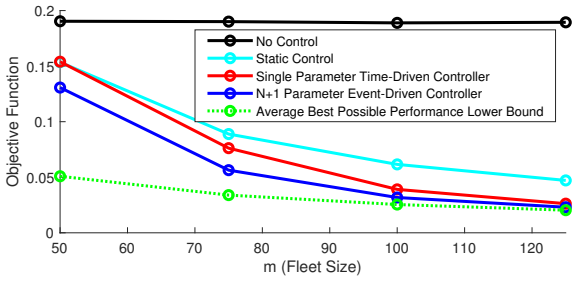

Fig. 3. The performance of a system with fleet sizes 50, 75, 100, and 125 under no control, static control, time-driven control, event-driven control, and the average best possible performance lower bound.

The intermediate fleet sizes studied here are where the true benefits of the event-driven controller are to be observed for this particular 6-region system. Note that the event and time-driven controllers quickly approach the lower bound as the fleet size increases but the static controller does not– this advantage is due to the event and time-driven controllers use of state information. We ignore fleets of less than 50: they are unstable and perform poorly no matter the control, since the underlying MoD system is under-capacitated with an insufficient number of vehicles to satisfy the given demand. We also ignore fleets over 125: they perform well no matter the control, as they are over-capacitated. Table II shows a detailed performance comparison of the controllers and the lower bound derived in (15) for a fleet size of 75 ATs.

TABLE II

N=6, m=75 System Performance Comparison

Control	None	Static	Time Driven	Event Driven	Lower Bound
% Users Rejected	38.0	11.8	7.0	3.4	0
% ATs Drive Empty	0	6.0	8.2	7.8	6.8
J $(w = 0.5)$	19.0	8.9	7.6	5.6	3.4

VI. Conclusions and Future Work

Load balancing in MoD systems requires empty vehicles be sent empty to mitigate the temporal demand patterns that deplete some service regions of available vehicles. We have defined an objective function to jointly minimize the fraction of user requests denied due to unavailability and the fraction of time vehicles drive empty– and derived its lower bound. As optimal control via DP quickly becomes intractable even for small dimensionality systems, we have developed a parametric controller using thresholds on the number of vehicles available in and en route to each region. Well performing parameters are determined using Concurrent Estimation methods which allow for the construction of multiple sample paths under different parameters from a single nominal sample path. Simulations show the proposed event-driven threshold-based controller performs significantly better than static controllers and approaches the lower bound for large fleet sizes. Future work will include using taxi data and exploring a wider range of the control parameter space by making a more efficient use of CE methods.

References

[1] R. Zhang and M. Pavone, "Control of robotic mobility-on-demand systems: a queueing-theoretical perspective," *The International Journal of Robotics Research*, vol. 35, no. 1-3, pp. 186–203, 2016.

[2] K. Spieser, K. Treleaven, R. Zhang, E. Frazzoli, D. Morton, and M. Pavone, "Toward a systematic approach to the design and evaluation of automated mobility-on-demand systems: A case study in singapore," in *Road Vehicle Automation*. Springer, 2014, pp. 229–245.

[3] M. Ramezani and M. Nourinejad, "Dynamic modeling and control of taxi services in large-scale urban networks: A macroscopic approach," *Transportation Research Part C: Emerging Technologies*, vol. 94, pp. 203–219, 2018.

[4] D. J. Fagnant and K. M. Kockelman, "The travel and environmental implications of shared autonomous vehicles, using agent-based model scenarios," *Transportation Research Part C: Emerging Technologies*, vol. 40, pp. 1–13, 2014.

[5] M. Pavone, S. L. Smith, E. Frazzoli, and D. Rus, "Robotic load balancing for mobility-on-demand systems," *The International Journal of Robotics Research*, vol. 31, no. 7, pp. 839–854, 2012.

[6] K. Spieser, S. Samaranayake, and E. Frazzoli, "Vehicle routing for shared-mobility systems with time-varying demand," in *American Control Conference (ACC), 2016*. IEEE, 2016, pp. 796–802.

[7] C. G. Cassandras and C. G. Panayiotou, "Concurrent sample path analysis of discrete event systems," *Discrete Event Dynamic Systems*, vol. 9, no. 2, pp. 171–195, 1999.

[8] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*. Springer Science & Business Media, 2009.

[9] R. Swaszek and C. Cassandras, "Load balancing in mobility-on-demand systems: Reallocation via parametric control using concurrent estimation," *arXiv preprint arXiv:1904.03755*, 2019.

[10] D. P. Bertsekas, *Dynamic programming and optimal control*. Athena scientific Belmont, MA, 2005, vol. 1, no. 3.

[11] M. S. Bazaraa, *Linear programming and network flows*, fourth edition.. ed. Hoboken, New Jersey: Wiley, 2010.