Asynchronous Scheduling for Energy Optimality in Systems with Multiple Servers

Michael J. Neely
University of Southern California
http://www-bcf.usc.edu/~mjneely

***Abstract*— We consider energy-aware scheduling in a multi-server system with N classes of jobs. Jobs arrive randomly and are queued according to their class. Servers operate asynchronously over their own timelines. Each server can be in either the active state or the idle state. At the beginning of each active period, a server chooses a *processing mode* from a collection of options that affect: (i) which classes of jobs are served, (ii) the service times, and (iii) the energy incurred. After processing, the server chooses an amount of time to remain idle. The goal is to make decisions over time that minimize time average power subject to stabilizing all queues. This is related to a non-convex optimization problem with fractional terms that have different denominators in the objective function and in the constraints. Such problems are generally intractable. However, the system has physical properties with special structure. Exploiting these properties, we develop a novel online algorithm that solves the problem. The algorithm does not require knowledge of the arrival rates. It can push time average power arbitrarily close to optimal, with a corresponding tradeoff in average queue size.**

I. Introduction

We consider a processing system with N classes of jobs and S servers. Jobs of each class arrive randomly and are stored in separate queues to await service. Servers act asynchronously over variable length frames. At the beginning of the rth frame for server s, the server chooses a *processing mode* $m_s[r]$ from a collection of mode options. The processing mode determines the number of jobs of each class that it serves on frame r, the duration of time required for service, and the resulting energy expenditure. After processing, the server chooses an amount of *idle time* $I_s[r]$. The server is put to sleep during this time to save power. The chosen idle time can be 0, so that the server can have back-to-back busy periods. After the (possibly 0) idle time, the server wakes up and a new frame is begun. An example timeline for one particular server s is shown in Fig. 1.

Each server makes these decisions separately according to its own timeline. That is, the service frame boundaries for each server are not synchronized, and different servers can start and end their frames at different times (see Fig. 2). For example, one server might finish three frames before another server finishes one. The servers can be heterogenous, having different mode selection options with different energy and delay performance. Further, some servers might be restricted from serving certain classes of jobs. The goal is to design an algorithm for dynamically making decisions at each server to minimize total average power expenditure subject to stabilizing each of the N queues.

This material is supported in part by one or more of: the NSF Career grant CCF-0747525, the Network Science Collaborative Technology Alliance sponsored by the U.S. Army Research Laboratory W911NF-09-2-0053.

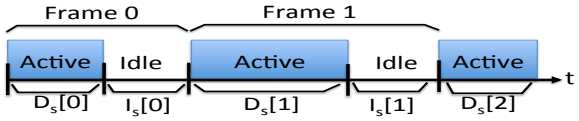

Fig. 1. The timeline for one server s.

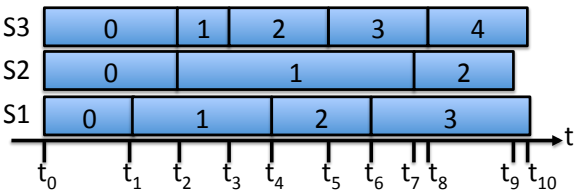

Fig. 2. Asynchronous timelines for 3 servers $S1$, $S2$, and $S3$.

This problem is important for energy-aware computing in systems with multiple processors. For example, work in [1][2] shows that processors can use methods such as frequency and voltage scaling to adjust service times and energies. There is much recent interest in developing scheduling algorithms that exploit these capabilities (see, for example, [3][4][5][6][7][8]). However, this prior work uses either single servers or synchronized service decisions, often with fixed-length timeslots.

The problem in the current paper is challenging because it involves a coupling between service decisions that are made over multiple timelines. The decisions at a particular server cannot be changed until the next frame begins for that server. This can be viewed as a system with *multiple embedded Markov chains* that are coupled through the common objective of supporting all traffic while using minimum average power. However, an attempt to use Markov chain theory reveals a seemingly intractable state space. Indeed, even modeling the active/idle state of each server involves 2^S states. Further, this exponential state space is still an incomplete description because it does not include the residual amount of time for each server to reach the end of its current frame.

We show that the problem can be viewed more simply in terms of an optimization involving ratios of averages of system quantities. The problem is non-linear and non-convex. However, it has a special structure that we exploit. Using this structure, we develop a simple dynamic algorithm to solve the problem. The algorithm stabilizes all queues whenever possible. Further, it is parameterized by a constant $V > 0$, and yields average power within $O(1/V)$ of the minimum average power required for stability, with an average queue

size tradeoff that is $O(V)$. That is, average power can be pushed arbitrarily close to optimal, with a tradeoff in average queue size. The algorithm suffers no curse of dimensionality, has polynomial delays and convergence times, and can be implemented online in a distributed manner at each server. The algorithm does not require knowledge of the job arrival rates, and naturally adapts if these rates change.

Our recent work [8] considers the single-server version of this problem, using the theory of Lyapunov optimization for renewal systems from [9]. The single-server case is related to finding an online solution to a linear fractional program. The current paper treats the multi-server case, where servers operate asynchronously over multiple timelines. This results in more complex optimization problems with fractional terms that have different denominators in both the objective function and in the constraints. The asynchronous operation treated in this paper is conceptually related to asynchronous optimization routines in [10][11] that treat different (and typically convex) problems. While [11] treats a network of queues, it still assumes all decisions are made according to a common timeline of fixed length slots, which cannot capture the fundamental complexities of the asynchronous behavior in our model.

Two aspects of our solution draw from techniques developed for restless bandit and multi-armed bandit systems. First, this paper uses a drift-plus-penalty ratio technique similar to a technique developed in [12][13] for analyzing restless bandit systems. Second, our solution requires penalties to be viewed as being continuously accumulated over time via integration of a penalty rate function. This is inspired by a similar treatment of rewards for multi-armed bandit problems in [14].

II. Problem Formulation

Consider a system with N classes of jobs and S server stations. Jobs of each class arrive randomly according to independent Poisson processes with rates $\lambda_1, \ldots, \lambda_N$. These jobs are stored in separate queues according to their class. Let $Q_n(t)$ represent the number of class n jobs queued at time t. The system operates in continuous time, and so the time index t takes values in the set of non-negative real numbers. The value of $Q_n(t)$ is a non-negative integer for all $n \in \{1, \ldots, N\}$ and for all $t \geq 0$. Assume the system is initially empty at time $t = 0$, so that $Q_n(0) = 0$ for all $n \in \{1, \ldots, N\}$.

Each server $s \in \{1, \ldots, S\}$ makes decisions over *renewal frames* (see Fig. 1). The first frame is labeled as frame 0 and starts at time 0. Successive frames can start at different times for each server. Consider the frames for a particular server s. At the beginning of each frame $r \in \{0, 1, 2, \ldots\}$, server s chooses a *processing mode* $m_s[r]$ within a set $\mathcal{M}_s$ of mode options. The sets $\mathcal{M}_s$ can be different for each $s \in \{1, \ldots, S\}$, and are possibly infinite. The processing mode $m_s[r]$ determines values $\mu_{sn}[r]$ for each $n \in \{1, \ldots, N\}$, representing the number of class n jobs that can be served by s on frame r. It also determines the duration of time $D_s[r]$ required for service, and the processing energy $e_s^{proc}[r]$ that is incurred. These are given by general functions of $m_s[r]$:

$$\begin{aligned} \mu_{sn}[r] &= \hat{\mu}_{sn}(m_s[r]) \\ D_s[r] &= \hat{D}_s(m_s[r]) \\ e_s^{proc}[r] &= \hat{e}_s^{proc}(m_s[r]) \end{aligned}$$

If server s is restricted from serving a certain class b, then $\hat{\mu}_{sb}(m_s) = 0$ for all $m_s \in \mathcal{M}_s$. At the end of the processing time, server s chooses an *idle time* $I_s[r]$ within an interval $[0, I_s^{max}]$, for some given non-negative value I_s^{max}. Assume the idle state expends energy at a rate p_s^{idle} for some constant $p_s^{idle} \geq 0$. Let $e_s[r]$ be the total energy expended by server s on frame r, and let $T_s[r]$ be the total frame size. These are given by functions $\hat{e}_s(m_s[r], I_s[r])$ and $\hat{T}_s(m_s[r], I_s[r])$:

$$e_s[r] = \hat{e}_s(m_s[r], I_s[r]) \triangleq \hat{e}^{proc}(m_s[r]) + p_s^{idle} I_s[r] \quad (1)$$

$$T_s[r] = \hat{T}_s(m_s[r], I_s[r]) \triangleq \hat{D}_s(m_s[r]) + I_s[r] \quad (2)$$

where the symbol "$\triangleq$" represents "defined to be equal to."

A. Mode Selection Assumptions

Assume that all service times are bounded by constants D_s^{min} and D_s^{max}, where $0 < D_s^{min} \leq D_s^{max}$, so that:

$$D_s^{min} \leq \hat{D}_s(m_s) \leq D_s^{max} \ \forall m_s \in \mathcal{M}_s$$

All frame sizes $T_s[r]$ are thus bounded, so that:

$$D_s^{min} \leq T_s[r] \leq D_s^{max} + I_s^{max} \ \forall r \in \{0, 1, 2, \ldots\}$$

Further assume the number of jobs served and the energy expenditures are upper bounded by constants μ_{sn}^{max} and e_s^{max}:

$$\begin{aligned} 0 \leq \hat{\mu}_{sn}(m_s) \leq \mu_{sn}^{max} \ \ \forall m_s \in \mathcal{M}_s \\ 0 \leq \hat{e}_s(m_s) \leq e_s^{max} \ \ \forall m_s \in \mathcal{M}_s \end{aligned}$$

Finally, assume that each set $\mathcal{M}_s$ contains an option $null_s$ where no jobs are served, so that:

$$\hat{\mu}_{sn}(null_s) = 0 \ , \ \hat{D}_s(null_s) = D_s^{min} \ \forall n \in \{1, \ldots, N\}$$

This is useful at times such as $t = 0$ when the system is empty.

B. Queueing

Define $R_s(t)$ as the number of full frames that server s has completed up to time t. This is an integer-valued function that is non-decreasing, has initial value $R_s(0) = 0$, and increases by 1 upon completion of each frame in server s. Define $R(t)$ as the sum process:

$$R(t) = \sum_{s=1}^{S} R_s(t)$$

The function $R(t)$ is also non-decreasing and integer valued. It can increase by more than 1 if multiple servers complete their frames at the same time. The intervals over which $R(t)$ is constant are called *sub-frames*. Define t_k as the start of the kth sub-frame, where $k \in \{0, 1, 2, \ldots\}$, and where $t_0 = 0$. Each time $t_k > 0$ marks a time when at least one server $s \in \{1, \ldots, S\}$ finishes a renewal frame and begins another (see Fig. 2). Define δ_k as the size of sub-frame k:

$$\delta_k \triangleq t_{k+1} - t_k \ \forall k \in \{0, 1, 2, \ldots\}$$

The queuing process $Q_n(t)$ is integer valued and decreases in discrete jumps immediately when packets are selected for service. Now define a related *non-integer* process $\tilde{Q}_n(t)$ that can be viewed as *unfinished work*, with dynamics:

$$\tilde{Q}_n(t_{k+1}) = \max[\tilde{Q}_n(t_k) - b_n[t_k, t_{k+1}) + a_n[t_k, t_{k+1}), 0] \quad (3)$$

where $a_n[t_k, t_{k+1})$ is the number of arrivals to queue n during the interval $[t_k, t_{k+1})$, and $b_n[t_k, t_{k+1})$ is the *fractional amount of service* offered during this interval. This fractional amount of offered service is computed by multiplying the interval size δ_k by the sum of the *effective service rates* of all servers serving class n jobs during this interval. Specifically, the effective service rate $\gamma_{sn}(t_k)$ allocated by server s to class n jobs at time t_k is defined as the ratio $\mu_{sn}[r]/T_s[r]$, where $r = R_s(t_k)$ is the server s frame that overlaps with time t_k:

$$\gamma_{sn}(t_k) \triangleq \frac{\mu_{sn}[R_s(t_k)]}{T_s[R_s(t_k)]} = \frac{\hat{\mu}_{sn}(m_s[R_s(t_k)])}{\hat{T}_s(m_s[R_s(t_k)], I_s[R_s(t_k)])}$$

For simplicity of notation, define m_{sk} and I_{sk} as the decisions made by server s for the frame that overlaps with time t_k:

$$m_{sk} \triangleq m_s[R_s(t_k)] \quad , \quad I_{sk} \triangleq I_s[R_s(t_k)] \quad (4)$$

The $b_n[t_k, t_{k+1})$ values are then defined:

$$b_n[t_k, t_{k+1}) \triangleq \delta_k \sum_{s=1}^{S} \frac{\hat{\mu}_{sn}(m_{sk})}{\hat{T}_s(m_{sk}, I_{sk})} \quad (5)$$

Each queue can easily update its $\tilde{Q}_n(t_k)$ values via (3). Further, it can be shown that $\tilde{Q}_n(t_k)$ closely tracks the actual queue value $Q_n(t_k)$, so that for all t_k we have:

$$\tilde{Q}_n(t_k) - \sum_{s=1}^{S} \mu_{sn}^{max} \leq Q_n(t_k) \leq \tilde{Q}_n(t_k) + \sum_{s=1}^{S} \mu_{sn}^{max}$$

Therefore, stabilizing the $\tilde{Q}_n(t_k)$ processes ensures that all actual queue processes are also stable. Formally, we say that $\tilde{Q}_n(t_k)$ is *mean rate stable* if $\lim_{k\to\infty} \mathrm{E}[\tilde{Q}_n(t_k)]/k = 0$.

C. The Effective Penalty Rate

Define the *effective penalty rate* for server s on frame r to be the ratio of the total energy incurred on the frame to the total frame size: $e_s[r]/T_s[r]$. Define *penalty rate function* $p_s(t)$ as the effective penalty rate associated with server s during the current frame:

$$p_s(t) \triangleq e_s[R_s(t)]/T_s[R_s(t)]$$

The accumulated penalty due to server s up to time t is thus:

$$\int_0^t p_s(\tau)d\tau$$

Because $p_s(t)$ is constant over any frame for server s, the integral of $p_s(t)$ over the duration of a full frame r is equal to the energy $e_s[r]$ expended over that frame. If server s begins a new frame at time t_k, then the accumulated penalty up to time t_k is the sum of the energies expended on each frame:

$$\int_0^{t_k} p_s(\tau)d\tau = \sum_{r=0}^{R_s(t_k)-1} e_s[r]$$

D. Time Averages

Consider an algorithm that makes decisions $m_s[r], I_s[r]$ for each $s \in \{1, \ldots, S\}$ and each frame $r \in \{0, 1, 2, \ldots\}$. For simplicity in this subsection, assume that, with probability 1, the algorithm yields well defined frame averages $\overline{e}_s$, $\overline{T}_s$, $\overline{\mu}_{sn}$, as defined below:

$$\overline{e}_s \triangleq \lim_{R\to\infty} \frac{1}{R} \sum_{r=0}^{R-1} e_s[r] \quad (6)$$

$$\overline{T}_s \triangleq \lim_{R\to\infty} \frac{1}{R} \sum_{r=0}^{R-1} T_s[r] \quad (7)$$

$$\overline{\mu}_{sn} \triangleq \lim_{R\to\infty} \frac{1}{R} \sum_{r=0}^{R-1} \mu_{sn}[r] \quad (8)$$

Let R be a positive integer. The empirical average power expended by server s over the first R frames is equal to the total energy expenditure divided by the total time:

$$\frac{\sum_{r=0}^{R-1} e_s[r]}{\sum_{r=0}^{R-1} T_s[r]} = \frac{\frac{1}{R}\sum_{r=0}^{R-1} e_s[r]}{\frac{1}{R}\sum_{r=0}^{R-1} T_s[r]}$$

Taking a limit as $R \to \infty$ gives the following expression for time average power at server s:

$$\lim_{R\to\infty} \frac{\sum_{r=0}^{R-1} e_s[r]}{\sum_{r=0}^{R-1} T_s[r]} = \frac{\overline{e}_s}{\overline{T}_s}$$

That is, the time average power (energy per unit time) is equal to the ratio of the average energy per frame to the average frame size. This simple observation is often used in renewal-reward theory [15].

Similarly, the time average rate that server s serves class n jobs (in jobs per unit time) is equal to:

$$\lim_{R\to\infty} \frac{\sum_{r=0}^{R-1} \mu_{sn}[r]}{\sum_{r=0}^{R-1} T_s[r]} = \frac{\overline{\mu}_{sn}}{\overline{T}_s}$$

Because all queues must be stabilized, the total time average rate of serving each queue n must be at least λ_n. Thus, the minimum power required to stabilize the system is defined by the following problem that optimizes ratios of time averages:

$$\begin{aligned} \text{Minimize:} \quad & \textstyle\sum_{s=1}^{S} \frac{\overline{e}_s}{\overline{T}_s} \\ \text{Subject to:} \quad & \textstyle\sum_{s=1}^{S} \frac{\overline{\mu}_{sn}}{\overline{T}_s} \geq \lambda_n \ \ \forall n \in \{1, \ldots, N\} \\ & m_s[r] \in \mathcal{M}_s \ \ \forall s \in \mathcal{S}, \forall r \in \{0, 1, \ldots\} \\ & 0 \leq I_s[r] \leq I_s^{max} \ \ \forall s \in \mathcal{S}, \forall r \in \{0, 1, \ldots\} \end{aligned}$$

where we define $\mathcal{S} \triangleq \{1, \ldots, S\}$.

Because the $e_s[r]$, $T_s[r]$, $\mu_{sn}[r]$ values are deterministically bounded for all r, if the limits (6)-(8) hold with probability 1, the *bounded moment convergence theorem* ensures they are the same as their frame average expectations [8]. That is:

$$\lim_{R\to\infty} \frac{1}{R} \sum_{r=0}^{R-1} \mathbb{E}\left[e_s[r]\right] = \overline{e}_s$$

Similarly, the frame average expectations of $T_s[r]$ and $\mu_{sn}[r]$ are equal to $\overline{T}_s$ and $\overline{\mu}_{sn}$, respectively. This motivates our use of frame average expectations in later sections.

III. Optimality over Randomized Algorithms

For each $s \in \{1, \ldots, S\}$, define the following vector-valued function:

$$\boldsymbol{v}_s(m_s, I_s) = (\hat{e}_s(m_s, I_s), T_s(m_s, I_s), \mu_{s1}(m_s, I_s), \ldots, \mu_{sN}(m_s, I_s))$$

Define $\mathcal{A}_s$ as the set of all vectors $(e_s, T_s, \mu_{s1}, \ldots, \mu_{sN})$ that can be achieved as vectors $\boldsymbol{v}_s(m_s, I_s)$ for some m_s, I_s that satisfy $m_s \in \mathcal{M}_s$, $0 \leq I_s \leq I_s^{max}$. That is, $\mathcal{A}_s$ is the set of all vectors that can be achieved over one frame, considering all decision options available to server s. Define $\overline{Conv(\mathcal{A}_s)}$ as the closure of the convex hull of set $\mathcal{A}_s$, being a closed and bounded set. This set can be viewed as the closure of the set of all expected vectors achievable in one frame by a randomized choice over the control decisions.

Now define Λ as the closure of the set of all (non-negative) rate vectors $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_N)$ for which there exists an algorithm that makes all queues mean rate stable. For each $\boldsymbol{\lambda} \in \Lambda$, define $p^{opt}(\boldsymbol{\lambda})$ as the infimum time average power expenditure achievable over the class of algorithms that make all queues mean rate stable when the arrival vector is $\boldsymbol{\lambda}$.

Theorem 1: A non-negative rate vector $\boldsymbol{\lambda}$ is in the set Λ if and only if there exist vectors:

$$(e_s, T_s, \mu_{s1}, \ldots, \mu_{sN}) \in \overline{Conv(\mathcal{A}_s)}$$

for each $s \in \{1, \ldots, S\}$ such that:

$$\sum_{s=1}^{S} \frac{\mu_{sn}}{T_s} \geq \lambda_n \ \forall n \in \{1, \ldots, N\}$$

Theorem 2: For any $\boldsymbol{\lambda} \in \Lambda$, the value $p^{opt}(\boldsymbol{\lambda})$ is equal to the solution of the following problem:

$$\text{Minimize:} \quad \sum_{s=1}^{S} \frac{e_s}{T_s} \tag{9}$$

$$\text{Subject to:} \quad \sum_{s=1}^{S} \frac{\mu_{sn}}{T_s} \geq \lambda_n \ \forall n \in \{1, \ldots, N\} \tag{10}$$

$$(e_s, T_s, \mu_{s1}, \ldots, \mu_{sN}) \in \overline{Conv(\mathcal{A}_s)} \ \forall s \in \mathcal{S} \tag{11}$$

The proofs of Theorems 1 and 2 are omitted for brevity (see [9] for a related result).

A. Relation to Fractional Programming

Suppose $\mathcal{M}_s$ has a finite number of options for each $s \in \{1, \ldots, S\}$. Then $\mathcal{A}_s$ is closed and bounded, as is $Conv(\mathcal{A}_s)$. It can be shown that the problem (9)-(11) is equivalent to finding values $q_s(m_s)$ for each $m_s \in \mathcal{M}_s$, representing the probability that server s chooses option m_s, together with constants $I_s \in [0, I_s^{max}]$, to solve:

$$\text{Minimize:} \quad \sum_{s=1}^{S} \left[\frac{p_s^{idle} I_s + \sum_{m_s \in \mathcal{M}_s} q_s(m_s) \hat{e}_s^{proc}(m_s)}{I_s + \sum_{m_s \in \mathcal{M}_s} q_s(m_s) \hat{D}_s(m_s)} \right]$$

$$\text{Subject to:} \quad \sum_{s=1}^{S} \left[\frac{\sum_{m_s \in \mathcal{M}_s} q_s(m_s) \hat{\mu}_{sn}(m_s)}{I_s + \sum_{m_s \in \mathcal{M}_s} q_s(m_s) \hat{D}_s(m_s)} \right] \geq \lambda_n \quad \forall n \in \{1, \ldots, N\}$$

$$q_s(m_s) \geq 0 \ \forall s \in \mathcal{S}, m_s \in \mathcal{M}_s$$

$$\sum_{m_s \in \mathcal{M}_s} q_s(m_s) = 1 \ \forall s \in \mathcal{S}$$

$$0 \leq I_s \leq I_s^{max} \ \forall s \in \mathcal{S}$$

The above problem is reminiscent of a linear fractional program. Linear fractional programs have linear constraints and a non-linear objective function with an affine numerator and denominator. Efficient offline methods for solving them are known, see, for example, [16] for general optimizations and [17][18] for applications to embedded Markov chains. An online solution is provided in [8][9]. The above problem is more complex due to fractional terms with different denominators appearing in both the objective function and in the constraints. Such problems are generally intractable. However, the above problem has the special structure that all fractional terms associated with the same server s have the same denominator. We use this to construct a novel online solution. Our resulting algorithm, presented in the next section, also works when the sets $\mathcal{M}_s$ are infinite and when the rates λ_n are unknown.

IV. The Dynamic Algorithm

Our algorithm is parameterized by a constant $V > 0$ that affects a performance tradeoff. Define $\mathcal{S}(t_k)$ as the set of servers that begin renewal frames at time t_k. For example, $\mathcal{S}(t_0) = \{1, \ldots, S\}$, since all servers begin frames at time $t_0 = 0$. In general, $\mathcal{S}(t_k)$ can be a strict subset of $\{1, \ldots, S\}$, often consisting of only one server. The following algorithm is implemented in a distributed fashion at each server:

- At each time t_k, each server $s \in \mathcal{S}(t_k)$ observes the queues $\tilde{Q}_1(t_k), \ldots, \tilde{Q}_N(t_k)$ and chooses m_{sk} and I_{sk}, defined in (4), as the solution to the following problem:

$$\text{Minimize:} \quad \frac{V\hat{e}_s(m_{sk}, I_{sk}) - \sum_{n=1}^{N} \tilde{Q}_n(t_k)\hat{\mu}_{sn}(m_{sk})}{\hat{T}_s(m_{sk}, I_{sk})} \tag{12}$$

$$\text{Subject to:} \quad m_{sk} \in \mathcal{M}_s \ , \ 0 \leq I_{sk} \leq I_s^{max} \tag{13}$$

  breaking ties arbitrarily.
- Update queues $\tilde{Q}_n(t_k)$ for each $n \in \{1, \ldots, N\}$ according to (3).

The above algorithm can be viewed as a generalization of the max-weight rule for queue stability from [19]. This generalization allows multiple asynchronous servers and treats joint stability and average power minimization.

A. The I_{sk} and m_{sk} decisions

Here we more precisely specify the decisions m_{sk} and I_{sk} that solve (12)-(13). The definitions of $\hat{T}_s(m_s, I_s)$ and $\hat{e}_s(m_s, I_s)$ in (1)-(2) imply that (12) is equivalent to:

$$\frac{V(\hat{e}_s^{proc}(m_{sk}) + p_s^{idle} I_{sk}) - \sum_{n=1}^{N} \tilde{Q}_n(t_k)\hat{\mu}_{sn}(m_{sk})}{\hat{D}_s(m_{sk}) + I_{sk}}$$

By fixing m_{sk} and taking a derivative of the above with respect to I_{sk}, it is easy to see that the above expression is non-decreasing in I_{sk} whenever:

$$V\hat{e}_s^{proc}(m_{sk}) \leq V\hat{D}_s(m_{sk})p_s^{idle} + \sum_{n=1}^{N} \tilde{Q}_n(t_k)\hat{\mu}_{sn}(m_{sk})$$

and is non-increasing in I_{sk} otherwise. Define $I_{sk}^*(m_{sk})$ as the optimal I_{sk} to choose, given a particular m_{sk}. Then:

$$I_{sk}^*(m_{sk}) = \begin{cases} 0 & \text{if } V\hat{e}_s^{proc}(m_{sk}) \leq V\hat{D}_s(m_{sk})p_s^{idle} \\ & \quad + \sum_{n=1}^{N} \tilde{Q}_n(t_k)\hat{\mu}_{sn}(m_{sk}) \\ I_s^{max} & \text{otherwise} \end{cases}$$

Thus, it suffices to choose $m_{sk} \in \mathcal{M}_s$ to minimize the following expression (breaking ties arbitrarily):

$$\frac{V(\hat{e}_s^{proc}(m_{sk}) + p_s^{idle} I_s^*(m_{sk})) - \sum_{n=1}^N \tilde{Q}_n(t_k)\hat{\mu}_{sn}(m_{sk})}{\hat{D}_s(m_{sk}) + I_s^*(m_{sk})}$$

If $\mathcal{M}_s$ is a finite set of options, as is typically the case, then we simply evaluate the above expression for each option and choose the minimizing one. In more complex cases where $\mathcal{M}_s$ is an infinite set, the infimum of the above expression may not be achievable over $m_{sk} \in \mathcal{M}_s$. Fortunately, the next section incorporates *approximate* minimization, where the m_{sk} and I_{sk} decisions result in an expression that differs by an additive constant from the infimum. Finally, because it is sufficient to come within an additive constant of the infimum, one can use the *actual* queue values $Q_n(t_k)$ in (12), rather than $\tilde{Q}_n(t_k)$.

B. C-Additive Approximations

Define $\tilde{\boldsymbol{Q}}(t_k) \triangleq (\tilde{Q}_1(t_k), \ldots, \tilde{Q}_N(t_k))$ as the queue vector at time t_k. Define $val(\tilde{\boldsymbol{Q}}(t_k))$ as the infimum objective function value for the problem (12)-(13). An algorithm for making decisions at each server over time is called a *C-additive approximation* if for a given constant $C \geq 0$, the following holds for all times t_k and all servers $s \in \mathcal{S}(t_k)$:

$$\frac{V\hat{e}_s(m_{sk}, I_{sk}) - \sum_{n=1}^N \tilde{Q}_n(t_k)\hat{\mu}_{sn}(m_{sk})}{\hat{T}_s(m_{sk}, I_{sk})} \leq val(\tilde{\boldsymbol{Q}}(t_k)) + C$$

where m_{sk}, I_{sk} are defined in (4). A 0-additive approximation is an algorithm that chooses decisions that exactly solve (12)-(13) on every sub-frame.

C. Performance Analysis

Define a *Lyapunov function* $L(t_k)$ by:

$$L(t_k) \triangleq \frac{1}{2}\sum_{n=1}^N \tilde{Q}_n(t_k)^2$$

Define $\Delta(t_k) \triangleq L(t_{k+1}) - L(t_k)$.

Lemma 1: For each t_k, we have:

$$\Delta(t_k) \leq B(t_k) + \sum_{n=1}^N \tilde{Q}_n(t_k)(a_n[t_k, t_{k+1}) - b_n[t_k, t_{k+1})) \quad (14)$$

where $B(t_k)$ is defined:

$$B(t_k) \triangleq \frac{1}{2}\sum_{n=1}^N (a_n[t_k, t_{k+1}) - b_n[t_k, t_{k+1}))^2$$

Proof: This follows immediately by squaring (3) and using the fact that $\max[x, 0]^2 \leq x^2$. □

Lemma 2: Suppose we use a C-additive approximation. Then for each time t_k and each server $s \in \mathcal{S}(t_k)$, that is, each server s that begins a new renewal frame at time t_k, the the decisions m_{sk} and I_{sk} satisfy:

$$\frac{V\hat{e}_s(m_{sk}, I_{sk}) - \sum_{n=1}^N \tilde{Q}_n(t_k)\hat{\mu}_{sn}(m_{sk})}{\hat{T}_s(m_{sk}, I_{sk})} \leq C + \frac{Ve_s - \sum_{n=1}^N \tilde{Q}_n(t_k)\mu_{sn}}{T_s}$$

for all vectors $(e_s, T_s, \mu_{s1}, \ldots, \mu_{sN}) \in \overline{Conv(\mathcal{A}_s)}$.

Proof: Recall that $\mathcal{A}_s$ is the set of all vectors $(e_s, T_s, \mu_{s1}, \ldots, \mu_{sN})$ achievable for server s by decisions $m_s \in \mathcal{S}$ and $I_s \in [0, I_s^{max}]$. Thus, by definition of $val(\tilde{\boldsymbol{Q}}(t_k))$:

$$val(\tilde{\boldsymbol{Q}}(t_k)) = \inf_{\boldsymbol{v}_s \in \mathcal{A}_s} \left[\frac{Ve_s - \sum_{n=1}^N \tilde{Q}_n(t_k)\mu_{sn}}{T_s}\right]$$

By the definition of a C-additive approximation, we have for each time t_k and each server $s \in \mathcal{S}(t_k)$:

$$\frac{V\hat{e}_s(m_{sk}, I_{sk}) - \sum_{n=1}^N \tilde{Q}_n(t_k)\hat{\mu}_{sn}(m_{sk})}{\hat{T}_s(m_{sk}, I_{sk})} \leq C + \inf_{\boldsymbol{v}_s \in \mathcal{A}_s} \left[\frac{Ve_s - \sum_{n=1}^N \tilde{Q}_n(t_k)\mu_{sn}}{T_s}\right]$$

It can be shown that the infimum over vectors in $\mathcal{A}_s$ on the right-hand-side of the above inequality is the same as the infimum over vectors in $Conv(\mathcal{A}_s)$ (see [8]), which in turn is equal to the infimum over the closure of $Conv(\mathcal{A}_s)$ (because the infimum of a continuous function over a set is equal to the infimum of the function over the closure of the set). □

Lemma 3: Suppose we use a C-additive approximation. Then there is a finite constant $G > 0$, independent of the V parameter, such that for any vectors that satisfy $(e_s, T_s, \mu_{s1}, \ldots, \mu_{sN}) \in \overline{Conv(\mathcal{A}_s)}$ for all $s \in \{1, \ldots, S\}$, and for each time t_k, we have:

$$\mathbb{E}\left[\Delta(t_k) + V\sum_{s=1}^S \int_{t_k}^{t_{k+1}} p_s(t)dt\right] \leq \mathbb{E}[\delta_k](G + CS) + V\mathbb{E}[\delta_k]\sum_{s=1}^S \frac{e_s}{T_s} + \sum_{n=1}^N \mathbb{E}[\delta_k Q_n(t_k)][\lambda_n - \sum_{s=1}^S \frac{\mu_{sn}}{T_s}]$$

The constant G is related to the bounds μ_{sn}^{max}, D_s^{min}, D_s^{max}, I_s^{max} and the rates $\lambda_1, \ldots, \lambda_N$. While G can be explicitly computed, we omit this computation for brevity.

Proof: (Lemma 3) First note that the penalty rate function $p_s(t)$ is constant over the sub-frame $[t_k, t_{k+1})$. During this frame the server s uses decisions m_{sk} and I_{sk}, and so:

$$\int_{t_k}^{t_{k+1}} p_s(t)dt = \delta_k \frac{\hat{e}_s(m_{sk}, I_{sk})}{\hat{T}_s(m_{sk}, I_{sk})}$$

where we recall that $\delta_k = t_{k+1} - t_k$. Using this equality with (14) gives the following drift-plus-penalty bound:

$$\begin{aligned}
&\Delta(t_k) + V\sum_{s=1}^S \int_{t_k}^{t_{k+1}} p_s(t)dt \\
&\leq B(t_k) + \delta_k \sum_{s=1}^S \left[\frac{V\hat{e}_s(m_{sk}, I_{sk})}{\hat{T}_s(m_{sk}, I_{sk})}\right] \\
&\quad + \sum_{n=1}^N \tilde{Q}_n(t_k)(a_n[t_k, t_{k+1}) - b_n[t_k, t_{k+1})) \\
&= B(t_k) + \sum_{n=1}^N \tilde{Q}_n(t_k)a_n[t_k, t_{k+1}) \\
&\quad + \delta_k \sum_{s=1}^S \left[\frac{V\hat{e}_s(m_{sk}, I_{sk}) - \sum_{n=1}^N \tilde{Q}_n(t_k)\hat{\mu}_{sn}(m_{sk})}{\hat{T}_s(m_{sk}, I_{sk})}\right] \quad (15)
\end{aligned}$$

where the final equality uses the definition of $b_n[t_k, t_{k+1})$ in (5). Now consider any vectors $(e_s, T_s, \mu_{s1}, \ldots, \mu_{sN}) \in \overline{Conv(\mathcal{A}_s)}$ for $s \in \{1, \ldots, S\}$. Fix a server $s \in \{1, \ldots, S\}$, and define t_{sk} as the time at which s begun the frame that overlaps with time t_k (so that $t_{sk} \leq t_k$). Then by Lemma 2:

$$\frac{V\hat{e}_s(m_{sk}, I_{sk}) - \sum_{n=1}^{N} \tilde{Q}_n(t_{sk})\hat{\mu}_{sn}(m_{sk})}{\hat{T}_s(m_{sk}, I_{sk})} \leq C + \frac{Ve_s - \sum_{n=1}^{N} \tilde{Q}_n(t_{sk})\mu_{sn}}{T_s} \quad (16)$$

Define θ_{sn} as an upper bound on the amount of service to queue n during the interval $[t_{sk}, t_k)$:

$$\theta_{sn} \triangleq (D_s^{max} + I_s^{max}) \sum_{s=1}^{S} \frac{\mu_{sn}^{max}}{D_s^{min}}$$

We thus have for all n, s, k:

$$\tilde{Q}_n(t_{sk}) - \theta_{sn} \leq \tilde{Q}_n(t_k) \leq \tilde{Q}_n(t_{sk}) + a_n[t_{sk}, t_k)$$

Using this enables one to replace the $\tilde{Q}_n(t_{sk})$ values in (16) with $\tilde{Q}_n(t_k)$ values, while appropriately adding a fudge-factor. This can be used in (15). By taking expectations and using the fact that arrivals are Poisson and $\mathbb{E}[a_n[t_k, t_{k+1})] = \delta_k \lambda_n$, one can complete the derivation by computing the bound G (computation omitted for brevity). □

Theorem 3: Suppose $\boldsymbol{\lambda} \in \Lambda$ and we use a C-additive approximation of the dynamic algorithm from Section IV. Then all queues are mean rate stable, and:

(a) For all $k > 0$, average power satisfies:

$$\frac{\sum_{s=1}^{S} \mathbb{E}\left[\int_0^{t_k} p_s(t)dt\right]}{\mathbb{E}[t_k]} \leq p^{opt}(\boldsymbol{\lambda}) + \frac{G + CS}{V}$$

(b) If $\boldsymbol{\lambda}$ is interior to Λ, so that there is an $\epsilon > 0$ such that $(\lambda_1 + \epsilon, \ldots, \lambda_N + \epsilon) \in \Lambda$, then for all $k > 0$:

$$\frac{\sum_{n=1}^{N} \mathbb{E}\left[\int_0^{t_k} \tilde{Q}_n(t)dt\right]}{\mathbb{E}[t_k]} \leq \sum_{n=1}^{N} \lambda_n + \frac{G + CS + G_0 V}{\epsilon}$$

for some non-negative constants G, G_0 that are independent of V and ϵ.

Proof: By Theorem 2, there exist vectors $(e_s^*, T_s^*, \mu_{s1}^*, \ldots, \mu_{sN}^*) \in \overline{Conv(\mathcal{A}_s)}$ such that: $\sum_{s=1}^{S} e_s^*/T_s^* = p^{opt}(\boldsymbol{\lambda})$ and $\sum_{s=1}^{S} \mu_{sn}^*/T_s^* \geq \lambda_n$ for all $n \in \{1, \ldots, N\}$. Using these vectors in Lemma 3 gives:

$$\mathbb{E}\left[\Delta(t_k) + V\sum_{s=1}^{S} \int_{t_k}^{t_{k+1}} p_s(t)dt\right] \leq \mathbb{E}[\delta_k](G + CS) + V\mathbb{E}[\delta_k] p^{opt}(\boldsymbol{\lambda})$$

Summing the above over all $k \in \{0, \ldots, K-1\}$ and using the fact that $\mathbb{E}\left[\sum_{k=0}^{K-1} \Delta(t_k)\right] = \mathbb{E}[L(t_K)] - 0 \geq 0$ yields:

$$V\sum_{s=1}^{S} \mathbb{E}\left[\int_0^{t_K} p_s(t)dt\right] \leq \mathbb{E}[t_K](G + CS) + V\mathbb{E}[t_K] p^{opt}(\boldsymbol{\lambda})$$

from which the power bound of part (a) follows by rearranging terms. That all queues are mean rate stable follows because the drift of the quadratic Lyapunov function is bounded by a constant [9]. We omit the proof of part (b) for brevity. □

Theorem 3 shows that our algorithm can push average power within $O(1/V)$ of optimality, with an $O(V)$ tradeoff in average queue size.

V. CONCLUSIONS

This work develops a dynamic scheduling method for distributed control of multiple asynchronous servers. We showed that the problem involves optimizing ratios of time averages, and results in a challenging non-convex optimization problem with fractional terms (with different denominators) in the objective function and in the constraints. However, we exploited the physical structure of the system and developed an online algorithm that solves the problem without knowledge of the arrival rates. Finally, we note that the technique can be viewed as a computational algorithm for purely deterministic problems with similar structure, and may be useful in other contexts.

REFERENCES

[1] E. Grochowski, R. Ronen, J. Shen, and H. Wang. Best of both latency and throughput. *Proc. IEEE Conf. on Computer Design (ICCD)*, pp. 236-243, October 2004.

[2] M. Annavaram, E. Grochowski, and J. Shen. Mitigating amdahl's law through epi throttling. *Proc. 32nd International Symposium on Computer Architecture (ISCA)*, pp. 298-309, June 2005.

[3] M. Lin, A. Wierman, L. L. H. Andrew, and E. Thereska. Dynamic right-sizing for power-proportional data centers. *Proc. IEEE INFOCOM*, 2011.

[4] A. Wierman, L. L. H. Andrew, and A. Tang. Power-aware speed scaling in processor sharing systems. *Proc. IEEE INFOCOM*, Rio de Janiero, Brazil, April 2009.

[5] C.-P. Li and M. J. Neely. Delay and rate-optimal control in a multi-class priority queue with adjustable service rates. *Proc. IEEE INFOCOM*, 2012.

[6] R. Urgaonkar, B. Urgaonkar, M. J. Neely, and A. Sivasubramaniam. Optimal power cost management using stored energy in data centers. *Proc. SIGMETRICS*, June 2011.

[7] Y. Yao, L. Huang, A. Sharma, L. Golubchik, and M. J. Neely. Data centers power reduction: A two time scale approach for delay tolerant workloads. *Proc. IEEE INFOCOM*, 2012.

[8] M. J. Neely. Low power dynamic scheduling for computing systems. *arXiv technical report, arXiv:1112.2797*, Dec. 2011.

[9] M. J. Neely. *Stochastic Network Optimization with Application to Communication and Queueing Systems*. Morgan & Claypool, 2010.

[10] J. N. Tsitsiklis, D. P. Bertsekas, and M. Athens. Distributed asynchronous deterministic and stochastic gradient optimization algorithms. *IEEE Transactions on Automatic Control*, vol. AC-31, no. 9, September 1986.

[11] L. Bui, E. Eryilmaz, R. Srikant, and X. Wu. Joint asynchronous congestion control and distributed scheduling for multi-hop wireless networks. *Proc. IEEE INFOCOM*, 2006.

[12] C. Li and M. J. Neely. Network utility maximization over partially observable markovian channels. *Proc. Intl. Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt)*, May 2011.

[13] C. Li and M. J. Neely. Network utility maximization over partially observable markovian channels. *Arxiv Technical Report: arXiv:1008.3421v1*, Aug. 2010.

[14] J. N. Tsitsiklis. A short proof of the gittens index theorem. *Annals of Applied Probability*, vol. 4, no. 1, pp. 194-199, 1994.

[15] R. Gallager. *Discrete Stochastic Processes*. Kluwer Academic Publishers, Boston, 1996.

[16] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[17] B. Fox. Markov renewal programming by linear fractional programming. *Siam J. Appl. Math*, vol. 14, no. 6, Nov. 1966.

[18] H. Mine and S. Osaki. *Markovian Decision Processes*. American Elsevier, New York, 1970.

[19] L. Tassiulas and A. Ephremides. Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks. *IEEE Transacations on Automatic Control*, vol. 37, no. 12, pp. 1936-1948, Dec. 1992.