Dynamic overload balancing in server farms

Dynamic Overload Balancing in Server Farms

Chih-ping Li*, Georgios S. Paschos*‡, Leandros Tassiulas§ and Eytan Modiano*

*LIDS, Massachusetts Institute of Technology, Cambridge, MA, USA

§Dept. of ECE, University of Thessaly, Volos, Greece

‡Informatics & Telematics Institute, CERTH, Greece

***Abstract*—We consider the problem of optimal load balancing in a server farm under overload conditions. A convex penalty minimization problem is studied to optimize queue overflow rates at the servers. We introduce a new class of α-*fair penalty functions*, and show that the cases of $\alpha = 0, 1, \infty$ correspond to minimum sum penalty, penalty proportional fairness, and min-max fairness, respectively. These functions are useful to maximize the time to first buffer overflow and minimize the recovery time from temporary overload. In addition, we show that any policy that solves an overload minimization problem with strictly increasing penalty functions must be throughput optimal. A dynamic control policy is developed to solve the overload minimization problem in a stochastic setting. This policy generalizes the well-known join-the-shortest-queue (JSQ) policy and uses intelligent job tagging to optimize queue overflow rates without the knowledge of traffic arrival rates.**

I. Introduction

Server farms suffer from overload conditions, i.e., when the user demand exceeds the service capacity, on a regular basis. The cause of server overload includes demand fluctuations, flash crowds, denial-of-service attacks, server and link failures, or turning servers off for power saving and maintenance. Server farms are expected to have high utilization in order to be cost-effective, leaving less room to accommodate unexpected events that may lead to overload. An overloaded system may incur reduced throughput and longer response time, resulting in low user satisfaction and loss of revenue for the service providers. Therefore, graceful and robust management of overload surges is an important task.

We consider the problem of optimal load balancing in a server farm consisting of load balancers and servers. Each load balancer routes incoming jobs to its connected servers, which may have different service rates. We focus on the overload scenario where the traffic demand is unknown and cannot be supported by the server farm. The performance metrics of interest are queue overflow rates at the servers and total system throughput. In general, a desired operating point in an overloaded system is one that maximizes the total throughput while enforcing the queue overflow rates to possess certain good properties, e.g., maximizing the time to first buffer overflow or minimizing the recovery time from temporary overload. Achieving these performance objectives introduces a nontrivial network control problem, because the desired system performance cannot be computed offline due to the unknown traffic demand, and existing throughput-optimal policies in a stable system (e.g., the max-weight policies) may be sub-optimal when the system becomes unstable [1]. Also, it is not understood under what conditions (if possible) can maximizing throughput and inducing desired queue overflow rates be achieved simultaneously in an overloaded system.

This paper studies optimal overload balancing in a server farm as convex penalty minimization problems. By properly choosing the penalty functions, we can achieve the desired system performance as solutions to the minimization problems. Our main contributions include:

- We characterize the set of all feasible queue overflow vectors and show that the set is convex. This is somewhat surprising because we show that the feasible region of queue overflow rates under the collection of work-conserving policies can be non-convex, which complicates the design of optimal control policies. Also, we identify useful queue overflow vectors in the feasible region, including: (i) the most-balanced vector which is throughput-optimal, min-max fair, and useful to maximize the time to first buffer overflow; (ii) the weighted min-max fair vector which minimizes the recovery time from temporary overload.
- We introduce a new class of α-fair convex penalty functions as a generalization of the well-known α-fair utility functions [2], and show that the cases of $\alpha = 0, 1, \infty$ correspond to minimum sum penalty, penalty proportional fairness, and min-max fairness, respectively. Furthermore, we prove that any policy solving a convex overflow minimization problem with strictly increasing penalty functions must be throughput optimal.
- In a stochastic setting, we develop a dynamic load balancing policy that solves queue overflow minimization problems and is throughput optimal. This policy generalizes the join-the-shortest-queue policy and proactively adjusts queue overflow rates by an intelligent job tagging mechanism. This method turns the overload control problem in an unstable queueing system into one that aims at stabilizing virtual queues.

The optimal overload balancing problem in this paper is fundamentally different from the traditional studies of network load balancing [3, Chap. 5], where the former focuses on the overload case and the latter assumes that the traffic demand is always supportable in the network. The study of network systems in overload has received significant attention recently. Fluid limits [4], [5] and throughput optimization problems [6], [7] are investigated. Work [8] shows that the most-balanced queue growth rate vector exists in a single-commodity network in overload; this work uses deterministic fluid models for

This work is supported by NSF grant CNS-0915988, ONR grant N00014-12-1-0064, ARO MURI grant W911NF-08-1-0238, and DTRA grants HDTRA1-07-1-0004 and HDTRA-09-1-005.

performance analysis and does not provide a stochastic control policy to achieve the most-balanced vector. The queue growth rates of max-weight and α-fair policies in overloaded networks are analyzed in [1], [9]. Tuning the max-weight policy to render a queue growth rate vector in a given direction is studied in [10]. Job tagging mechanisms are used to maximize rewards in overloaded many-server systems [11]. Adaptive overload control for web servers is considered in [12], [13].

The outline of the paper is as follows. Section II describes the system model, characterizes the feasible queue overflow region, and discusses useful queue overflow vectors. In Section III, we introduce the overflow minimization problem and the α-fair penalty functions, and prove that throughput optimality is achieved using optimal queue overflow vectors. The dynamic overload balancing policy is proposed and analyzed in Section IV, followed by simulation results in Section V.

II. System Model

We consider a server farm that comprises a set B of load balancers and a set S of servers. According to the system architecture, a load balancer $b \in B$ is allowed to route incoming jobs to a subset S_b of servers, $S_b \subseteq S$. Let $B_s \subseteq B$ be the set of load balancers that can direct traffic to a server $s \in S$. Consider a time-slotted system. In slot t, load balancer b receives $A_b(t)$ jobs and forwards them to the connected servers in S_b. We assume that $A_b(t)$ are i.i.d. over slots with mean λ_b, and have bounded support $A_b(t) \leq A_{\max}$. Each job brings some workload to the system, referred to as the job size. The size of incoming jobs is i.i.d. and denoted by a random variable X. Suppose X has bounded support with $X \leq x_{\max}$, and its probability distribution is otherwise arbitrary. Assume that the size of the jobs is unknown to the system, but its mean $\mathbb{E}[X]$ is available. Server s has a constant processing rate c_s, and takes x/c_s slots to complete a job of size x. Jobs are processed in a nonpreemptive fashion at each server. Load balancers forward incoming jobs to the servers for processing. Specifically, load balancer b routes $A_{bs}(t)$ jobs to server s in slot t, where $\sum_{s\in S} A_{bs}(t) = A_b(t)$. If load balancer b is not connected to server s, i.e., $b \notin B_s$, then $A_{bs}(t) = 0$ for all t. Each server stores received jobs that are not yet processed in a queue. See Fig. 1(a) for an example of the server farm.

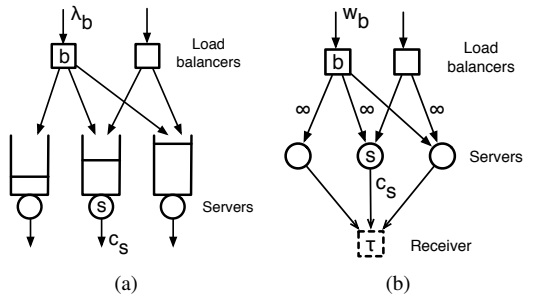


Fig. 1. (a) A server farm with two load balancers and three servers. (b) A fluid network representation of the server farm.

A. Achievable performance

We seek to develop stochastic control policies that optimize long-term queue overflow rates at the servers. For that, it is important to understand the set of achievable queue overflow vectors using a simple fluid model. It is useful to regard the server farm as a single-commodity network with an augmented receiver τ, see Fig. 1(b). The receiver τ is connected to the servers s with directed links (s, τ) of capacity c_s. Each load balancer b is connected to its connected servers with directed links (b, s), where $s \in S_b$. Since there is no queueing effect at the load balancers, let the links between load balancers and servers have infinite capacity. The exogenous traffic rate of load balancer b is $w_b \triangleq \lambda_b \mathbb{E}[X]$ in the fluid model, where w_b is also the *workload* arrival rate in the stochastic model.

We denote by q_s the queue overflow rate at server s, i.e., the workload growth rate at server s.[1] Let $\mathcal{Q}$ be the set of feasible overflow vectors $\boldsymbol{q} = (q_s, s \in S)$. We denote by w_{bs} and $w_{s\tau}$ the data flows over links (b, s) and (s, τ), respectively. The flow w_{bs} is the traffic rate at which load balancer b directs its traffic to server s, and $w_{s\tau}$ is the throughput at server s. A queue overflow vector $\boldsymbol{q}$ is feasible if and only if there exist flow variables w_{bs} and $w_{s\tau}$ satisfying the following constraints:

$$w_b = \sum_{s\in S_b} w_{bs}, \ b \in B \tag{1}$$

$$\sum_{b\in B_s} w_{bs} = q_s + w_{s\tau}, \ s \in S \tag{2}$$

$$w_{s\tau} \leq c_s, \ s \in S \tag{3}$$

$$w_{bs} \geq 0, w_{s\tau} \geq 0. \tag{4}$$

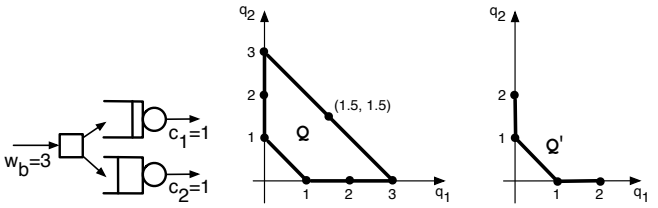
Equations (1) and (2) are flow conservation constraints, where (2) shows that the total flow rate into server s is equal to the throughput plus the queue overflow rate at that server. Equation (3) is the link capacity constraint. We define the vector $\boldsymbol{w} = (w_{bs}, w_{s\tau}, q_s)_{b\in B, s\in S}$ that satisfies (1)-(4) as a *superflow*, which reduces to the standard "flow" definition if the queue overflow rates q_s are zero for all s. As a result, the feasible set $\mathcal{Q}$ is

$$\mathcal{Q} = \{ \boldsymbol{q} \mid \boldsymbol{w} \text{ is a superflow} \}, \tag{5}$$

which is a convex and compact set. See Figures 2(a) and 2(b) for an example.

The feasible set $\mathcal{Q}$ in (5) is fully achievable if the servers are allowed to be non-work-conserving. As an example, consider a given load balancing allocation (w_{bs}), which are flows between load balancers and servers. We observe from (2) that $q_s = \sum_{b\in B_s} w_{bs}$ is a feasible queue overflow rate that is achieved by always idling server s (i.e., setting $w_{s\tau} = 0$). The reason for allowing suboptimal non-work-conserving servers is that the resulting feasible queue overflow region $\mathcal{Q}$ is convex. On the contrary, Fig. 2(c) gives an example of the set of feasible queue overflow vectors under work-conserving policies, and the region is not convex. Comparing Fig. 2(b) and Fig. 2(c), we observe that allowing non-work-conserving servers "convexifies" the feasible queue overflow region by including suboptimal points. This enables us to formulate the overload balancing problem as a convex optimization problem without affecting the optimal solution.

[1] We use the terms "queue overflow" and "queue growth" interchangeably.



(a) (b) (c)

Fig. 2. (a) A multi-server example. (b) The set $\mathcal{Q}$ of queue overflow vectors under both work-conserving and non-work-conserving server operations. The overflow vector $(3, 0)$ is achieved by sending all traffic to server 1 but keeping both servers idle. The middle point $(1.5, 1.5)$ is achieved by equal time-sharing between sending all traffic to one of the two servers. (c) The non-convex set $\mathcal{Q}'$ of feasible queue overflow vectors when both servers are work-conserving. The overflow vector $(2, 0)$ is achieved by sending all traffic to server 1. Here, equal time-sharing over the two policies that achieve queue overflow rates $(2, 0)$ and $(0, 2)$ respectively sends an average traffic rate of 1.5 to each server, resulting in the queue overflow rates $(0.5, 0.5)$, which is not a convex combination of $(2, 0)$ and $(0, 2)$. Therefore, the time-sharing argument cannot be used in $\mathcal{Q}'$.

B. Useful performance objectives

We describe three useful performance objectives and identify the queue overflow vectors that achieve them. Consider maximizing the system throughput. Summing (1) and (2) over b and s, we get

$$\sum_{b\in B} w_b = \sum_{s\in S} q_s + \sum_{s\in S} w_{s\tau}, \tag{6}$$

where the first sum is the total exogenous traffic rate and the last sum is the system throughput. This shows that maximizing total throughput is the same as minimizing the sum of queue overflows.

Next, consider the queue overflow vector that solves

$$\min_{\boldsymbol{q}\in\mathcal{Q}} \max_{s\in S} q_s. \tag{7}$$

Its solution is useful for preventing queue buffer overflow [8]. To see this, suppose each server has a finite buffer of equal size A, and all buffers are initially empty. At the fluid level, $\min_{s\in S}\{A/q_s\}$ is the earliest time epoch at which buffer overflow occurs, and is maximized by the solution to (7). A solution to (7) is the min-max fair queue overflow vector $\boldsymbol{q}^{(\infty)}$, which exists in the convex and compact set $\mathcal{Q}$ [14, Theorem 1].[2] The vector $\boldsymbol{q}^{(\infty)}$ solves (7) because $\boldsymbol{q}^{(\infty)}$ is the leximax minimum in $\mathcal{Q}$, i.e., it is the lexicographically smallest vector over feasible queue overflow vectors sorted in a decreasing order. Both leximax minimum and min-max fairness are stronger than the criterion (7).

Motivated by the simulation example in the introduction, we are interested in the queue overflow vector that solves

$$\min_{\boldsymbol{q}\in\mathcal{Q}} \max_{s\in S} \{q_s/c_s\}, \tag{8}$$

where c_s is the capacity of server s. Its solution is useful for mitigating temporary system overload in minimum time. To see this, suppose that the system is initially empty and suffers from temporary overload for a duration of t seconds, where server s has queue overflow rate q_s. At the end of the overload period, the amount of workload accumulated at server s is tq_s. Thus, $\max_{s\in S} tq_s/c_s$ is the remaining time the system is affected by temporary overload after the overload event ends, and is minimized by the solution to (8). A solution to (8) is the weighted min-max fair queue overflow vector with weights c_s.

In the next section, we show that the queue overflow vectors discussed above are achieved as solutions to convex penalty minimization problems.

III. Overload Balancing and Optimization

We formulate the overload balancing problem as a convex optimization problem. Each server $s \in S = \{1, \ldots, S\}$ has a penalty function h_s of the queue overflow rate q_s, where h_s is increasing, continuous, and convex. We consider the optimization problem

$$\text{minimize} \quad \sum_{s=1}^{S} h_s(q_s) \tag{9}$$

$$\text{subject to} \quad \boldsymbol{q} \in \mathcal{Q}, \tag{10}$$

where $\mathcal{Q}$ is given in (5). The optimal solution exists because (9)-(10) minimizes a continuous function over a compact set; the solution is unique if the objective function is strictly convex.

We wish to design penalty functions that can achieve desired queue overflow vectors, e.g., those discussed in Section II-B, by solving (9)-(10). An important result in [8] is that there exists a most-balanced queue overflow vector $\boldsymbol{q}^{\text{MB}}$ in a single-commodity network under overload, of which the server farm in this paper is a special case. The vector $\boldsymbol{q}^{\text{MB}}$ has two important properties [8]: (i) it simultaneously maximizes system throughput and achieves min-max fair queue overflow rates; (ii) if $h_s = h$ for all servers s where h is any convex and increasing function, then $\boldsymbol{q}^{\text{MB}}$ solves the corresponding optimization problem (9)-(10). Consequently, if we choose $h_s(x) = x^2$ or any convex increasing function for all s, then the unique solution to (9)-(10) achieves both maximum system throughput and min-max fair queue overflow rates. In general, however, the vector $\boldsymbol{q}^{\text{MB}}$ does not yield weighted min-max fairness, for which we introduce an interesting class of penalty functions next.

A. α-fair penalty function

We propose a family of convex penalty functions that can be used to achieve several fairness objectives. We define the class of α-fair *penalty* functions

$$h^{(\alpha)}(q) = \frac{q^{1+\alpha}}{1+\alpha}, \quad \alpha \geq 0,\ q \geq 0, \tag{11}$$

which can be regarded as a natural generalization of the α-fair utility functions [2]

$$g^{(\alpha)}(r) = \begin{cases} r^{1-\alpha}/(1-\alpha), & \alpha \in \mathbb{R}^+ \setminus \{1\} \\ \log(r), & \alpha = 1 \end{cases}$$

by letting α take negative values in $g^{(\alpha)}(r)$. In general, the convex and increasing functions $h^{(\alpha)}(q)$ are useful in penalty

[2]The notation $\boldsymbol{q}^{(\infty)}$ is explained in Section III-A.

minimization problems, whereas the concave functions $g^{(\alpha)}(r)$ are useful for reward maximization problems.

Consider the problem (9)-(10) with α-fair penalty functions:

$$\text{minimize} \quad \sum_{s=1}^{S} \frac{(q_s)^{1+\alpha}}{1+\alpha} \tag{12}$$

$$\text{subject to} \quad \boldsymbol{q} \in \mathcal{Q}, \tag{13}$$

where $\mathcal{Q}$ is compact and convex. Let its optimal solution be $\boldsymbol{q}^{(\alpha)}$. There are three values of α of particular interest: $\alpha \in \{0, 1, \infty\}$. When $\alpha = 0$, the problem (12)-(13) minimizes the total queue overflow. When $\alpha = 1$, the α-fair penalty function is $q^2/2$ and the first-order optimality condition for the solution $\boldsymbol{q}^{(1)}$ is [15],

$$\sum_{s=1}^{S} (q_s - q_s^{(1)})\, q_s^{(1)} \geq 0, \ \forall \boldsymbol{q} \in \mathcal{Q}. \tag{14}$$

Prior work [16] shows that the condition (14) incurs the same proportional tradeoff as rate proportional fairness [17] for an optimal reward vector (y_n^*)

$$\sum_{n=1}^{N} \frac{y_n - y_n^*}{y_n^*} \leq 0, \quad \text{for all feasible } (y_n). \tag{15}$$

The difference is that (14) is meaningful in the context of penalty minimization and (15) in reward maximization. We refer to $\boldsymbol{q}^{(1)}$ as the vector that achieves *penalty proportional fairness*. The criterion (14) has the product form instead of the ratio form (15) because we favor large reward in (15) but desire small penalty in (14). We note that rate proportional fairness uses logarithmic utility functions while penalty proportional fairness uses quadratic penalty functions. Finally, the case of $\alpha \to \infty$ corresponds to min-max fairness, as shown in the next lemma.

Lemma 1. The vector $\boldsymbol{q}^{(\infty)} = \lim_{\alpha\to\infty} \boldsymbol{q}^{(\alpha)}$, where $\boldsymbol{q}^{(\alpha)}$ is the solution to (12)-(13), exists and achieves min-max fairness in the compact and convex set $\mathcal{Q}$. (The proof is omitted due to the space constraint.)

An obvious generalization of the penalty functions (11) is to consider *weighted* α-fair penalty functions $(q/c)^{1+\alpha}/(1+\alpha)$ with a weight $c > 0$. This is useful to minimize the weighted sum of queue overflow rates by choosing $\alpha = 0$, and this is directly related to maximizing the weighted sum throughput in the server farm. Also, the weighted min-max fair queue overflow vector with weights c_s in $\mathcal{Q}$ is min-max fair in the set of feasible weighted vectors $(q_1/c_1, \ldots, q_S/c_S)$. Therefore, it is the solution to (9)-(10) with the objective function

$$\sum_{s=1}^{S} \frac{(q_s/c_s)^{1+\alpha}}{1+\alpha}$$

as $\alpha \to \infty$. In general, however, the set of feasible weighted queue overflow vectors $(q_1/c_1, \ldots, q_S/c_S)$ in the server farm does not contain a most-balanced vector. Therefore, weighted min-max fairness and minimum weighted sum penalty do not coincide.

B. Throughput optimality

Throughput is a performance metric as important as queue overflow, and it is desired to understand under what conditions does the optimal queue overflow vector that solves (9)-(10) induces maximum system throughput. One known condition is when the penalty functions h_s in (9)-(10) are the same for all s, in which case the solution is the most-balanced overflow vector $\boldsymbol{q}^{\text{MB}}$. More generally, the next theorem shows that maximum total throughput is always achieved whenever the functions h_s are convex and strictly increasing.

Theorem 1. If the convex penalty functions $\{h_s, s \in S\}$ are strictly increasing, then any superflow $\boldsymbol{w}$ with the optimal queue overflow vector solving (9)-(10) achieves maximum system throughput.

Proof: See Appendix A. ■

IV. Dynamic Control Algorithm

A well-known policy for balancing loads in server farms is the join-the-shortest-queue (JSQ) policy [18], according to which a load balancer routes jobs to the server with the shortest queue. When the server farm is underloaded, it is not difficult to see that the JSQ policy is a max-weight policy and thus throughput optimal [19], i.e., JSQ stabilizes the system whenever the exogenous traffic rates are within the stability region. When the system is in overload, however, we do not expect the JSQ policy to always yield optimal queue overflow rates—the long-term performance of JSQ, if it converges, corresponds to one point in the overflow feasible region $\mathcal{Q}$, but the overload balancing problem (9)-(10) with different objective functions can have different solutions.

We propose a simple generalization to the JSQ policy that yields optimal queue overflow rates to solve (9)-(10) with different penalty functions. The idea is to virtually categorize the jobs into two groups, tagged and untagged, so that the job tagging rate at a server corresponds to its queue overflow rate and the rate at which jobs are untagged is the throughput at the server. Then, virtual queues are used to optimize the job tagging rates according to the objective functions. In the following, we provide intuition behind the design of our policy.

1) Job Tagging: For each job i, we define the indicator function $I_i = 1$ if the job is tagged, and 0 otherwise; the optimal job tagging mechanism is given in Section IV-3. Let $N_s(t)$ be the number of untagged jobs stored at server s and $n_s(t)$ the number of untagged jobs departing the server in slot t. Recall that $A_{bs}(t)$ is the number of jobs routed from load balancer b to server s in slot t. The queue process $\{N_s(t)\}$ at server s is updated over slots according to:

$$N_s(t+1) = N_s(t) - n_s(t) + \sum_{b\in B} \sum_{i=1}^{A_{bs}(t)} (1 - I_i). \tag{16}$$

For the purpose of explanation, we assume that the system is heavily overloaded so that the servers can only process untagged jobs, and tagged jobs stay in the system forever. Suppose untagged jobs have strict priorities over tagged jobs at each server. Consider a policy that keeps the number of untagged jobs bounded in the system by stabilizing the queues

$N_s(t)$. Under this policy, the average throughput at a server is the departure rate of untagged jobs multiplied by the average job size. In addition, the queue overflow rate at a server is the product of average job tagging rate and average job size, i.e.,

$$q_s = \theta_s \, \mathbb{E}[X],$$

where θ_s is the long-term job tagging rate at server s:

$$\theta_s = \lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \sum_{b\in B} \mathbb{E}\Big[\sum_{i=1}^{A_{bs}(t)} I_i \Big]. \tag{17}$$

Therefore, it is possible to solve the overload balancing problem (9)-(10) by designing a control policy that stabilizes the queues $N_s(t)$, and a job tagging policy that achieves the optimal job tagging rates $\theta_s^* = q_s^*/\mathbb{E}[X]$, where $(q_1^*, \ldots, q_S^*)$ is the optimal solution to (9)-(10).

Due to the randomness of job arrivals, some tagged jobs may receive service, in which case the queue overflow rate at a server is upper bounded by the weighted job tagging rate, or $q_s \le \theta_s \, \mathbb{E}[X]$. The virtual queue mechanism described next ensures that the queue overflow rates q_s can still be optimized by minimizing the average job tagging rates θ_s, $s \in S$.

2) Virtual-Queue Mechanism: Optimizing queue overflow rates amounts to minimizing the penalty function $\sum_{s=1}^S h_s(q_s)$ in (9). From Lyapunov drift theory, one useful method is to turn this penalty minimization into an optimal control problem over virtual queues. Specifically, we observe that the problem (9)-(10) is equivalent to the alternative problem

$$\text{minimize} \quad \sum_{s=1}^{S} h_s(y_s \mathbb{E}[X]) \tag{18}$$

$$\text{subject to} \quad q_s \le y_s \mathbb{E}[X],\ y_s \ge 0,\ s \in S \tag{19}$$

$$\boldsymbol{q} \in \mathcal{Q}, \tag{20}$$

where y_s are auxiliary control variables and $\mathbb{E}[X]$ is the average job size. As a result, minimizing $\sum_{s=1}^S h_s(q_s)$ in (9) is equivalent to achieving the following two objectives: (i) minimizing the alternative penalty function $\sum_{s=1}^S h_s(y_s \mathbb{E}[X])$ over control variables y_s; (ii) satisfying the constraints (19).

To design a control policy that satisfies the constraints (19), we set up a virtual queue $Y_s(t)$ at server s with the queueing dynamics:

$$Y_s(t+1) = \max\big[Y_s(t) - y_s(t), 0\big] + \sum_{b\in B} \sum_{i=1}^{A_{bs}(t)} I_i, \tag{21}$$

where $y_s(t) \in [0, y_{\max}]$ is a decision variable at time t and $y_{\max}$ is a finite but sufficiently large constant. In (21), the amount of arrivals at the virtual queue $Y_s(t)$ in a slot is the number of tagged jobs in that slot. Thus, the arrival rate of queue $Y_s(t)$ is the job tagging rate θ_s. If we regard y_s in (19) as the limiting time average of the decision variable $y_s(t)$, i.e.,

$$y_s = \lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}[y_s(t)], \tag{22}$$

then the service rate of queue $Y_s(t)$ is y_s. If the virtual queue $Y_s(t)$ is stable, then from queueing theory its arrival rate must be less than or equal to the service rate, i.e.,

$$\theta_s \le y_s \quad \Rightarrow \quad q_s \le \theta_s \mathbb{E}[X] \le y_s \mathbb{E}[X]. \tag{23}$$

Thus, stabilizing the queues $Y_s(t)$ satisfies the constraints (19).

Next, to minimize the alternative penalty function (18), by using Jensen's inequality and convexity of h_s we have

$$\lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}\Big[h_s\big(y_s(t)\mathbb{E}[X]\big)\Big] \ge h_s(y_s \mathbb{E}[X]). \tag{24}$$

This inequality suggests that minimizing $\sum_{s=1}^S h_s(y_s \mathbb{E}[X])$ can be achieved by minimizing $\sum_{s=1}^S \mathbb{E}\Big[h_s\big(y_s(t)\mathbb{E}[X]\big)\Big]$ in every slot. In summary, to optimize the queue overflow penalty $\sum_{s=1}^S h_s(q_s)$, it suffices to stabilize the virtual queues $Y_s(t)$ and minimize the sum $\sum_{s=1}^S \mathbb{E}\Big[h_s\big(y_s(t)\mathbb{E}[X]\big)\Big]$ in every slot.

3) Server Selection: From the above discussions, we need to stabilize both queues $N_s(t)$ and $Y_s(t)$ at the servers $s \in S$. It suffices for each load balancer b to run the normal JSQ policy (i.e., the max-weight policy) over the queues $N_s(t)$ and $Y_s(t)$ at its connected servers $s \in S_b$. This policy decides where to forward incoming jobs at each load balancer. After a server receives a job, it needs to decide whether to tag the job or not. We observe that the "arrivals" to the queue $N_s(t)$ are untagged jobs and the "arrivals" to $Y_s(t)$ are tagged jobs. To impose the effect of "joining" the shortest queue at server s, the job is tagged if $N_s(t) > Y_s(t)$ and untagged otherwise.

A. The control policy

The complete overload balancing policy is given as follows.

The JSQ-with-Tagging (JSQT) policy:

1) *Server selection:* Load balancer $b \in B$ routes all incoming jobs in slot t to the server

$$s(t) \in \operatorname{argmin}_{s\in S_b} \min\{N_s(t), Y_s(t)\},$$

where ties are broken arbitrarily.

2) *Job tagging:* Server $s \in S$ tags all received jobs in slot t if $N_s(t) > Y_s(t)$, and tags none of the jobs otherwise. Untagged jobs have strict priority over tagged ones.

3) *Virtual-queue mechanism:* Update $N_s(t)$ and $Y_s(t)$ at the end of slot t according to (16) and (21), respectively, where $y_s(t)$ is the solution to

$$\text{minimize} \quad \frac{V}{\mathbb{E}[X]^2} h_s\big(y_s(t)\,\mathbb{E}[X]\big) - Y_s(t)\, y_s(t), \tag{25}$$

$$\text{subject to} \quad y_s(t) \in [0, y_{\max}], \tag{26}$$

where $V > 0$ is a control parameter and $y_{\max}$ is a finite but sufficiently large constant.

We make the following remarks.

(a) The queues $N_s(t)$ keep track of the number of untagged jobs at server $s \in S$. The virtual queues $Y_s(t)$ are implemented as a means to optimize average job tagging rates, and are *not* the amount of tagged jobs waiting at the servers. We may view $Y_s(t)$ as a dynamic threshold beyond which the amount of untagged jobs $N_s(t)$ at server s triggers an overload signal, making the server start tagging jobs.

(b) The objective function in (25) captures an inherent tradeoff that appears in solving the problem (18)-(20) by stabilizing the virtual queues $Y_s(t)$. Since h_s are increasing, minimizing $\sum_{s=1}^{S} h_s\big(y_s(t)\mathbb{E}[X]\big)$ requires small values of $y_s(t)$. But stabilizing the virtual queues $Y_s(t)$ requires large service rates $y_s(t)$ (see (21)). To optimize this tradeoff, it is natural to minimize the weighted difference of $h_s(y_s(t)\mathbb{E}[X])$ and $Y_s(t)y_s(t)$ in (25), where the weight V captures the relative importance of the two conflicting goals. The product $Y_s(t)y_s(t)$ in (25) reflects the fact that if the virtual backlog $Y_s(t)$ is large, then a large service rate $y_s(t)$ is desired for queue stability.

(c) The value of $y_{\max}$ in (25)-(26) needs to be large enough so that choosing $y_s(t) = y_{\max}$ for all t is sufficient to stabilize the virtual queue $Y_s(t)$. A feasible choice is $y_{\max} = B_{\max}A_{\max}$, where $B_{\max} = \max_{s\in S}|B_s|$ is the maximum number of load balancers connected to a server. Note that $B_{\max}A_{\max}$ is a universal upper bound on the amount of per-slot arrivals to the virtual queue $Y_s(t)$ (see (21)).

(d) The JSQT policy makes optimal decisions by dividing jobs into two groups, tagged and untagged. Since job sizes are assumed to be unknown but have i.i.d. distribution, the identity of jobs in each group is irrelevant. Therefore, jobs can be processed in the order of their arrival without affecting the performance of the JSQT policy, by simply re-marking the $N_s(t)$ head-of-line jobs at each server s as untagged jobs and the rest at the server as tagged in slot t. In other words, we can keep track of the values of $N_s(t)$ as counters and serve all jobs in a first-in-first-out fashion.

B. Examples

We consider two practical cases and provide the solution to the problem (25)-(26) in closed form. First, consider the most-balanced queue overflow vector $\boldsymbol{q}^{\text{MB}}$, which is also the min-max fair vector. Using $h_s(q_s) = (q_s)^2/2$ for all servers s, the virtual-queue mechanism of the JSQT policy chooses $y_s(t) = \min\{y_{\max}, Y_s(t)/V\}$ as the solution to (25)-(26) in every slot. This solution can be used to maximize the time to first buffer overflow in a system with finite buffers. Second, consider the weighted α-fair queue overflow vector with weights c_s. Using the penalty function $h_s(q_s) = (q_s/c_s)^{1+\alpha}/(1+\alpha)$ for server s, the JSQT policy chooses

$$y_s(t) = \min\left\{y_{\max},\ \frac{c_s}{\mathbb{E}[X]}\sqrt[\alpha]{\frac{Y_s(t)\mathbb{E}[X]c_s}{V}}\right\}.$$

By choosing α to be a large integer, the JSQT policy with this choice of $y_s(t)$ yields the weighted min-max fair queue overflow vector, which is useful to minimize the time duration in which the system is affected by temporary overload.

C. Performance analysis

The next theorem proves the optimality of the JSQT policy.

Theorem 2. Let q_s be the long-term queue growth rate of server s under the JSQT policy. We have

$$\sum_{s\in S} h_s(q_s) \le h^* + \frac{F}{V}, \tag{27}$$

where h^* is the optimal objective value of the overload balancing problem (9)-(10), F is a finite constant, and $V > 0$ is a control parameter that can be chosen sufficiently large to diminish the performance gap in (27). Furthermore, the JSQT policy is nearly throughput optimal (from Theorem 1).

Proof of Theorem 2: Omitted due to the space constraint. ■

V. Simulation Results

We simulate the JSQT policy for different scenarios of interest. In all simulations, the arrivals are Poisson processes and the job sizes are exponential random variables with unit mean.

A. Performance of JSQT

We examine the performance of the JSQT policy in a simple load balancing system with two heterogenous servers of capacity $(c_1, c_2) = (1, 2)$ and a load balancer with incoming job arrival rate $\lambda = 4$; see Fig. 3. Since the jobs are assumed to have unit mean, the workload arrival rate is $w = 4$ as well. The topology and the feasible set $\mathcal{Q}$ are shown in Fig. 3. We choose the value of $V = 100$, and use the weighted α-fair penalty functions

$$h_s(q_s) = \frac{(q_s/\xi_s)^{1+\alpha}}{1+\alpha}, \quad s = 1, 2,$$

where ξ_s are positive weights. We examine the following three cases under the JSQT policy; each simulation is executed for one million slots.

1) Symmetric objective: We use the JSQT policy to achieve the most-balanced vector, which is $(0.5, 0.5)$ in this case. We choose weights $\xi_1 = \xi_2 = 1$ and $\alpha = 1$. Hence, the α-fair penalty functions become $h_s(q_s) = (q_s)^2/2$. The queue growth rate vector under the JSQT policy in the simulation is $(.5008, .5007)$, which is point A in Fig. 3. We also simulate the JSQT policy for other values of $\alpha \in \{0.1, 10, 100\}$. We observe that the same queue growth rates are achieved and the workload difference between the four cases is negligible.

2) Asymmetric objective: We seek to achieve weighted min-max fairness with weights proportional to server capacity; this is helpful to minimize the recovery period after temporary overload. We choose the weights $\xi_1 = 0.5$, $\xi_2 = 1$, and let $\alpha = 100$. The corresponding weighted min-max fair point is $(1/3, 2/3)$. The measured queue overflow vector under the JSQT policy is $(0.3340, 0.6596)$, which is point B in Fig. 3.

In addition, if we assign to the two servers different weights, i.e., $\xi_1 \neq \xi_2$, then different values of α correspond to different performance points. To illustrate this, we simulate the JSQT policy with weights $(\xi_1, \xi_2) = (0.5, 1)$ and different values of α. Fig. 4 shows that the JSQT policy yields the empirical queue overflow rates $\tilde{q}_1^{(\alpha)}$ for server 1 as follows: $\tilde{q}_1^{(0.1)} = 0.02$, $\tilde{q}_1^{(1)} = 0.21$, $\tilde{q}_1^{(10)} = 0.32$, and $\tilde{q}_1^{(100)} = 0.34$. As α increases, the queue overflow rate of server 1 approaches $q_1^{(\infty)} = 1/3$.

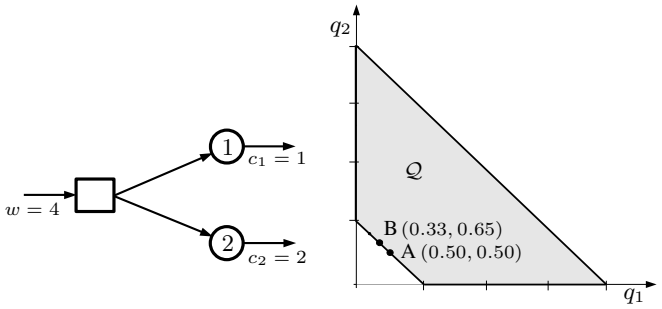


Fig. 3. The performance of the JSQT policy in the load-balancing example on the left, with different performance objectives: (a) the most-balanced vector $(1/2, 1/2)$; (b) the weighted min-max fair point $(1/3, 2/3)$.

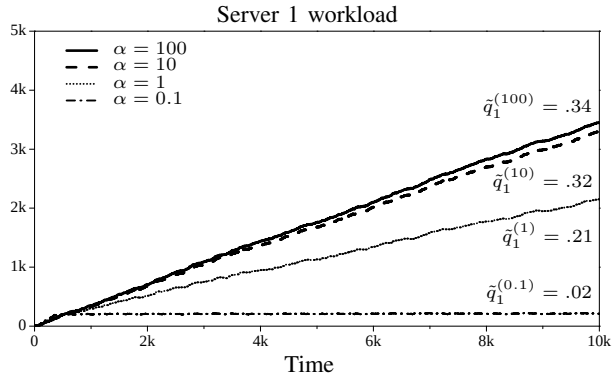


Fig. 4. The empirical workload process of server 1 under the JSQT policy with weights $(\xi_1, \xi_2) = (0.5, 1)$ and different values of α. As α increases, the empirical queue overflow rate $\tilde{q}_1^{(\alpha)}$, i.e., the slope of each workload process, approaches $q_1^{(\infty)} = 1/3$.

B. Behavior under bursty arrivals

The recovery time after a temporary overload surge is minimized by balancing queue overflows proportional to server capacities. Therefore, it is interesting to investigate the system performance under bursty arrivals where every burst corresponds to an overload surge. We study the two-server system in Fig. 3 with different server capacities $(c_1, c_2) = (1, 10)$. The load balancer has a Poisson arrival process with time-varying arrival rates alternating between $\lambda_{\text{low}} = 1$ and $\lambda_{\text{high}} = 20$ every 250 slots. We compare the JSQT policy and the JSQ policy in this setup. Note that the system is stable under both policies because $(w_{\text{low}} + w_{\text{high}})/2 = 10.5 < c_1 + c_2$. For JSQT, we choose $\xi_1 = 0.1$, $\xi_2 = 1$ and $\alpha = 100$ to achieve the weighted min-max fair point. Fig. 5 shows the sample paths of the two policies in the first 10^4 slots. We observe that the JSQT policy yields smaller backlogs, i.e., better delay performance, than the JSQ policy.

C. On the most-balanced queue overflow vector

The results of this paper show that the JSQT policy achieves the most-balanced vector by using the α-fair penalty functions for some $\alpha > 0$. On the other hand, the JSQ policy seeks to balance the server backlogs equally. This leads to the conjecture that the queue overflow performance of the JSQ policy achieves the most-balanced vector. Next, we compare the JSQT policy and the JSQ policy by simulations to support this conjecture.

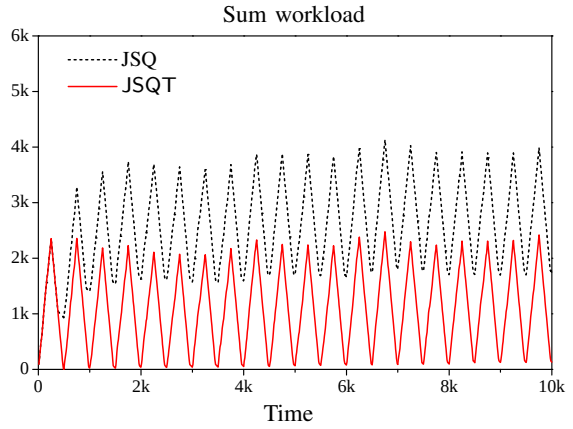


Fig. 5. Comparison of the JSQT and the JSQ policy in a stable system with bursty arrivals.

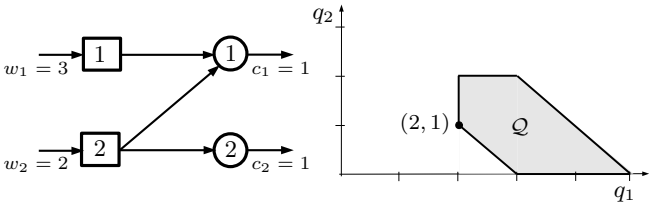


Fig. 6. (Left) A load balancing system used to examine the conjecture that the JSQ policy achieves the most-balanced queue overflow vector. (Right) The feasible set of queue overflow vectors with the most-balanced overflow vector $\boldsymbol{q}^{\text{MB}} = (2, 1)$.

Consider the example in Fig. 6 where two load balancers route jobs to two servers, each has unit capacity. The balancers receive jobs at the rates $(w_1, w_2) = (3, 2)$ and the reachable servers for each load balancer are $S_1 = \{1\}$ and $S_2 = \{1, 2\}$. The feasible set of queue overflow vectors is shown in Fig. 6. The most balanced vector in this set is $(2, 1)$. We simulate the JSQ and the JSQT policy using the same sample path of the arrival process for 10^4 slots, where we choose $V = \alpha = 100$ and $(\xi_1, \xi_2) = (1, 1)$ for the JSQT policy. We observe the following empirical queue overflow rates for both policies: $\tilde{\boldsymbol{q}}^{\text{JSQ}} = (2.0116, 0.9901)$ and $\tilde{\boldsymbol{q}}^{\text{JSQT}} = (2.0117, 0.99)$. Fig. 7 presents the sample paths of the workload processes under the two policies. The subfigure in Fig. 7 shows that the difference between the two sample paths is very small.

VI. Conclusion

This paper studies the problem of balancing queue overflow rates in a server farm under overload conditions, formulated as a convex penalty minimization problem. We introduce the class of α-fair penalty functions that are useful to achieve desired queue overflow vectors, including those that maximize the time to first buffer overflow and minimize the recovery time from temporary overload. We show that throughput optimality is always achieved by policies that minimize strictly increasing overload penalty functions. A generalized JSQ policy is proposed to optimize queue overflow rates via intelligent job tagging in a stochastic setting. The simulation results show that our policy yields better delay performance than the JSQ policy, and that the JSQ policy renders the most-balanced queue overflow vector.

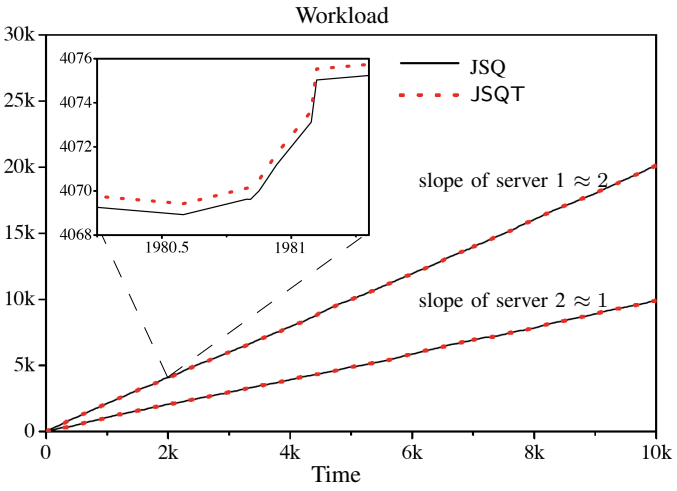


Fig. 7. The sample paths of the workload processes of the JSQ policy and the JSQT policy.

Broadly speaking, this paper provides a useful analysis to study stochastic control in more complex overloaded systems. There are many future research directions. One is to extend the results in this paper to multi-commodity multi-hop networks in overload conditions. In such problems, a control policy that optimizes queue overflow rates may not be throughput optimal, and it is of interest to characterize the joint queue overflow-throughput performance region and develop stochastic control policies there. Another interesting direction is to study robust control in overloaded systems with arbitrarily time-varying demands. In this scenario, it is of interest to design unified control algorithms that seamlessly provide optimal load balancing in both underload cases (i.e., balancing flows over links) and overload cases (i.e., balancing queue overflows); see [11] for an example.

REFERENCES

[1] D. Shah and D. Wischik, "Fluid models of congestion collapse in overloaded switched networks," *Queueing Syst.*, vol. 69, pp. 121–143, 2011.

[2] J. Mo and J. Walrand, "Fair end-to-end window-based congestion control," *IEEE/ACM Trans. Netw.*, vol. 8, no. 5, pp. 556–567, Oct. 2000.

[3] D. P. Bertsekas and R. G. Gallager, *Data Networks*, 2nd ed. Prentice Hall, 1992.

[4] O. Perry and W. Whitt, "A fluid limit for an overloaded X model via a stochastic averaging principle," *Math. of Oper. Res.*, vol. 38, no. 2, pp. 294–349, 2013.

[5] R. Talreja and W. Whitt, "Fluid models for overloaded multiclass many-server queueing systems with first-come, first-served routing," *Manage. Sci.*, vol. 54, no. 8, pp. 1513–1527, Aug. 2008.

[6] S. Tekin, S. Andradottir, and D. G. Down, "Dynamic server allocation for unstable queueing networks with flexible servers," *Queueing Syst.*, vol. 70, no. 1, pp. 45–79, 2012.

[7] K. Katsalis, G. S. Paschos, L. Tassiulas, and Y. Viniotis, "Dynamic CPU scheduling for QoS provisioning," in *Mini-Conference of IFIP/IEEE Intl. Symp. Integrated Network Management (IM)*, 2013.

[8] L. Georgiadis and L. Tassiulas, "Optimal overload response in sensor networks," *IEEE Trans. Inf. Theory*, vol. 52, no. 6, pp. 2684–2696, Jun. 2006.

[9] R. Egorova, S. Borst, and B. Zwart, "Bandwidth-sharing in overloaded networks," in *Conf. Information Science and Systems (CISS)*, Princeton, NJ, USA, Mar. 2008, pp. 36–41.

[10] C. W. Chan, M. Armony, and N. Bambos, "Fairness in overloaded parallel queues," 2011, working paper.

[11] A. L. Stolyar and T. Tezcan, "Shadow-routing based control of flexible multiserver pools in overload," *Oper. Res.*, vol. 59, no. 6, pp. 1427–1444, Nov.-Dec. 2011.

[12] M. Welsh and D. Culler, "Adaptive overload control for busy internet servers," in *USENIX Symp. Internet Technologies and Systems*, 2003.

[13] H. Chen and P. Mohapatra, "Session-based overload control in QoS-aware web servers," in *IEEE Proc. INFOCOM*, 2002.

[14] B. Radunovic and J.-Y. Le Boudec, "A unified framework for max-min and min-max fairness with applications," *IEEE/ACM Trans. Netw.*, vol. 15, no. 5, pp. 1073–1083, Oct. 2007.

[15] D. P. Bertsekas, *Nonlinear Programming*, 2nd ed. Athena Scientific, Sep. 1999.

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

[17] F. P. Kelly, "Charging and rate control for elastic traffic," *European Trans. Telecommunications*, vol. 8, pp. 33–37, 1997.

[18] V. Gupta, M. Harchol-Balter, K. Sigman, and W. Whitt, "Analysis of join-the-shortest-queue routing for web server farms," *Performance Evaluation*, vol. 64, no. 9-12, pp. 1062–1081, Oct. 2007.

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

[20] J. H. van Lint and R. M. Wilson, *A Course in Combinatorics*, 2nd ed. Cambridge University Press, 2001.

APPENDIX A
PROOF OF THEOREM 1

Our proof needs the construction of a useful graph G. Given the exogenous traffic rates $(w_b, b \in B)$, we define a graph $G = (V, E)$ with the node set V consisting of a source node σ, a sink node τ, load balancers $b \in B$, and servers $s \in S$. The source σ is connected to each load balancer b with a directed link (σ, b) of capacity $c_{\sigma b} = w_b$. Each server s is connected to the sink τ with a link (s, τ) of capacity c_s. There is an infinite-capacity link $(b, s) \in E$ if balancer b has access to server s. An important observation is that the maximum throughput in the server farm is achieved by the maximum flow in the graph G. We denote by r^* the maximum throughput.

Consider a superflow $\boldsymbol{w} = (w_{bs}, w_{s\tau}, q_s)_{b\in B, s\in S}$, defined in Section II-A, in the graph G. This superflow $\boldsymbol{w}$ must saturate all links (σ, b) in G because it injects flow rate w_b into load balancer b, incurring an overflow rate q_s at server $s \in S$. Consider pruning flows from the superflow $\boldsymbol{w}$ to remove all queue overflow rates q_s; there may be more than one way of pruning. We get an *induced flow* $\boldsymbol{f} = (f_{\sigma b}, f_{bs}, f_{s\tau})_{b\in B, s\in S}$ according to the standard flow definition. That is, the induced flow $\boldsymbol{f}$ satisfies

$$f_{\sigma b} = \sum_{s\in S} f_{bs}, \ \ b \in B, \tag{28}$$

$$\sum_{b\in B} f_{bs} = f_{s\tau}, \ \ s \in S, \tag{29}$$

$$f_{s\tau} = w_{s\tau}, \ \ s \in S, \tag{30}$$

$$f_{bs} \le w_{bs}, \ \ (b, s) \in E, \ \ f_{\sigma b} \le w_b, \ \ (\sigma, b) \in E. \tag{31}$$

Equations (28) and (29) represent flow conservation at nodes b and s, respectively. Equation (30) shows that the flow $w_{s\tau}$ from server s to sink τ is the throughput of server s because only queue overflows are removed from $\boldsymbol{w}$. Inequalities (31) result from pruning flows over links (b, s) and (σ, b). We observe that two flows $\boldsymbol{f}_1$ and $\boldsymbol{f}_2$ induced from $\boldsymbol{w}$ must deliver the same total throughput.

To prove the theorem by contradiction, consider a feasible superflow $\boldsymbol{w}$ that is not throughput optimal, i.e., any induced

flow $\boldsymbol{f}$ from $\boldsymbol{w}$ has the total throughput $\sum_s f_{s\tau} < r^*$. We show that the queue overflow rates q_s in $\boldsymbol{w}$ cannot be the solution to (9)-(10). Since the induced flow $\boldsymbol{f}$ is not throughput optimal, $\boldsymbol{f}$ cannot be the maximum flow in the graph G from the above observation. From the Augmenting Path Theorem [20, Chap. 7], there must exist an acyclic augmenting path p from σ to τ in G. Due to the structure of the graph G, the acyclic path p must be of the form

$$\sigma \to b_1 \to s_1 \to b_2 \to s_2 \to \cdots \to b_m \to s_m \to \tau, \quad (32)$$

where $b_i \in B$, $s_i \in S$, and $m \geq 1$. By definition of an augmenting path, the following inequalities are satisfied in the path p: (i) $f_{\sigma b_1} < c_{\sigma b_1} = w_{b_1}$, (ii) $f_{s_m\tau} < c_{s_m}$, (iii) $f_{b_{i+1}s_i} > 0$ for $i = 1, \ldots, m-1$. The next lemma is useful.

Lemma 2. Given the augmenting path p in (32), load balancer b_1 is connected to a server s_0 that has positive queue overflow $q_{s_0} > 0$ and positive flow $w_{b_1 s_0} > 0$ over the link (b_1, s_0).

Proof of Lemma 2: From the augmenting path property $f_{\sigma b_1} < w_{b_1}$, load balancer b_1 must prune a positive flow from the super $\boldsymbol{w}$ over link (b_1, s_0) for at least one connected server $s_0 \in S_b$. Therefore, the server s_0 exists and $w_{b_1 s_0} > 0$. In addition, since only queue overflow is pruned, the server s_0 must have positive overflow $q_{s_0} > 0$. ∎

Now, among the servers $\{s_0, s_1, \ldots, s_m\}$ that have positive queue overflow rates, consider the server s_{i^*} closest to s_m:

$$i^* = \max_{0 \leq i \leq m} \{i \mid q_{s_i} > 0\}.$$

Server s_{i^*} exists because s_0 is a feasible choice. Consider three cases: (i) $s_{i^*} = s_m$, (ii) $s_{i^*} \notin \{s_0, s_m\}$, and (iii) $s_{i^*} = s_0$.

(i) When $s_{i^*} = s_m$, we have $q_{s_m} > 0$. From the augmenting path property, we have $w_{s_m\tau} = f_{s_m\tau} < c_{s_m}$. It is feasible to remove an ϵ amount of flow out of q_{s_m} to improve the throughput of server s_m from $w_{s_m\tau}$ to $w_{s_m\tau} + \epsilon$. As a result, the queue overflows of all servers remain the same except that the queue overflow of server s_m is improved, strictly reducing the overflow penalty. This contradicts that the superflow $\boldsymbol{w}$ is the optimal solution to (9)-(10).

(ii) When $s_{i^*} \notin \{s_0, s_m\}$, we have $q_s = 0$ for all servers $s \in \hat{S} = \{s_j \mid i^* < j \leq m\}$. It follows that, under the superflow $\boldsymbol{w}$, the throughput $f_{s\tau}$ of each server $s \in \hat{S}$ is equal to the total incoming rate at server s, i.e.,

$$\sum_{b \in B_s} w_{bs} = f_{s\tau} = \sum_{b \in B_s} f_{bs}, \quad s \in \hat{S}. \quad (33)$$

Since $f_{bs} \leq w_{bs}$ by (31), equality (33) indicates that

$$f_{bs} = w_{bs}, \quad \text{for all } s \in \hat{S} \text{ and all } b \in B_s. \quad (34)$$

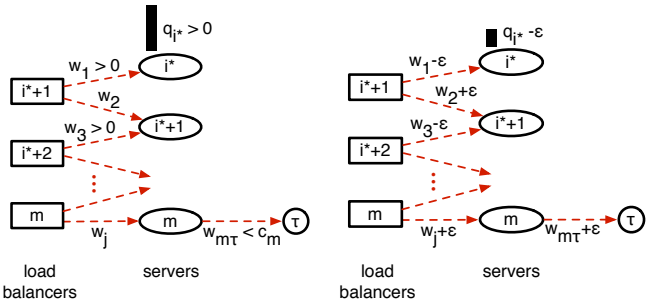
Now, consider the path

$$s_{i^*} \to b_{i^*+1} \to s_{i^*+1} \to \cdots b_m \to s_m \to \tau, \quad (35)$$

which is part of the augmenting path p in (32). From the augmenting path property, we have

$$w_{b_{i^*+1}s_{i^*}} = f_{b_{i^*+1}s_{i^*}} > 0 \quad (36)$$

$$w_{b_{i^*+2}s_{i^*+1}} = f_{b_{i^*+2}s_{i^*+1}} > 0 \quad (37)$$



(a) the superflow $\boldsymbol{w}$ (b) the adjusted superflow $\boldsymbol{w}'$

Fig. 8. An illustration of adjusting the superflow $\boldsymbol{w}$ by removing ϵ queue overflow from s_{i^*} and forwarding it along the red-dashed augmenting path to server s_m for throughput improvement.

$$\vdots$$

$$w_{b_m s_{m-1}} = f_{b_m s_{m-1}} > 0 \quad (38)$$

where the equalities follow (34). See Fig. 8(a) for an example. Consider a new superflow $\boldsymbol{w}'$ that takes an ϵ amount of queue overflow from server s_{i^*} and forwards it along the augmenting path in (35) to improve the throughput of server s_m by ϵ. In particular, define

$$q'_{s_{i^*}} = q_{s_{i^*}} - \epsilon \quad (39)$$

$$w'_{b_{i^*+1}s_{i^*}} = w_{b_{i^*+1}s_{i^*}} - \epsilon \quad (40)$$

$$w'_{b_{i^*+1}s_{i^*+1}} = w_{b_{i^*+1}s_{i^*+1}} + \epsilon \quad (41)$$

$$w'_{b_{i^*+2}s_{i^*+1}} = w_{b_{i^*+2}s_{i^*+1}} - \epsilon \quad (42)$$

$$\vdots$$

$$w'_{b_m s_m} = w_{b_m s_m} + \epsilon \quad (43)$$

$$w'_{s_m\tau} = w_{s_m\tau} + \epsilon \quad (44)$$

See Fig. 8(b) for an illustration of the superflow adjustment. The new allocations (40) and (42) are feasible because of the augmenting path property in (36)-(38). The allocations (41) and (43) are feasible because the links (b, s) are assumed to have infinite capacity. The allocation (39) reduces the queue overflow at server s_{i^*} and is feasible because $q_{s_{i^*}} > 0$. The allocation (44) improves the throughput of server s_m and is feasible because $w_{s_m\tau} < c_{s_m}$ by the augmenting path property. It is easy to observe that the queue overflow rates of all servers are the same expect that the queue overflow of server s_{i^*} is improved. Consequently, the superflow $\boldsymbol{w}$ cannot be the optimal solution to (9)-(10), which is a contradiction.

(iii) The case of $s_{i^*} = s_0$ is almost the same as the case (ii), and thus the analysis is omitted. We remark that the analysis here requires the condition $w_{b_1 s_0} > 0$, which is shown in Lemma 2.