Optimal Blind and Adaptive Fog Orchestration under Local Processor Sharing

Francesco De Pellegrini[◇⋆], Francescomaria Faticanti[⋆‡], Mandar Datar[†], Eitan Altman[†] and Domenico Siracusa[⋆]

Abstract—**This paper studies the tradeoff between running cost and processing delay in order to optimally orchestrate multiple fog applications. Fog applications process batches of objects' data along chains of containerised microservice modules, which can run either for free on a local fog server or run in cloud at a cost. Processor sharing techniques, in turn, affect the applications' processing delay on a local edge server depending on the number of application modules running on the same server. The fog orchestrator copes with local server congestion by offloading part of computation to the cloud trading off processing delay for a finite budget. Such problem can be described in a convex optimisation framework valid for a large class of processor sharing techniques. The optimal solution is in threshold form and depends solely on the order induced by the marginal delays of N fog applications. This reduces the original multidimensional problem to an unidimensional one which can be solved in $O(N^2)$ by a parallelised search algorithm under complete system information. Finally, an online learning procedure based on a primal-dual stochastic approximation algorithm is designed in order to drive optimal reconfiguration decisions in the dark, by requiring only the unbiased estimation of the marginal delays. Extensive numerical results characterise the structure of the optimal solution, the system performance and the advantage attained with respect to baseline algorithmic solutions.**

Index Terms—**fog computing, processor sharing, convex optimisation, stochastic approximation**

I. Introduction

Fog-computing parts from the model of a cloud-based IoT service by displacing computation at the edge of the network. Information flows generated by objects and mobile applications can thus be processed by edge servers in proximity [1]. Moving service's tasks closer to target objects has gained consensus since it solves several technical issues at once. Virtualisation techniques on edge gateways simplify management and maintenance of IoT services [2], copes with heterogeneity of IoT technologies [3] while local computation overcomes privacy issues by confining sensitive raw data at the data owner's premises [4]. Finally, local servers can save running costs compared to remote execution in cloud where charges may involve, for instance, number of connected IoT devices, IoT events detection, and/or remote device management [5].

Fog applications adhere to the modular microservice paradigm, the de-facto application design standard in cloud computing, where monolithic solutions are deprecated. Modularisation of applications into microservice modules increases availability and scalability [6]. Services can be assembled using basic building components, e.g., sensor reading modules, graphical user interfaces, monitoring units, etc. By cascading such components, it is possible to grant scalability, minimality and cohesiveness of the resulting application architecture.

In a typical fog or cloud service, data batches generated from target objects are processed sequentially over a sequence of containerised microservice modules [7], [8], briefly modules in the rest of the paper. In principle, the corresponding virtual machines or containers can be run either in the central cloud or run hosted on a fog server. The fog orchestrator is the controller which decides for each application which part of the computation, i.e., which modules, should run on the fog server and which ones in cloud [8], [9]. The resulting placement has a key impact on the processing delay experienced by fog applications. In fact, even a top-notch edge server has limited capacity compared to the aggregated capacity of overprovisioned cloud systems, so that processor sharing on a fog server may induce unacceptably long processing delays. Hence, optimisation of computation capacity of edge units to meet customers' demands is emerging as a central problem in fog computing [3], [10]. In this paper, fog applications are assumed to receive data batches to process at given rate. Each data batch entails a given processing delay to be processed by an application module.

Under processor sharing, concurrent applications' modules run in parallel on the same fog server, at the price of increased processing delays. However, in case of performance degradation, it is still possible to migrate part of the running applications to the cloud. Throughout this work, the objective is to study the trade-off between the load on local edge servers, reflecting in the processing delay, and the cost for offloading to the cloud. To this aim, the fog orchestrator can tune the performance of the system by increasing or decreasing the number of modules of an application which are executed on the fog server. The resulting control problem is a constrained convex problem where the control is the number of modules to be run in fog for each application.

Related Work. In IoT and fog scenarios, applications are not monolithic, but formed by decoupled and interdependent modules [7]. A few recent works acknowledge the microservice structure to design placement procedures for applications in edge/fog computing [8], [9]. Queuing models for independent chains of microservice applications of the type considered here appear in [9]. More advanced DAG-connected, modular architectures of the type studied in [8] will be part of future works. In the fog computing literature very few studies take

[◇]Avignon Université, Avignon (France),[⋆]Fondazione Bruno Kessler, Trento (Italy),[‡]University of Trento, via Sommarive, Trento, Italy,[†]INRIA, Sophia Antipolis (France). This work has received funding from the European Unions Horizon 2020 Research and Innovation Programme under grant agreement no. 815141 (DECENTER: Decentralised technologies for orchestrated Cloud-to-Edge intelligence).

ISBN 978-3-903176-29-4 © 2020 IFIP

into account the tradeoff between the local processing and the cloud cost for the deployment of multiple-modules applications. The objective here is to account for processor sharing effects on the placement of concurrent applications' chains in fog under the constraint of a limited budget for cloud usage.

Applications scaling and migration have been discussed extensively in the cloud literature [11]–[15]. Heuristic threshold policies have been broadly adopted before to solve feasibility problems [11], [12]. Reactive migration methods employ such thresholds to divert virtual machine instances from congested servers [13]. Conversely, the threshold policies described in this work result from the minimisation of the processing delay. Load balancing in cloud, based on multi-server queue sampling has been studied, e.g., in [14]. The learning algorithm is based on a queuing sampling scheme as well, but, it performs vertical load balancing between cloud and fog. Deterministic primal-dual algorithms appear, e.g., in [15]. The stochastic solution proposed here converges to the optimal policy with imperfect state information leveraging noisy estimates of processing delays. In [16], the authors propose an online algorithm for service reconfiguration of edge-clouds; while considering limited storage capacity of edge servers, processor sharing effects are not accounted for. Wang et al. [17] studied dynamic edge service migration via Markov Decision Processes (MDP), where migration depends on the desired service location. These two latter works represent the applications as monolithic services without considering a more general structure consisting of interdependent modules.

Processor sharing is usually described as a *preemptive* policy where jobs can be stopped and resumed through their execution [18]. It has been analysed in depth in the queuing literature [19], [20]. In that context, threshold policies as best responses have been identified [20] for a game where customers can choose to be served on a private machine or on a remote mainframe. A main difference with respect to those models is that in the proposed framework, applications incur a service rate slowdown because all application queues run simultaneously on the same server without job interruptions.

Main Contributions. The first part of this work focuses on the structure of the solution by minimising the cumulative batch processing delay. Under a general processor sharing policy, the optimal solution is determined by 1) the budget expenditure and 2) by a crucial metric, namely the *marginal delay* of applications. The marginal delay represents the performance gain obtained by offloading a module of a tagged application towards the cloud. The N-dimensional optimisation problem can thus be solved via a one-dimensional search in the space of the spent budget. The optimal solution sorts applications in order of increasing marginal delays. The optimal policy is of threshold type in all cases of practical interest, randomised on at most one control. In the second part of the paper, the precise knowledge of the structure of the optimal solution suggests a polynomial time algorithm to determine the optimal placement in $O(N^2)$ time. Furthermore, the optimal policy can be adjusted in case the system load varies in time using an adaptive algorithm based on stochastic approximations of the Robinson-Monroe type, whose convergence proof is derived for a primal-dual convex minimisation using the ODE method [21]. It is worth noting that this type of solution is able to converge to the optimal threshold policy in the dark: this is key when there is no apriori information on certain system parameters. Specifically, this is crucial when facing unknown data batch arrival rates or unknown applications' processing rates, either in cloud, in fog or both. The proposed algorithm converges to the optimal solution leveraging only online unbiased estimates of the processing delay. More important, provided that the processing delays are increasing and convex in the server's load, blind convergence to the optimal restpoint shall hold *irrespective of the fog server processor sharing policy*.

The rest of the paper is organized as follows. The next section describes the system model and Sec. III the fog placement problem. Sec. IV develops results on the threshold structure of the optimal solution and the marginal delays' role. In Sec. V a scalable algorithm solves the problem under perfect system information, whereas Sec. VI develops a learning algorithm able to converge to the optimal policy in the dark. Numerical results are reported in Sec. VII, and a concluding section ends the paper. Some proofs not reported in the current manuscript can be found in [22].

II. System Model

Let consider a set of N fog applications. Each application i is composed of n_i microservice modules. Batches of data are received by the first module of the i-th application at some rate λ_i and processed sequentially along the chain formed by the other modules downstream. For each application i, the first u_i modules can be placed in fog, whereas the rest of the chain, i.e., the $n_i - u_i$ modules downstream, in cloud. Thus, $0 \leq u_i \leq n_i$ is a control variable representing the number of modules of application i placed on the fog server. Under policy $\mathbf{u} = (u_1, \ldots, u_n)$, the load of modules running on the fog server is $u = \sum_{i=1}^{N} u_i$.

When the fog server is serving u modules, computing resources are shared among the modules running on the server. Let $G_i(u)$ be the batch sojourn time for a module of application i, when the fog server hosts u modules (dependence on λ_i is omitted for notation's sake): it is the time elapsing from the instant when a data batch is received from the tagged module till the end of the processing, after which the resulting output is sent to the module downstream.

Processor sharing techniques reduce application's processing rate when multiple modules are hosted on the server. Every application module running on the fog server is subject to a stability condition, that is there exists critical load $u_{i,\max}$ such that for each $u \geq u_{i,\max}$, it holds $G_i(u) = +\infty$, and $G_i(u) < +\infty$ for $u \leq u_{i,\max}$. Within their respective stability region $S_i = \{1 \leq u \leq u_{i,\max}\}$, the $G_i(u)$s are assumed convex increasing in the fog server load.

Conversely, since cloud systems provide a large number of servers, d_i denotes the constant average processing delay for application i modules running in cloud. Finally, the processing

TABLE I
MAIN NOTATION USED THROUGHOUT THE PAPER

Symbol	*Meaning*
N	number of apps
b_0	budget to place apps in cloud
b	min. number of modules in fog $b := \sum n_i - b_0/c$
c	cost to place one app module in cloud
n_i	number of microservice modules for app i
u_i	number of modules in fog of app i
$\mathbf{u}$	placement policy $\mathbf{u} = (u_1, \ldots, u_N)$
u	fog server load $u = \sum_i u_i$
$u_{\max}$	maximum number of modules in fog
$G_i(u)$	proc. delay of app i modules fog batch at load u
μ_i	fog service rate for app i modules (M/M/1 PS)
λ_i	data batches per second towards app i (M/M/1 PS)
d_i	batch processing time in cloud for app i
D_i	batch processing delay for app i
D	cumulative processing delay $D := \sum_i D_i$

delay of a data batch consumed by application i is given by the following governing equation

$$D_i(\mathbf{u}) = u_i G_i(u) + (n_i - u_i)\, d_i \tag{1}$$

In (1) the last $n_i - u_i$ modules of application i are executed in cloud: let c the cost paid to place a module in cloud. Also, the overall budget available to place modules in cloud is denoted $b_0 \geq 0$; $b := (\sum n_i - b_0/c) \geq 0$ is the minimum number of modules to be placed in fog. Finally, the cumulative delay $D(\mathbf{u}) := \sum_i D_i(\mathbf{u})$ is the target utility function used in the rest of the paper. In the rest of the paper, $\mathbf{u}$ is continuous control vector, where the placement of the last module in fog of application i occurs with probability $u_i - \lfloor u_i \rfloor$, so that governing equation (1) is still exact for integer values. Relaxing the continuous control (1), the objective function introduced next $\sum_{i=1}^{N} D_i(\mathbf{u})$ becomes a smooth convex interpolation, as tighter as larger is the number of application modules.

*Benchmark model:*let consider a M/M/1 queue for the i-th application module. The service rate is μ_i data batches per second when the module runs alone on the server (let $\mu_i > \lambda_i$ in order to avoid trivial conditions). Conversely, when the server CPU is shared across application modules, applications continue processing in parallel but with a slowdown factor for the batch processing rate: under policy $\mathbf{u}$, service rate becomes $\mu_i(u) = \mu_i/u$ data batches per second. For the slowed M/M/1 case, the explicit expression for the batch processing delay writes $G_i(u) = (\frac{\mu_i}{u} - \lambda_i)^{-1}$. By direct calculations the $G_i(x)$s, in this case, are convex increasing in the stability region. Note that, while this model is handy to explain the theoretical development, the results derived in the following apply in general to any convex increasing $G_i(u)$.

III. Problem Formulation

The objective of the fog orchestrator is to optimally place the modules of each application in between a fog server and the cloud. The optimal placement policy can be determined according to the following formulation

Problem 1. Fog Placement

$$\begin{aligned}
\underset{\mathbf{u}}{\text{minimize}} \quad & \sum_{i=1}^{N} D_i(\mathbf{u}) \\
\text{subject to} \quad & \sum_{i=1}^{N} c \cdot (n_i - u_i) \leq b_0 && (2)\\
& u_i > 0 \quad \textit{iff} \quad u \in S_i, \quad i = 1, \ldots, N && (3)\\
& 0 \leq u_i \leq n_i, \quad i = 1, \ldots, N && (4)
\end{aligned}$$

At this point, let precise the search space of the optimal solution. Since processor sharing techniques inevitably reduce application's processing rate, constraint (3) forces the orchestration strategy to be such that all applications can complete batch processing within a finite time.

For the sake of concreteness, let refer to the benchmark model: there such constraint has the familiar form

$$\sum u_i < \frac{\mu_i}{\lambda_i}, \quad i = 1, \ldots, N$$

The processor sharing technique running on the fog server grants the stability condition for application i if and only if $u < \mu_i/\lambda_i$. Thus, when the search of an optimal solution $\mathbf{u}$ is performed in an interval where $u > \mu_i/\lambda_i$ for some application i, a candidate optimal solution needs to be such that $u_i = 0$, so that $D_i(\mathbf{u}) = d_i$ and the corresponding constraint can be removed. It follows immediately that the original problem maps into (at most) N subproblems, one for each partition induced by the stability conditions (3), which can be solved in parallel to finally determine the optimal solution over the set of the (at most) N local minima. Each of such problems, as showed in the next section, is convex.

In order to keep the discussion simple, in the rest of the paper such regions of instability are excluded, by assuming $u_{i,\max} \leq u_{\max}$, where $u_{\max} = \sum n_i$; in the case benchmark model, this entails $\mu_i > \lambda_i u_{\max}$ for all $i = 1, \ldots, N$. The analysis hence restricts to the case when $u \in [b, u_{\max}]$ so as to neglect bound (3).

The next section performs the general analysis of the problem and characterises the optimal solution. Before, the benchmark model for a single application serves as a concrete introduction to the general case.

A. Benchmark model for $N = 1$

From direct calculations on (1), in this case $D(u) = u/(\mu/u - \lambda) + d(n - u)$. Thus, the problem is apparently strictly convex in the stability region. Dropping all indexes for the notation's sake, the unique optimal fog placement control writes

$$u^* = \begin{cases} n & \text{if} \quad 0 < \lambda \leq \underline{\lambda} \\ \overline{u} & \text{if} \quad \underline{\lambda} < \lambda < \overline{\lambda} \\ b & \text{if} \quad \lambda \geq \overline{\lambda} \end{cases} \tag{5}$$

where $\overline{u} := \frac{\mu}{\lambda}(1 - (1 + \lambda d)^{-\frac{1}{2}})$ is the unique solution of the unconstrained minimization problem. Here, $\underline{\lambda}$ and $\overline{\lambda}$ are two thresholds: $\underline{\lambda}$ is the unique positive solution of $\frac{1}{\sqrt{1+xd}} = 1 - n\frac{x}{\mu}$ if $d < 2n/\mu_0$ or $\underline{\lambda} = +\infty$ otherwise; $\overline{\lambda}$ is the unique positive solution of $\frac{1}{\sqrt{1+xd}} = 1 - b\frac{x}{\mu}$ if $d < 2b/\mu$ or $\overline{\lambda} = +\infty$ otherwise; since $b \leq n$, then it holds $\underline{\lambda} \leq \overline{\lambda}$.

Even this simple case reveals a threshold structure varying with continuity with λ from very low batch arrival rates, where modules are all placed in fog, to high batch arrival rates, where it is optimal to place as many modules in cloud as possible, due to large delays introduced by the fog server.

IV. Optimal Policy

Extending the result obtained for $N = 1$ to the general case, let identify a metric able to sort applications in order of importance, i.e., represent the convenience of storing an application module either in cloud or in fog.

First, let characterise the convexity of the problem. Using auxiliary functions $u_i G_i(\sum u_i)$, the Hessian of $D(\mathbf{u})$ is positive definite according to the following result, whose proof is reported in [22].

Theorem 1. *In the stability region $\sum_{i=1}^N D_i(\mathbf{u})$ is strictly convex so that Problem 1 has a unique solution.*

A. Marginal Delays

In order to find the structure of the optimal solution, the Lagrangian for the problem can be written as

$$\begin{aligned} L(\mathbf{u}, \boldsymbol{\alpha}, \boldsymbol{\beta}, \gamma) \qquad &= \sum_{j=1}^{N} u_j \Big(G_j(u) - d_j \Big) - \sum_{j=1}^{N} \alpha_j u_j \\ &- \sum_{j=1}^{N} \beta_j (n_j - u_j) - \gamma (u - b) \end{aligned}$$

The constraint (3) does not appear since in the stability region the corresponding multipliers must vanish. For any policy $\mathbf{u}$ let define the key metric used in the rest of the paper

Definition 1. Marginal delay: *the marginal delay of application i under policy $\mathbf{u}$ is the parameter $r_i(u) := G_i(u) - d_i$.*

The interpretation of this parameter is immediate: it is the difference of the batch processing delay in fog $G_i(u)$ and in cloud d_i. I.e., it measures the increase of processing delay when an application is placed on the fog node instead of in cloud, under load u.

The marginal delays appear in the application of KKT conditions; here they are necessary and sufficient for Problem 1 since all constraints are affine [23]. First order KKT conditions write

$$L_{u_i} = r_i + \sum u_j \dot{G}_j(u) - \alpha_i + \beta_i - \gamma$$

The optimal solution will correspond to the set of nonnegative multipliers $\boldsymbol{\alpha}^*$, $\boldsymbol{\beta}^*$ and γ^*. Previous relations and complementary slackness bring

$$\beta_j = -r_j + \gamma - \sum_{j=1}^{N} u_j \dot{G}_j(u), \; \alpha_j = 0, \quad \text{for } j = 1, \ldots, S$$

$$\alpha_j = r_j - \gamma + \sum_{j=1}^{N} u_j \dot{G}_j(u), \; \beta_j = 0, \quad \text{for } j = V+1, \ldots, N$$

where indexes are sorted such that $u_j = 0$ for $j = 1, \ldots, S$ and $u_j = n_j$ for $j = V+1, \ldots, N$, respectively.

Now, for a given control vector $\mathbf{u}$, it is possible to sort conveniently the indexes based on complementary slackness conditions.

Proposition 1. *Let $r_1 \leq \ldots \leq r_S$ and $r_{V+1} \leq \ldots \leq r_N$, then $r_1 \leq \ldots \leq r_S \leq r_{S+1} = \ldots = r_V \leq r_{V+1} \leq \ldots \leq r_N$.*

Since $S \leq V$, denote $S = V = 0$ to indicate that all application modules are placed in cloud, whereas $S = 0$ but $V > 0$ means no application has been fully placed in fog; $S = V > 0$ indicates that all applications are entirely placed either in fog or cloud.

Now, it is possible to draw a few conclusions from Prop. 1:

1) For any optimal solution $\mathbf{u}^*$, the number of fog modules u^* induces the order according to which app modules should be placed in fog or cloud.

2) An optimal solution for which $\gamma = 0$, i.e., $\sum u_i > b$, implies that the constraint is not active, i.e., budget b_0 is not saturated. But, for such an optimal solution, it is always possible to replace the budget constraint such that $\sum u_i = b' > b$, i.e., resorting to a case where the constraint is active, e.g., $\gamma > 0$.

3) Once the structure of the optimal solution in the case $\gamma > 0$ has been determined, the optimal solution can be found in the one-dimensional space of parameters $b \leq u \leq u_{\max}$.

The above observations will be the basis for the algorithmic solutions solving Prob. 1 which are proposed in the following sections.

B. Quasi-threshold structure

Hereafter, let further describe the optimal solution $\mathbf{u}^*$ for $\gamma > 0$: the result is a specific waterfilling type of policy, which becomes a threshold policy for all cases of interest.

Let assume the budget was saturated, then two indexes for the optimal solution S^* and V^* can be characterised by the following result:

Theorem 2. *Let the optimal solution $\mathbf{u}^*$ be attained for $\gamma^* > 0$, i.e., $u^* = \sum u_i = b$, then*
i. $V^* = \min\{1 \leq j \leq N | \sum_{i=j+1}^{N} n_i \leq b_0\} := \overline{V}$
ii. If $\sum_{i=V^*+1}^{N} n_i = b_0$, then $S^* = V^*$, or else

$$I = \{j | r_j = r_{V^*} \; \wedge \; n_j d_j = n_{V^*} d_{V^*}\}$$

and $S^* = \min\{I\} - 1$.

The actual structure of the optimal solution is now derived. In order to simplify the discussion, with no loss of generality, whenever $r_i = r_j$, the indexes in I will be sorted for nonincreasing values of $n_i d_i$. The key role of the order of the r_is induced by u is reflected in the following

Definition 2. Marginal delays order. *For any value of $b \leq x \leq u_{\max}$, $u_{\max} := \sum n_i$, let $\sigma_x : \{1, \ldots, N\} \rightarrow \{1, \ldots, N\}$ be the permutation ordering of the marginal delays such that $r_{\sigma_x(1)} \leq \ldots \leq r_{\sigma_x(N)}$ and $n_{\sigma_x(i)} d_{\sigma_x(i)} \geq n_{\sigma_x(i+1)} d_{\sigma_x(i+1)}$ in case equality holds. Denote $\Sigma = \{\sigma_u | b \leq u \leq b_{\max}\}$ the set of such permutations in the stability region.*

The role of this definition is needed in the proof of the following

Theorem 3. *Let $\gamma^* > 0$, and assume $u^* = b$. Then the optimal solution $\mathbf{u}^*$ has the following structure*

$$u_i^* = \begin{cases} n_i & \text{if} \quad 1 \le i \le S^* \\ \frac{\gamma^* - r_{V^*} - \sum_{j=1}^{S^*} n_j \dot{G}_j(b)}{(V^* - S^*)\dot{G}_i(b)} & \text{if} \quad S^* + 1 \le i \le V^* \\ 0 & \text{if} \quad V^* < i \le N \end{cases} \tag{6}$$

where $V^* = V^*(b)$ and $S^* = S^*(b)$ are as in Thm. 2 and the order of the indexes is determined by σ_b.

Finally, when $\text{card}(I) = 1$, the solution is a threshold policy with at most one non-extremal control $u_{V^*} = b - \sum_{j=1}^{V^*-1} n_j$. In this case, indeed, the calculation of γ^* is not needed in order to determine the optimal solution. Conversely, when $\text{card}(I) > 1$, the policy is "almost threshold", i.e., it is extremal for all $i \notin I$. For the sake of notation, in the rest of the discussion, we assume $n_i d_i \neq n_j d_j$ for all $i, j = 1, \dots, N$; the next results can be extended to the case when equality holds for some index.

V. Algorithmic solution

Here we describe the idea of the Marginal Delays Threshold Algorithm (`MDTA`) presented in detail in [22]. The basic idea is to perform a search in the space of the threshold policies in the form determined in Thm.3, parametrised in the one-dimensional domain of the actual budget spent. First, it can be proved that the cardinality of the set of permutations Σ induced by the order of the r_is is polinomially bounded by the number of applications due to the monotonicity and convexity of marginal delays, $|\Sigma| = O(N^2)$. Hence, with this remark, it is possible to partition the interval $[b, u_{\max}]$ in subintervals defined by each permutation of the r_is within the stability region, $[b, u_{\max}] = \cup_{k=1}^{K-1} A_k$, where K is the maximum number of intersections between two different marginal delay functions. In this manner, proving that the objective function is piecewise convex in $u \in A_k$, for each k, it is possible to perform a bisection search on each interval computing the placement policy with the minimum delay on that interval. Finally, the minimum among all the intervals is taken.

The complexity of the sequential implementation is dominated by the number of permutations and the complexity of the bisection method, leading to $O(N^2 \log(\frac{u_{\max}}{\epsilon}))$, where ϵ is the error accuracy parameter. However, with a simple parallel implementation of the search over the intervals, the complexity can be reduced to $O(N^2)$. Furthermore, as showed in the numerical section, this estimation for the number of intervals σ is rather conservative and it appears almost linear in the input size. While the problem analysis was meant to describe the structure of the optimal solution, state of the art tools for convex optimisation can also be used to solve Prob. 1; however, general interior point methods and ϵ-accuracy methods rarely provide polynomial complexity bounds in the input size so that their scalability is debated [24].

Example. Fig. 1 reports on the dynamics of the r_is for an example for $N = 3$ with $n_1 = 8$, $n_2 = 9$ and $n_3 = 7$ modules, respectively. The order on the r_is is determined by the intersections $r_i(u) = r_j(u)$. The exploration space is partitioned according to the sets $\{A_k\}$, i.e., three valid intervals $A_k = [a_k, a_{k+1}]$, where $a_1 = b$, a_2 solves for $r_2(a_2) = r_3(a_2)$, a_3 solves for $r_1(a_3) = r_3(a_3)$ and $a_4 = u_{\max}$. In the example $b_0 = 24$, hence the minimum number of modules to be deployed on fog is $b = 0$: the optimal solution u^*, highlighted by vertical red line, lies in the first interval. Observe that $u^* > b$: the solution identifies the optimal fraction of the budget to be spent.

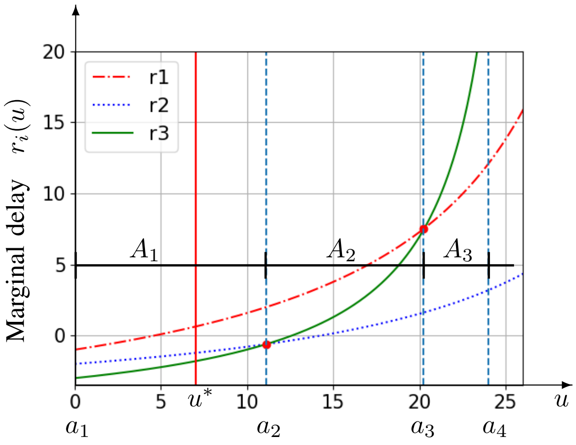

Fig. 1. Example for $N = 3$ with $\boldsymbol{\lambda} = (0.14, 0.26, 0.3)$ s^{-1}, $\boldsymbol{\mu} = (5.33, 10.9, 7.96)$ s^{-1}, $\boldsymbol{d} = (1, 2, 3)$ s, $n_1 = 8$, $n_2 = 9$, and $n_3 = 7$.

VI. Online Learning

In much part of current literature, cloud and fog orchestration is performed with an initial placement, typically based on nominal load values, and using later online migration techniques to reduce hostpots developing at runtime [13]. In fact, several system parameters may change over time: batch arrival rates $\{\lambda_i\}$, applications' fog processing delays $\{G_i\}$ and cloud processing delays $\{d_i\}$ may fluctuate around their nominal values or drift, e.g., due to variations in fog applications' workloads. They might even switch to different values as a consequence of sudden changes in operating conditions (e.g., objects data generation rates may change). As a result, the $\{r_i\}$s, considered so far as deterministic quantities, form in fact a random process. Let assume that the system is configured in some interior point of the domain $\mathbf{x}_0 > 0$: this is sufficient to obtain, for each tagged application i, the estimates $\{\widetilde{G}_{i,n}\}_{n \in \mathbb{N}}$ and $\{\widetilde{d}_{i,n}\}_{n \in \mathbb{N}}$ for the batch processing time in fog or in cloud, respectively. Such samples are generated at rate λ_i. If this is not the case, for the next scheme to work it is necessary to introduce probing schemes able to migrate some modules to the fog or the cloud in order to obtain the needed samples: for the sake of space, such schemes are out of the scope of the present work.

Hereafter, based on the structure of the optimal solution, a continuously adaptive procedure is designed. It works under the assumption that an optimal policy saturates the available budget, i.e., $\sum n_i - u_i^* = b_0$. The algorithm performs the optimisation of marginal delays

Problem 2. Marginal Delay Optimisation (MDO)

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimise}} \quad & -\sum_{i=1}^{N} r_i \cdot n_i \cdot x_i \\ \text{subject to} \quad & \sum_{i=1}^{N} n_i \cdot x_i - b_0 \leq 0 \\ & 0 \leq x_i \leq 1, \quad i = 1, \ldots, N \end{aligned}$$

where the identification is for the sake of notation $x_i = 1 - u_i/n_i$, obtaining a fractional knapsack problem [25]. It easy to verify that the optimal solution solves Prob. 1 under budget saturation. However, let assume to have just noisy measurements of both the objective function and the constraints. A tool to handle this situation are stochastic approximations [21]. Hence, hereafter a stochastic primal-dual optimisation algorithm of the family discussed in [26] is introduced. This entails a learning procedure of the Robinson-Monroe type, which is known to have efficient noise rejection properties [21]. However, it is convenient to replace the objective function in Prob. 2 with a convex one as follows

Problem 3. Convexified MDO

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimise}} \quad & f(\mathbf{x}) := -\sum_{i=1}^{N} r_i \cdot n_i \cdot \frac{1 - e^{-x_i}}{1 - e^{-1}} \\ \text{subject to} \quad & q(\mathbf{x}) := \Big(\sum_{i=1}^{N} n_i \cdot x_i - b_0\Big) \leq 0 \\ & 0 \leq x_i \leq 1, \quad i = 1, \ldots, N \end{aligned}$$

It is immediate to observe that the optimal control attained by Prob. 3 coincides with the solution Prob. 2 (even if the value of the attained minimum is different). The online learning procedure is based on the lagrangian associated to Prob. 3, thus minimising

$$L(\mathbf{x}, \theta) = f(\mathbf{x}) + \theta\, q(\mathbf{x}) - \boldsymbol{\xi} \cdot \mathbf{x} + \boldsymbol{\nu} \cdot (\mathbf{x} - \mathbf{1}) \tag{7}$$

where $\boldsymbol{\xi} \geq \mathbf{0}$ and $\boldsymbol{\nu} \geq \mathbf{0}$ are the multiplier vectors accounting for the box constraints $0 \leq x_i \leq 1$, for $i = 1, \ldots, N$, and $\theta \geq 0$ accounts for the coupled constraint.

The iteration for the update of the primal and of the dual variables writes as follows

$$\begin{aligned} x_{i,n+1} &= \Pi_{[0,1]} \Big[x_{i,n} - \varepsilon_n \widetilde{L}_{x_i}(\mathbf{x}_n, \theta_n)\Big] \qquad (8) \\ \theta_{n+1} &= \Pi_{\geq 0} \left[\theta_n + \varepsilon_n q(\mathbf{x}_{n+1})\right] \\ \xi_{i,n+1} &= \Pi_{\geq 0} \left[\xi_{i,n} - \varepsilon_n x_{i,n+1}\right] \\ \nu_{i,n+1} &= \Pi_{\geq 0} \left[\nu_{i,n} + \varepsilon_n (x_{i,n+1} - 1)\right] \end{aligned}$$

where $\{\varepsilon_n\}_{n\in\mathbb{N}}$ are the stepsizes of the algorithm, and obey to the following assumption

$$\varepsilon_n \geq 0, \qquad \sum_{n=0}^{+\infty} \varepsilon_n = +\infty, \qquad \sum_{n=0}^{+\infty} {\varepsilon_n}^2 < +\infty \tag{9}$$

In $(8), \Pi_{[0,1]}(y) = \max(0, \min(y, 1))$ and $\Pi_{\geq 0}(y) = \max(0, \min(y))$ denote projection functions.

Let $\widetilde{L}_{x_i}$ indicate noisy estimates of the k-th component of the gradient of the lagrangian: each time an estimate is produced, the i-th control component is updated, and the dual variables θ, $\boldsymbol{\xi}$ and $\boldsymbol{\nu}$ as well. The update instants triggering the updates in (8) are those when a new measurement of the marginal delay of application k is available. Actually, the explicit expression appearing in (8) is

$$\widetilde{L}_{x_i}(\mathbf{x}_n, \theta_n) = n_i \left(\frac{e}{e-1}(\widetilde{G}_i - \widetilde{d}_i)e^{-x_{i,n}} - \theta_n\right) - \xi_{i,n} + \nu_{i,n}$$

where $\widetilde{G}_i$ and $\widetilde{d}_i$ are estimates of the system time of modules in fog and in cloud.

The following result ensures the convergence to the unique solution of Prob. 2.

Theorem 4. *Let sequence* $\{\varepsilon_n\}$ *satisfy* (9) *and let the* $\widetilde{G}_i$ *and* $\widetilde{d}_i$ *be unbiased estimates with finite second order moments. Then the sequence of policies* $\mathbf{x}_n$ *converges to the optimal policy* $\mathbf{x}^*$ *with probability one.*

Finally, to increase of the numerical stability of the algorithm, in the numerical section it is employed a penalty function $p(\mathbf{x}) =: \frac{1}{2}\, p_0\, q^2(\mathbf{x})$ [27], where p_0 is a tunable penalty factor.

A. Adaptive Version

In order to adapt faster to changing parameters, it is possible to use a variant of the stochastic approximation technique where the step of approximations has small but constant size. Using the same algorithm, the constant stepsize $\varepsilon_n = \varepsilon > 0$ is used for all n: because the approximation step does not vanish over time, the system continues to adapt.

However, while for decreasing stepsizes convergence in probability to the solution of the KKT conditions is proved, for constant stepsizes convergence results are weaker. In particular, let consider neighborhoods of radius δ around the optimal solution $\boldsymbol{\varphi}^* = (\mathbf{x}^*, \theta^*, \boldsymbol{\xi}^*, \boldsymbol{\nu}^*)$ of the type $N_\delta(\boldsymbol{\varphi}^*) = \{\boldsymbol{\varphi} \in \mathbb{R}^{3N+1} : ||\boldsymbol{\varphi} - \boldsymbol{\varphi}^*||_2 < \delta\}$. The following result grants that, as long as a sufficiently small stepsize is chosen, the algorithm produces an output which remains confined around a suitable neighborhood of the optimal solution

Theorem 5. *For any* $\delta > 0$, *define by* $N_\delta(\boldsymbol{\varphi}^*)$*: for* $\epsilon \to 0$, *the sample paths of the algorithm defined in the iteration* (8) *for constant stepsize converge in distribution to elements in* $N_\delta(\boldsymbol{\varphi}^*)$. *The fraction of time spent by the process in* $N_\delta(\boldsymbol{\varphi}^*)$ *during* $[0, T]$ *goes to* 1 *as time horizon* T *diverges.*

The above result can be proved by verifying that Thm.2.1 at pp. 248 in [21] holds true.

VII. Numerical Results

In this section the characterisation of the optimal fog orchestration policy is completed by performing numerical exploration, as depicted in Fig. 2. Fig. 2a describes the optimal policy for a set of $N = 10$ applications adhering to the benchmark model, where the parameters of applications are chosen uniformly at random in their respective intervals, i.e., $n_i \in [1, 10], \lambda_i \in [0.1, 0.3]$ s^{-1}, $\mu_i \in [5, 16]$ s^{-1}, $d_i \in [0.5, 3.3]$ s. The index of the applications have been sorted according to the r_is corresponding to the optimal solution. The optimal solution has been produced by Matlab® nonlinear

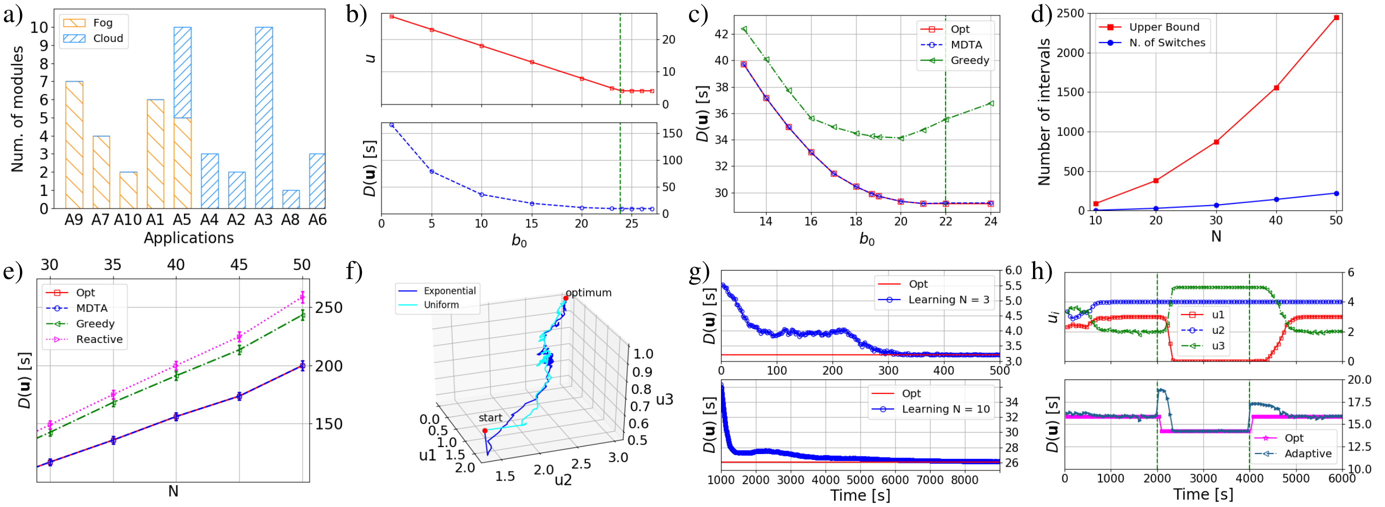

Fig. 2. a) Threshold structure of the optimal solution; b) Optimal control and cumulative delay for increasing budget b_0; c) Cumulative delay attained by Opt, MDTA, and greedy algorithm, respectively, for increasing budget b_0; d) Number of intervals $\{A_k\}$ and upper bound; e) Cumulative delay for Opt, MDTA, greedy algorithm, and reactive control, respectively; f) Convergence of the stochastic algorithm to the optimal solution;g) Convergence of the stochastic algorithm in terms of cumulative delay for $N = 3$ and $N = 10$; h) Dynamics of the three components and cumulative delay under the adaptive stochastic algorithm.

optimisation toolbox, which uses an interior point algorithm within the feasibility region. The optimal policy is clearly of threshold type. Furthermore, Fig. 2b depicts the behaviour of the optimal control and the cumulative delay at the increase of the budget available for offloading to the cloud. It is seen there that there exists a certain threshold ($b_0 = 23.8$ in the figure): above such value, in fact, the aggregated delay does not improve by further offloading to the cloud, i.e., with larger costs. Rather, the local server provides a computational advantage for a subset of the applications under moderate load. Fig. 2b confirms that, as a byproduct of the optimisation, one can obtain a precise answer to the practical dilemma whether or not saturating the available budget is optimal, whose answer is not obvious apriori from the system parameters. The figure also confirms that an identically null solution, $u = 0$, cannot be an optimal solution. Hence, even though the budget is sufficient to put all the applications in cloud, the optimal solution never takes into account this option when, as in the experiment, $r_i(1) < 0$ for some application i.

Fig. 2c describes the comparison, for increasing budget size, of the optimal solution obtained via the optimisation toolbox (Opt), the MDTA algorithm and a benchmark greedy, load-based fog orchestration algorithm (greedy). The last algorithm sorts the applications based on the nominal load of the i-th application, i.e., the ratio λ_i/μ_i, but neglects the effect of the processor sharing on the fog server. As proved in the theoretical development, MDTA attains indeed the optimal solution. The greedy algorithm has a significant performance loss with respect to the optimal policy for the same budget.

In the worst-case complexity analysis, the computational complexity of MDTA has been dominated by the number of the intervals A_k. Fig. 2d provides a numerical evaluation of the actual number of the intervals A_k. From that experiment the quadratic upperbound on the number of intervals appears conservative, and it is possible to conjecture that the complexity of MDTA result moderately superlinear in the input size.

Fig. 2e depicts the comparison, for increasing number of applications, of the optimal solution, the MDTA algorithm, the benchmark greedy algorithm and the reactive control algorithm [13]. The reported results are averaged over 100 instances, with 95% confidence interval, for a scenario with parameters $n_i \in [1, 5], \lambda_i \in [0.1, 0.3]$ s^{-1}, $\mu_i \in [10, 60]$ s^{-1} and $d_i \in [0.5, 3.3]$ s, under budget saturation.

Reactive control is a standard dynamic algorithm used in cloud for virtual machine (VM) migration: the algorithm performs online migrations in order to mitigate server overload conditions. When overload is detected, the VM with the highest ratio between the memory demand and the actual volume occupied by the VM is migrated. The actual migration is triggered when the server utilization goes beyond a certain threshold. This mechanism has been adapted to the fog placement problem by assuming that one application is placed in cloud every time the server utilization exceeds the threshold and, at the same time, the budget is sufficient to perform the offload. In the experiment of Fig. 2e, the threshold is fixed as 40% of the fog server utilization. Every time a violation condition is detected, the application with the highest load in fog is chosen for the placement in cloud. The figure shows that a fixed-threshold strategy leads to large cumulative delay, especially under limited cloud budget. Also, since an optimal threshold policy accounts for the effect of processor sharing, as expected, it outperforms the greedy solution, which is based on the nominal load of the applications.

Fig. 2f, Fig. 2g, and Fig. 2h evaluate the performance of the stochastic approximation algorithm. Fig. 2f shows the convergence of the algorithm to the optimal solution for

$N = 3$, where $\boldsymbol{\lambda} = (1.7, 1.2, 1.5)$ s^{-1}, $\boldsymbol{\mu} = (53.3, 108.9, 79.6)$ s^{-1}, $d_1 = 5$ s, $d_2 = 10$ s, $d_3 = 15$ s, $b_0 = 3$, $n_1 = 3$, $n_2 = 3$, and $n_3 = 1$. The convergence of the algorithm under an exponential and under a uniform distribution for the fog system times is tested, using single samples as estimators. The algorithm reaches the optimal solution within 5000 iterations with a constant step-size $\epsilon_n = 1/100$, irrespective of the chosen distribution. Fig. 2g reports the dynamics of the optimal cumulative delay under relatively slow batch arrival rates - 1.5 data batches per second per application on average - for $N = 3$ and $N = 10$, respectively; the algorithm converges to the optimal policy within 300 s and 6000 s, respectively.

Finally, Fig. 2h describes the adaptive capabilities of the stochastic approximation algorithm. The graphic shows the dynamics of the components u_i and of the cumulative processing delay. For this example, the scenario has three applications with $n_1 = 3$, $n_2 = 4$, $n_3 = 5$ and two configurations: Config.1 = $\{\boldsymbol{\lambda}_1 = (3.4, 2.6, 3)$ s^{-1}, $\boldsymbol{\mu}_1 = (103.3, 108.9, 79.6)$ s^{-1} $\}$ and Config.2 = $\{\boldsymbol{\lambda}_2 = (1.4, 2.6, 3)$ s^{-1}, $\boldsymbol{\mu}_2 = (5.33, 10.89, 7.96)$ s^{-1} $\}$, respectively. At 2000 s, the batch arrival rate and the service rate of the first application suddenly drop, thus changing the optimal policy. The algorithm adapts to the configuration change, and converges to the optimal placement for the second configuration. After 4000 s the behaviour of the first application is restored to the initial configuration. Again, the algorithm reaches the optimal policy under Config.1. The learning algorithm proves able to detect sudden changes of the applications configuration in the dark, i.e., without apriori knowledge of the system parameters.

VIII. Conclusions

Fog applications may experience long processing delays on local servers due to processor sharing effects. In order to minimize the processing delay of a set of N fog applications, it is possible to execute the overloaded ones in cloud at a cost. In this context, a key metric is the marginal delay of fog applications measuring their relative performance when executed in cloud or in fog. Algorithms for the optimal placement have been introduced both under perfect information, with a search in the consumed budget, and when the load is unknown at runtime, a primal-dual stochastic approximation procedure can learn the optimal policy in the dark.

This work has only tackled a few aspects of the fog placement problem. First, while the proposed adaptive learning procedure works when stability conditions are met for all fog applications, a valuable objective is to perform in parallel also the online detection of the critically loaded ones, so as, e.g., to proactively enforce their placement in cloud. Furthermore, the proposed framework can be extended to fog clusters accounting for, e.g., standard servers placement policies adopted in cloud computing literature [14], where one needs to split the budget over specific servers, based on their current load.

References

[1] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the Internet of things," in *Proc. of ACM MCC*, Helsinki, Finland, August 13–17 2012.

[2] M. Chiang and T. Zhang, "Fog and IoT: an overview of research opportunities," *IEEE IoT Journal*, vol. 3, no. 6, pp. 854–864, Dec 2016.

[3] G. Li, J. Wu, J. Li, K. Wang, and T. Ye, "Service popularity-based smart resources partitioning for fog computing-enabled industrial Internet of Things," *IEEE Trans. on Industrial Informatics*, vol. 14, no. 10, pp. 1–1, Oct. 2018.

[4] Y. Guan, J. Shao, G. Wei, and M. Xie, "Data security and privacy in fog computing," *IEEE Network*, vol. 32, no. 5, pp. 1–6, 2018.

[5] Amazon.com, Inc., "Amazon aws pricing," 2018.

[6] Y. Gan and C. Delimitrou, "The architectural implications of cloud microservices," *IEEE Computer Arch. Letters*, vol. 17, no. 2, pp. 155–158, July 2018.

[7] J.-P. Arcangeli, R. Boujbel, and S. Leriche, "Automatic deployment of distributed software systems: Definitions and state of the art," *Journal of Systems and Software*, vol. 103, pp. 198–218, 2015.

[8] R. Yu, V. T. Kilari, G. Xue, and D. Yang, "Load balancing for interdependent iot microservices," in *Proc. of IEEE INFOCOM*, Paris, France, 29 April – 2 May 2019.

[9] Y. Niu, F. Liu, and Z. Li, "Load balancing across microservices," in *Proc. of IEEE INFOCOM*, Honolulu, Hawaii, USA, April 15–19 2018.

[10] R. Yu, G. Xue, and X. Zhang, "Application provisioning in fog computing-enabled internet-of-things: a network perspective," in *Proc. of INFOCOM*, Honolulu, Hawaii, USA, April 15–19 2018.

[11] M. Mao and M. Humphrey, "Auto-scaling to minimize cost and meet application deadlines in cloud workflows," in *Proc. of IEEE SC*, Seattle, WA, USA, November 12–18 2011.

[12] P. Jamshidi, A. Ahmad, and C. Pahl, "Cloud migration research: a systematic review," *IEEE Trans. on Cloud Computing*, vol. 1, no. 2, pp. 142–157, 2013.

[13] A. Wolke, M. Bichler, and T. Setzer, "Planning vs. dynamic control: Resource allocation in corporate clouds," *IEEE Trans. on Cloud Computing*, vol. 4, no. 3, pp. 322–335, 2014.

[14] L. Ying, R. Srikant, and X. Kang, "The power of slightly more than one sample in randomized load balancing," *Math. Oper. Res.*, vol. 42, no. 3, pp. 692–722, Aug. 2017.

[15] Y. Guo, A. L. Stolyar, and A. Walid, "Online algorithms for joint application-vm-physical-machine auto-scaling in a cloud," *SIGMETRICS Perform. Eval. Rev.*, vol. 42, no. 1, pp. 589–590, Jun. 2014.

[16] I. Hou, T. Zhao, S. Wang, K. Chan *et al.*, "Asymptotically optimal algorithm for online reconfiguration of edge-clouds," in *Proc. of ACM Mobihoc*, Paderborn, Germany, July 5–8 2016.

[17] R. Urgaonkar, S. Wang, T. He, M. Zafer, K. Chan, and K. K. Leung, "Dynamic service migration and workload scheduling in edge-clouds," *Perform. Eval.*, vol. 91, no. C, pp. 205–228, Sep. 2015.

[18] M. Harchol-Balter, *Performance modeling and design of computer systems: queueing theory in action*. Cambridge University Press, 2013.

[19] E. Altman, K. Avrachenkov, and U. Ayesta, "A survey on discriminatory processor sharing," *Queueing Systems*, vol. 53, no. 1, pp. 53–63, Jun 2006.

[20] E. Altman and N. Shimkin, "Individual Equilibrium and Learning in Processor Sharing Systems," *Operations Research*, vol. 46, no. 6, pp. 776–784, December 1998.

[21] H. J. Kushner and G. G. Yin, *Stochastic Approximation and Recursive Algorithms and Applications*. Springer-Verlag, 2003.

[22] "Extended version: Optimal blind and adaptive fog orchestration under local processor sharing." [Online]. Available: https://drive.google.com/open?id=1RXR3oEduyB8nSyboBdfD32kZvssn3Mcy

[23] S. Boyd and L. Vandenberghe, *Convex Optimisation*. Cambridge University Press, 2004.

[24] A. Nemirovski, "Advances in convex optimization: conic programming," in *Proc. of ICM*, Madrid, Spain, August 22–30 2006.

[25] B. Korte and J. Vygen, *Approximation Algorithms*. Springer, 2012.

[26] H. J. Kushner and D. S. Clark, *Stochastic Approximation Methods for Constrained and Unconstrained Systems*. Springer-Verlag, 1978.

[27] D. Bertsekas, "Convexification procedures and decomposition methods for nonconvex optimisation problems," *Journal of Optimisation Theory and Applications*, vol. 29, no. 2, pp. 169–197, 1979.