Optimal Service Mode Assignment in a Simple Computation Offloading System

Darin Jeff and Eytan Modiano
LIDS, Massachusetts Institute of Technology, Cambridge, MA, USA
Email: {djeff, modiano}@mit.edu

***Abstract*—We consider a simple computation offloading model where jobs can either be fully processed in the cloud or be partially processed at a local server before being sent to the cloud to complete processing. Our goal is to design a policy for assigning jobs to service modes, i.e., full offloading or partial offloading, based on the state of the system, in order to minimize delay in the system. We show that when the cloud server is idle, the optimal policy is to assign the next job in the system queue to the cloud for processing. However, when the cloud server is busy, we show that, under mild assumptions, the optimal policy is of a threshold type, that sends the next job in the system queue to the local server if the queue exceeds a certain threshold. Finally, we demonstrate this policy structure through simulations.**

***Index Terms*—computation offloading, delay-optimal control, dynamic scheduling, switch-type policies, cloud computing.**

I. Introduction

A. Motivation

Modern computational workflows are increasingly resource-intensive and reliant on cloud computing infrastructure. This trend has accelerated significantly with the widespread adoption of Large Language Models (LLMs), such as ChatGPT and Grok, whose memory and compute intensive architectures typically surpass the capabilities of individual user devices. Contemporary systems often rely almost exclusively on cloud-based processing, with little to no local computations. At the same time, processing capacities at intermediate network nodes, such as user devices, edge servers, and intermediate wireless nodes continue to improve. Leveraging these intermediate resources can enable data compression, reducing demands on communication links and enhancing the computational capacity of the system. Such hybrid strategies open opportunities to enhance overall system capacity and reduce end-to-end delay.

Figure 1 illustrates a representative service pipeline commonly encountered in modern offloading workflows. A recurring theme in such pipelines is that increased processing at an upstream stage reduces the workload at subsequent stages. In general, incoming jobs can be processed under multiple service modes, each placing different levels of load on the various stages. When one stage becomes relatively congested, a backpressure-driven intuition suggests that selecting service modes which shift load away from that stage can facilitate implicit load balancing. However, this introduces a fundamental delay trade-off: assigning more processing to an upstream stage can reduce downstream load and delay at this stage, but will increase the load and delay at upstream stages.

This work was supported by the National Science Foundation (NSF) under Grant Nos. CNS-2148183 and CNS-2148128.

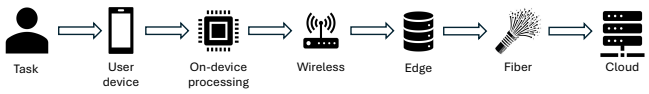


Fig. 1. Illustration of a typical offloading pipeline with multiple computationally capable network nodes.

B. Contributions

We formally analyze this trade-off by modeling a two-stage system with two distinct service modes: (i) pure offloading, where the job is entirely processed at the second stage, and (ii) mixed service, where the job undergoes partial processing at the first stage before being forwarded. The system is modeled as a continuous-time Markov Decision Process (CTMDP) with the objective of minimizing average job delay (i.e. sojourn time). To characterize delay-optimal scheduling policies, we formulate the problem within a dynamic programming framework.

Our objective is to show that the optimal policy admits a simple structural characterization. We show that when the second-stage (cloud) resources are idle, it is always optimal to offload the next job. When the cloud is busy, under mild conditions, the optimal decision follows a *switch-type* structure: assign the job to the mixed-processing mode if queue lengths exceed certain thresholds, and idle otherwise. Our analysis leverages a coupling-based approach to establish this structure, and we verify our results through simulation.

C. Related work

Trade-offs in processing networks have been studied from various perspectives. In [1], the authors investigate optimal estimation under computation-communication trade-offs. The work in [2] considers the co-design problem for computation and communication, where multiple agents transmit updates to a common base station with the goal of minimizing Age of Information (AoI). In the context of service distribution, [3] proposes an extension of the Lyapunov drift-plus-penalty method that achieves throughput optimality while minimizing link-related costs.

Closer to our setting, scheduling in distributed computing networks with both packet transmission and processing requirements is addressed in [4]. This work also focuses on

throughput-optimal dynamic control policies, though delay minimization is not explicitly addressed.

In the limiting regime of our model – where one service mode does all the processing locally and the other service mode offloads all the processing to the cloud, our formulation reduces to the classical heterogeneous server problem initially proposed in [5]. Here, the delay-optimal policy was conjectured to have a threshold structure - a result later proved using policy iteration techniques in [6], and subsequently simplified via coupling arguments in [7]. While threshold-type policies are known to be optimal for two heterogeneous servers, extending this result to larger systems has remained an open problem. Several attempts have been made in this direction [8], [9], but later scrutiny revealed issues in their proofs [10], [11].

Paper Structure. The remainder of the paper is organized as follows. Section II introduces the system model. In Section III, we analyze the optimal policy and establish its structural properties. Section IV presents simulation results that verify our theoretical findings and compare the delay performance of the optimal policy with baseline strategies. Section V concludes with open directions.

II. System Model

A. Operational Regime

We consider an offloading system comprising two sequential servers: a local server with processing capacity μ_0, and a cloud server with capacity $K\mu_0$, where $K > 1$. This setup is illustrated in Figure 2. Jobs arrive according to a Poisson process with rate λ. The system employs two representative service strategies:

- **Service Mode 1 (Offloading):** Jobs are sent directly to the cloud without local processing. The service time at the cloud under this mode is exponential with rate $\mu_{c1} = K\mu_0$.
- **Service Mode 2 (Mixed):** Jobs receive a fraction f of their total service requirement at the local server, and the remainder at the cloud. The service times at the local and cloud servers under this mode are independent exponential random variables with rates $\mu_{l2} = \mu_0/f$ and $\mu_{c2} = K\mu_0/(1-f)$, respectively.

Jobs first enter a centralized *base queue*, as shown in Figure 2, and remain there until assigned a service mode by a *Dynamic Mode Selector* (DMS). The control decisions of mode assignment are taken at *transition instants* namely, upon a job arrival or a service completion at either server. Possible control actions include idling, assigning a job from the base queue to Service Mode 1 (SM1), or assigning it to Service Mode 2 (SM2). The servers are non-idling and maintain separate queues for each service modes. The cloud uses Shortest Expected Processing Time (SEPT) scheduling, prioritizing SM2 over SM1 jobs.

Once assigned a service mode, a job must complete its service under that mode. To avoid premature commitment without compromising optimality, the DMS assigns jobs to SM1 (or SM2) only if no other SM1 (or SM2) jobs are queued at the cloud (or local server, respectively). Consequently, the local SM2 queue and cloud SM1 queue each require only unit storage capacity.

To enable effective load balancing between the servers, the fraction of processing assigned locally in SM2 must satisfy:

$$f > \frac{1}{K+1}$$

Given the server capacities μ_0 and $K\mu_0$, the term $1/(K+1)$ represents the capacity of the local server relative to the total system capacity. Since SM1 jobs are processed entirely at the cloud, this condition enables the system to redistribute the load to the local server by assigning SM2.

To simplify our analysis, we introduce a slightly stronger assumption:

$$f > \frac{1}{K}$$

which is equivalent to requiring that $\mu_{c1} > \mu_{l2}$. Since cloud resources typically dominate ($K \gg 1$), this condition only marginally tightens the requirement for load balancing and facilitates performance analysis without compromising the model's generality.

Remark 1. **(Heterogenous case)** While the system description above focuses on computational tasks, where job processing is split across the two stages in fractions f and $1-f$, our model naturally extends to heterogeneous settings. In particular, the first and second stages may represent different types of operations, such as computation and communication, respectively, as in the on-device – wireless segment of the offloading pipeline shown in Figure 1. In this generalized case, jobs under SM1 are served at rate μ_{c1} at the second stage, while SM2 jobs receive service at rates μ_{l2} and μ_{c2} at the first and second stages, respectively, with $\mu_{c2} > \mu_{c1}$. This system can be cast in our modeling framework with parameters $\mu_0 = \frac{\mu_{l2}(\mu_{c2}-\mu_{c1})}{\mu_{c2}}$, $K = \frac{\mu_{c1}\mu_{c2}}{\mu_{l2}(\mu_{c2}-\mu_{c1})}$ and $f = \frac{\mu_{c2}-\mu_{c1}}{\mu_{c2}}$.

B. State and admissible controls

To facilitate the analysis of this system using a dynamic programming framework, we introduce notation to rigorously describe the system state, and controls. Let $\mathbb{N}_0$ denote the set of non-negative integers. We define the system state as a 4-tuple

$$s \coloneqq (n_0, i_2, i_1, n_2) \in \mathcal{S},$$

with state space $\mathcal{S} = \mathbb{N}_0 \times \{0,1\}^2 \times \mathbb{N}_0$.

The interpretation of each component is summarized in Table I. We use

$$n(s) \coloneqq n_0 + i_2 + i_1 + n_2$$

to denote the total number of jobs in the system in state s.

Given a state s, admissible control actions at the DMS are:

$$\mathcal{U}(s) = \begin{cases} \{\texttt{idle}, \texttt{SM1}, \texttt{SM2}\}, & \text{if } n_0 \geq 1,\ i_2 = i_1 = 0 \\ \{\texttt{idle}, \texttt{SM1}\}, & \text{if } n_0 \geq 1,\ i_2 = 1,\ i_1 = 0 \\ \{\texttt{idle}, \texttt{SM2}\}, & \text{if } n_0 \geq 1,\ i_2 = 0,\ i_1 = 1 \\ \{\texttt{idle}\}, & \text{otherwise} \end{cases}$$

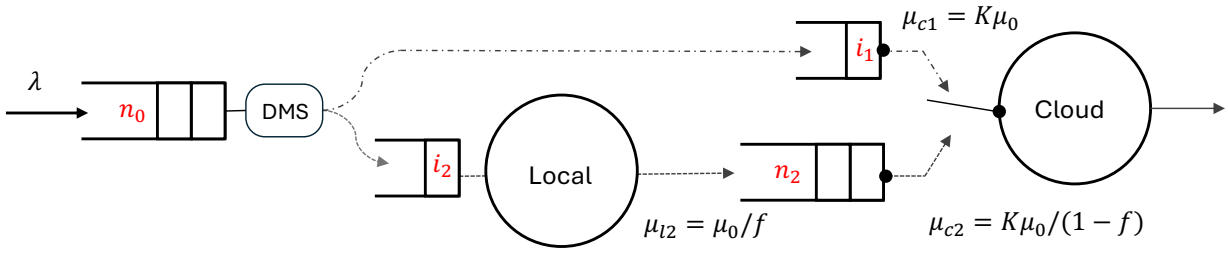


Fig. 2. System model with two service modes: SM1 (pure offloading) and SM2 (mixed processing). Jobs are assigned to modes by the scheduler (DMS) based on system state. The cloud server maintains separate queues and prioritizes SM2 jobs using SEPT scheduling.

TABLE I
INTERPRETATION OF STATE VARIABLES

Symbol	Description
n_0	Number of jobs in the base queue
i_2	Indicator for a local SM2 job in service
i_1	Indicator for a cloud SM1 job (in queue or service)
n_2	Number of SM2 jobs at the cloud

with each control action defined as follows:

`idle`: No service mode assignment is made.

`SM1`: A job from the base queue is assigned to the SM1 cloud queue.

`SM2`: A job from the base queue is assigned to the SM2 local queue.

Note: We use '`SM1`' to refer to the control action selected by the DMS, and plain 'SM1' to denote the corresponding mode or queue.

To concisely describe state transitions due to arrivals, service completions, and service mode assignments, we define state transition and service-mode operators in Table II. These operators map a state $s \in \mathcal{S}$ to another state $s' \in \mathcal{S}$. For example, if a system is in state $s_0 \in \mathcal{S}$ and experiences an arrival, transitioning to state $s_1 = s_0 + (1, 0, 0, 0)$, and subsequently assigns a job using `SM1`, transitioning to $s_2 = s_1 + (-1, 0, 1, 0)$, we can concisely represent this as $s_1 = As_0$ and $s_2 = U_1 As_0$.

C. Dynamic Programming Formulation

To quantitatively compare delay performance under various system policies, we utilize a continuous-time dynamic programming (DP) framework. The DP formulation involves the state space $\mathcal{S}$, the state-dependent action space $\mathcal{U}(s)$, cost rate function $c(s) = n(s)$ and discount factor β.

A *system policy*, π prescribes state-dependent control actions for the Dynamic Mode Selector (DMS). Formally, given the state $s \in \mathcal{S}$, the policy specifies an action:

$$\pi(s) \in \mathcal{U}(s)$$

We focus exclusively on stationary Markovian policies since they are known from standard DP theory (e.g., [12]), to be optimal for minimizing the expected discounted cost.

Note: If a policy wants to assign two jobs from the base queue one to SM1 and one to SM2, it can do this by picking the respective control actions in immediate succession. Without loss of generality, we require control action `SM1` to be **picked first** in such cases.

Given an initial state $s_0 \in \mathcal{S}$ and a policy π, the *discounted cost-to-go* is defined as:

$$J_\beta(s_0; \pi) = \mathbb{E}_\pi \left[\int_0^\infty e^{-\beta t} n(t)\, dt \mid s(0) = s_0 \right]$$

and the corresponding *value function* at state s_0 is:

$$V_\beta(s_0) = \min_\pi J_\beta(s_0; \pi).$$

Finally, a policy π^* is termed *delay-optimal* if it minimizes the expected discounted delay cost among all stabilizing policies, within the stability region, i.e.

$$J_\beta(s_0; \pi^*) = V_\beta(s_0) \quad \text{for all } s_0 \in \mathcal{S}.$$

D. Special Policies

In our main result, we identify specific structural properties of the optimal policy. To formalize these properties, we introduce two special classes of policies.

First, define

$$\mathcal{S}_1 := \{s = (n_0, i_2, i_1, n_2) \in \mathcal{S} \mid n_0 \in \mathbb{N},\ i_1 = n_2 = 0\},$$

representing states where the cloud queues are empty and jobs are available in the base queue. Since the cloud prioritizes SM2 jobs, assigning a job to SM1 in a state in $\mathcal{S} \setminus \mathcal{S}_1$ (where the SM2 cloud queue is non-empty), leads to that job being blocked until all SM2 jobs are processed. Hence an optimal policy doesn't assign SM1 in states outside $\mathcal{S}_1$. Knowing that SM1 is not assigned outside $\mathcal{S}_1$, we now define a policy to be *cloud-first* if it assigns SM1 for all states in $\mathcal{S}_1$.

Definition 1: *Cloud-First Policy*

A policy π is said to be *cloud-first* if

$$\pi(s) = \texttt{SM1} \quad \forall\ s \in \mathcal{S}_1.$$

where,

$$\mathcal{S}_1 := \{s = (n_0, i_2, i_1, n_2) \in \mathcal{S} \mid n_0 \in \mathbb{N},\ i_1 = n_2 = 0\},$$

Identifying an optimal policy as cloud-first fully characterizes its SM1 assignments, reducing the remaining specification

TABLE II
STATE TRANSITION OPERATORS

Operator	Definition	Domain	Description
A	$(n_0, i_2, i_1, n_2) \mapsto (n_0 + 1, i_2, i_1, n_2)$	$\mathcal{S}$	Arrival
D_1	$(n_0, i_2, i_1, n_2) \mapsto (n_0, i_2, (i_1 - 1)^+, n_2)$	$i_1 = 1$	Cloud SM1 departure
D_2	$(n_0, i_2, i_1, n_2) \mapsto (n_0, i_2, i_1, (n_2 - 1)^+)$	$n_2 \in \mathbb{N}$	Cloud SM2 departure
D_L	$(n_0, i_2, i_1, n_2) \mapsto (n_0, (i_2 - 1)^+, i_1, n_2 + i_2)$	$i_2 = 1$	Local SM2 completion
U_1	$(n_0, i_2, i_1, n_2) \mapsto (n_0 - 1, i_2, i_1 + 1, n_2)$	$n_0 \in \mathbb{N},\ i_1 = 0$	Assign to SM1
U_2	$(n_0, i_2, i_1, n_2) \mapsto (n_0 - 1, i_2 + 1, i_1, n_2)$	$n_0 \in \mathbb{N},\ i_2 = 0$	Assign to SM2

to states in $\mathcal{S} \setminus \mathcal{S}_1$, where possible actions are only either assigning SM2 or idling.

We now define a class of policies that have a switching structure in their use of control SM2.

Definition 2: *Switch-Type Policy*

A policy π is said to be *switch-type* if it satisfies the following monotonicity property:

$$\pi(s) = \texttt{SM2} \Rightarrow \pi(s + (x_0, 0, 0, x_2)) = \texttt{SM2}$$

for all $s \in \mathcal{S}$ and $(x_0, x_2) \in \mathbb{N}_0^2$.

This property means that if a policy assigns SM2 in a given state, it also assigns SM2 in states derived by adding jobs to either the base queue or the SM2 queue.

III. ANALYSIS

A. Main Result

We begin by stating the main result of our analysis.

Theorem 1. The delay-optimal policy is *cloud-first*. Furthermore, assuming the *urgency-monotonicity* conjecture, the policy is *switch-type*.

Proof: This result is established through a sequence of supporting theorems which we prove subsequently. First, Theorem 2 shows that the delay-optimal policy is *cloud-first*. Then, Theorem 3 demonstrates that any optimal *cloud-first* policy has a partial switch-type structure. Finally, assuming the *urgency-monotonicity conjecture* (stated below), Theorem 4 concludes that the optimal policy is fully *switch-type*. ■

Before delving into the technical details of the proofs, we provide intuition motivating these structural results.

The *cloud-first* property is motivated by the assumption that the cloud has strictly higher capacity than the local server. As a result, it is never optimal to leave the cloud idle when there are jobs available in the base queue that can be offloaded. This implies the policy should always prioritize offloading to the cloud whenever it is idle.

To gain intuition for the emergence of a switch-type structure, consider the comparative analysis between service modes SM1 and SM2. Under our operational assumptions, we have $\mu_{c1} > \mu_{l2}$, implying that the total expected processing time for SM2 ($1/\mu_{l2} + 1/\mu_{c2}$) is strictly greater than for SM1 ($1/\mu_{c1}$). Based solely on processing times, SM1 appears more efficient.

However, when the cloud is backlogged, starting processing locally under SM2 can reduce the job's subsequent processing time at the cloud instead of having the job waiting in the base queue. Hence, SM2 becomes optimal when the queue sizes cross certain thresholds.

B. Structural Conjecture

Although we do not provide a rigorous proof for this conjecture, it is consistent with structural insights from related dynamic queueing models in the existing literature (e.g., [6], [13]) and supported by simulation evidence in our setting. A similar conjecture is employed in the related open problem of delay minimization in the M/M/3 system with heterogenous servers, where proving explicit threshold structures remains analytically challenging [11].

Without the conjecture, we still establish *cloud-first* behavior and a *partial switch-type* structure in the optimal policy but require it for the full *switch-type* characterization.

Conjecture: *Urgency-Monotonicity*

Let π^* be a delay-optimal policy. Then for all $s \in \mathcal{S}$,

$$\pi^*(As) = \texttt{idle} \Rightarrow \pi^*(s) = \texttt{idle} \tag{1}$$

This conjecture posits that if idling is optimal in a state with more jobs, then reducing the number of jobs in the base queue does not make non-idling preferable. Intuitively, fewer jobs in the base queue reduce urgency, reinforcing the optimality of an idling action. Note that while a switch-type policy requires monotonicity in both base-queue and SM2-cloud directions, this condition only imposes monotonicity for the idle action along the base queue, making it natural and minimal.

C. Cloud-First Behavior at DMS

In this section, we establish that an optimal policy must be *cloud-first*.

Theorem 2. (Cloud-First Behaviour) Let π^* be a delay-optimal policy, then π^* is *cloud-first* (Definition 1).

Proof: This result follows directly from Propositions 2.1 and 2.3, which we state and prove subsequently. ■

Proposition 2.1. Let π^* be a delay-optimal policy. Then:

$$\pi^*(n,0,0,0) = \texttt{SM1}, \quad \forall\, n \in \mathbb{N}.$$

Proof: This follows directly from Lemmas 2.1 and 2.2. ■

Lemma 2.1. Let π^* be a delay-optimal policy. Then:

$$\pi^*(n,0,0,0) \neq \texttt{idle}, \quad \forall\, n \in \mathbb{N}.$$

Proof. Proof deferred to the technical report [14]. □

Lemma 2.2. Let π^* be a delay-optimal policy. Then:

$$\pi^*(n,0,0,0) \neq \texttt{SM2}, \quad \forall\, n \in \mathbb{N}.$$

Proof. Proof deferred to the technical report [14]. □

Proposition 2.1 states that assigning SM1 is optimal whenever the base queue contains jobs while other queues are empty. Doing this transitions the system to a state $(n_0 - 1, 0, 1, 0)$, for some $n_0 \in \mathbb{N}_0$, where the system can now either idle or assign SM2. We now highlight in Proposition 2.2 that the optimal policy at these states exhibits a threshold-based switching structure, shifting from idling to assigning SM2 as n_0 increases.

Proposition 2.2. Let π^* be a delay-optimal policy. Then, there exists some threshold $N_0 \in \mathbb{N}_0$ such that:

$$\pi^*(n_0,0,1,0) = \begin{cases} \texttt{idle} & \text{if } n_0 < N_0 \\ \texttt{SM2} & \text{if } n_0 \geq N_0 \end{cases} \tag{2}$$

Proof: The proof relies on Lemmas 2.3 and 2.5, proven subsequently. Full details are provided in the technical report [14]. ■

Lemma 2.3. Let π^* be a delay-optimal policy. If, for some $m_1, m_2 \in \mathbb{N}_0$ with $m_1 < m_2$:

$$\pi^*(m_1,0,1,0) = \pi^*(m_2,0,1,0) = \texttt{SM2}.$$

then it follows that:

$$\pi^*(n,0,1,0) = \texttt{SM2} \quad \forall\, n \in [\![m_1, m_2]\!]$$

Proof. Proof deferred to the technical report [14]. □

Lemma 2.4. Let π^* be optimal and satisfy

$$\pi^*(n,0,1,0) = \texttt{idle} \quad \forall\, n \geq M_0$$

for some $M_0 \in \mathbb{N}$. Then, the value function satisfies:

$$V_\beta(n_0,0,1,0) - V_\beta(n_0,0,0,1) \geq \epsilon_\beta \quad \forall\, n_0 \geq M_0,$$

where $\epsilon_\beta > 0$ is a constant independent of n_0.

Proof. Proof deferred to the technical report [14]. □

Lemma 2.4 states that if a job assigned SM2 completes local service, the system is in a consistently better state than if that job were still queued up and waiting to be served under SM1. This result helps us establish the inevitability of eventually assigning SM2 under an optimal policy. We formalize this in the following Lemma.

Lemma 2.5. Any policy π that satisfies

$$\pi(n,0,1,0) = \texttt{idle} \quad \forall\, n \geq M_0$$

for some $M_0 \in \mathbb{N}$. Then, policy π is not delay-optimal.

Proof. Proof deferred to the technical report [14]. □

Having established a threshold structure in the optimal assignment of SM2 among states in $\{(n,0,1,0) \mid n \in \mathbb{N}\}$, we now restrict our attention exclusively to policies adhering to this structure. We explicitly define:

$$N_0 \coloneqq \min\{n \in \mathbb{N} \mid \pi^*(n,0,1,0) = \texttt{SM2}\}. \tag{3}$$

Having previously shown that assigning SM1 is optimal at states in $\{(n,0,0,0) \mid n_0 \in \mathbb{N}\}$, it remains to show that SM1 is also optimal at states in $\{(n,0,1,0) \mid n_0 \in \mathbb{N}\}$. We show this in the following Proposition.

Proposition 2.3. Let π^* be a delay-optimal policy, Then,

$$\pi^*(n,1,0,0) = \texttt{SM1}, \quad \forall\, n \in \mathbb{N}.$$

Proof: We first demonstrate in Lemma 2.6 that:

$$\pi^*(n,1,0.0) = \texttt{SM1} \quad \text{for } n \geq N_0 - 1$$

Proposition 2.4 then inductively extends this property downward to all $n \in \mathbb{N}$. ■

Lemma 2.6. Let π^* be a delay-optimal policy. Then

$$\pi^*(n,1,0,0) = \texttt{SM1} \quad \text{for } n \geq N_0.$$

Proof. Let $s = (n+1,0,0,0)$ for some $n \geq N_0$.
From Propositions 2.1 and 2.2, we have:

$$\pi^*(s) = \texttt{SM1} \quad \text{and} \quad \pi^*(U_1 s) = \texttt{SM2}$$

indicating that the optimal policy successively assigns SM1 and then SM2 at state s. Since assigning both SM1 and SM2 is optimal at this higher state, assigning SM1 must also be optimal at the lower state, $U_2 s$. □

Proposition 2.4. Let π^* be a delay-optimal policy. Consider a state $s = (n_0,1,0,0)$ for some $n_0 \in \mathbb{N}$. Then:

$$\pi^*(A^2 s) = \pi^*(As) = \texttt{SM1} \Rightarrow \pi^*(s) = \texttt{SM1}$$

Proof: Let $s = (n_0,1,0,0)$ for some $n_0 \in \mathbb{N}$. At this state, the admissible DMS controls are $\mathcal{U}(s) = \texttt{idle}, \texttt{SM1}$. Thus, it suffices to show that the scenario:

$$\pi^*(s) = \texttt{idle}, \quad \pi^*(As) = \pi^*(A^2 s) = \texttt{SM1}$$

cannot occur under optimality. Suppose for contradiction this scenario holds. Consider the state $D_L s$, where the admissible controls are $\mathcal{U}(D_L s) = \{\texttt{idle}, \texttt{SM2}\}$. We examine two cases:

Case 1: $\pi^*(D_L s) = \texttt{SM2}$

This case is ruled out by Lemma 2.7, proven subsequently.

Case 2: $\pi^*(D_L s) = \texttt{idle}$

This case is ruled out by Lemma 2.8, also proven subsequently. Since both cases lead to contradictions, the original assertion follows. ■

Lemma 2.7. Let $s = (n_0, 1, 0, 0)$ for some $n_0 \in \mathbb{N}$, and suppose there exists a policy π such that:

$$\pi(s) = \texttt{idle}, \quad \pi(As) = \texttt{SM1}, \quad \pi(D_L s) = \texttt{SM2}$$

Then, π is not delay-optimal.

Proof. Proof deferred to the technical report [14]. □

Lemma 2.8. Let $s = (n_0, 1, 0, 0)$ for some $n_0 \in \mathbb{N}$, and suppose there exists a policy π satisfying:

$$\pi(s_0) = \pi(D_L s_0) = \texttt{idle}$$
$$\pi(As_0) = \pi(A^2 s_0) = \texttt{SM1}$$

Then, π is not delay-optimal.

Proof. Proof deferred to the technical report [14]. □

This completes the analysis underpinning Theorem 2, establishing that the optimal policy must be *cloud-first*.

D. Switch-Type Behavior at DMS

Recall that identifying the optimal policy as *cloud-first* completely characterizes SM1 assignments, leaving only the actions of assigning SM2 or idling to be determined at states in $\mathcal{S} \setminus \mathcal{S}_1$. In Theorem 3, we now establish that an optimal, *cloud-first* policy exhibits a partial *switch-type* structure in this region.

Theorem 3 (Partial Switch-Type Structure). Let π^* be a delay-optimal, cloud-first policy. Then there exist non-increasing sequences of integers $\{N_k\}_{k\in\mathbb{N}_0}$ and $\{N'_k\}_{k\in\mathbb{N}}$ defining a family of switching states:

$$\mathcal{S}^{\text{sw}} := \{(N_k, 0, 1, k),\ (N'_k, 0, 0, k) \mid k \in \mathbb{N}\},$$

Such that for all $s \in \mathcal{S}^{\text{sw}}$ and $(x_0, x_2) \in \mathbb{N}_0^2$

$$\pi^*\big(s + (x_0, 0, 0, x_2)\big) = \texttt{SM2}$$

Theorem 3 highlights a near-complete *switch-type* characterization. For each $k \in \mathbb{N}$, the states $(N_k, 0, 0, k)$ and $(N'_k - 1, 0, 1, k)$ serve as effective switching points beyond which assigning SM2 becomes optimal in the increasing base-queue size or SM2 cloud queue size directions. For a complete *switch-type* characterization, it remains to show that idling is optimal in the reverse directions, namely, as we decrease base-queue or SM2 cloud queue size. This final step employs the *urgency-monotonicity* conjecture.

Proof of Theorem 3: The theorem follows directly from Lemma 3.9, establishing the sequences' monotonicity, and Proposition 3.7, showing optimal switching to SM2 in both increasing base-queue and SM2 cloud-queue directions. These results are stated and proved subsequently. ■

Proposition 3.5. Let π be a *cloud-first*, delay-optimal policy. If there exists a state $s \in \mathcal{S}$ such that:

$$\pi(s + (x_0, 0, 0, 0)) = \texttt{SM2} \quad \forall\, x_0 \in \mathbb{N}_0$$

Then:

$$\pi(s + (0, 0, 0, 1)) = \texttt{SM2}$$

Proof: Proof deferred to the technical report [14]. ■

Proposition 3.5 is a key result supporting the construction of the partial switch-type structure.

Proposition 3.6. Let π^* be a delay-optimal policy, and define states:

$$s = (N_0, 0, 1, 0) \text{ and } s' = (N_0 + 1, 0, 0, 1)$$

with N_0 as defined in (3). Then:

(A) $\pi^*(s + (x_0, 0, 0, x_2)) = \texttt{SM2} \quad \forall\, (x_0, x_2) \in \mathbb{N}_0^2$

(B) $\pi^*(s' + (x_0, 0, 0, x_2)) = \texttt{SM2} \quad \forall\, (x_0, x_2) \in \mathbb{N}_0^2$

Proof: Proof deferred to the technical report [14]. ■

We now define the following thresholds for each value of the SM2 cloud queue length, $k \in \mathbb{N}$:

$$N_k := \min\{n \in \mathbb{N} \mid \pi^*(m, 0, 1, k) = \texttt{SM2}\ \ \forall\, m \geq n\}, \quad (4)$$
$$N'_k := \min\{n \in \mathbb{N} \mid \pi^*(m, 0, 0, k) = \texttt{SM2}\ \ \forall\, m \geq n\}. \quad (5)$$

From Proposition 3.6, these thresholds exist and are finite, as they are bounded above by N_0.

Proposition 3.7. Let π^* be delay-optimal. Define:

$$s_k = (N_k, 0, 1, k), \quad s'_k = (N'_k, 0, 0, k).$$

Then, for all $k \in \mathbb{N}$:

(A) $\pi^*(s_k + (x_0, 0, 0, x_2)) = \texttt{SM2} \quad \forall\, (x_0, x_2) \in \mathbb{N}_0^2$

(B) $\pi^*(s'_k + (x_0, 0, 0, x_2)) = \texttt{SM2} \quad \forall\, (x_0, x_2) \in \mathbb{N}_0^2$

Proof: This is a direct consequence of Proposition 3.5. ■

Lemma 3.9. The sequences $\{N_k\}_{k\in\mathbb{N}_0}$ and $\{N'_k\}_{k\in\mathbb{N}}$ defined above are non-increasing.

Proof. Proof deferred to the technical report [14]. □

Theorem 4. (Switch-Type Structure) Let π^* be delay-optimal and cloud-first. Under the *urgency-monotonicity* conjecture, π^* is *switch-type* (Definition 2).

Proof: The result follows from Proposition 3.7 and the *urgency-monotonicity* conjecture. Full details are provided in the technical report [14]. ∎

IV. Simulations

In this section, we empirically validate the structural properties of the delay-optimal policy established in our analysis. We also benchmark its performance against two baseline scheduling policies.

A. Simulation Setup

Uniformization is a classical technique in the theory of continuous-time Markov decision processes (CTMDPs), which transforms a CTMDP into an equivalent discrete-time MDP [15]. We adopt this approach to convert our discounted continuous-time model into a discounted discrete-time formulation.

To compute the value function $V(s)$, we apply the standard value iteration algorithm. At each iteration, the Bellman equation is used to update the cost-to-go for every state. Upon convergence, we obtain the optimal policy by selecting the action that minimizes the updated value function at each state.

Our simulation framework uses the following settings: we impose a buffer limit of $n_{\max} = 300$ on both the base queue and the SM2 cloud queue. The discount factor for the discrete-time system is set to $\alpha = 1 - 10^{-5}$.

We define the system load as $\rho = \lambda/(K+1)\mu_0$, and normalize the capacity of the local server by fixing $\mu_0 = 1$. As a result, the tunable parameters in our simulation are: ρ, which determines system utilization; K, which captures the relative capacity of the cloud; and f, which specifies the fraction of local processing in service mode 2.

B. Cloud-First and Switch-Type structure

We now empirically verify that the optimal policy exhibits both the *cloud-first* and *switch-type* structural properties by examining four system configurations, labeled (a) through (d).

Configuration (a) serves as the baseline, with parameters $\rho = 0.4$, $f = 0.4$, and $K = 8$. In configuration (b), we increase system utilization to $\rho = 0.8$, while keeping $f = 0.4$ and $K = 8$ fixed. Configuration (c) increases the degree of local processing in SM2 by setting $f = 0.8$ (with $\rho = 0.4$ and $K = 8$), and configuration (d) boosts cloud capacity with $K = 15$ (keeping $\rho = 0.4$ and $f = 0.4$).

We visualize the policy using 2D grids that represent slices of the full system state space. Recall that the system state is given by the quadruple (n_0, i_2, i_1, n_2). In each grid, we fix two of these variables and use the remaining two as axes to represent the queue lengths. Each cell in the grid corresponds to a unique state and is annotated with the corresponding action: `0` for `idle`, `1` for `SM1`, and `2` for `SM2`.

In Figure 3, we show the policy restricted to states in $\mathcal{S}_1 = \{(n_0, i_2, i_1, n_2) \in \mathcal{S} \mid i_1 = n_2 = 0\}$. The figure confirms that across all system configurations, the optimal policy always assigns SM1 whenever admissible, verifying the *cloud-first* property established analytically.

In Figure 4, we display the policy over states in $\mathcal{S} \backslash \mathcal{S}_1$. States where no jobs are available in the base queue (i.e. $n_0 = 0$) are excluded, as `idle` is trivially the only admissible action. The plots, shown for the four system configurations (a)-(d), highlight the *switch-type* structure of the SM2 assignments: the optimal policy exhibits a threshold behavior along both the base and cloud queue dimensions. Together, the figures confirm that the optimal policy maintains the *cloud-first* and *switch-type* structures across a range of system conditions.

We observe that the threshold for assigning SM2 increases with higher values of f and K. This aligns with the intuition developed in Section III-A: both larger f and higher K raise the expected total processing time under SM2 (i.e. $1/\mu_{l2} + 1/\mu_{c2}$), making SM1 more favorable and increasing the switching threshold to SM2. Conversely, when system utilization ρ increases, the threshold for assigning SM2 decreases. This is expected, as higher utilization necessitates offloading more jobs via SM2 for load balancing and throughput stability.

C. Delay Performance

We compare the delay performance of the optimal *switch-type* policy against two baseline policies:

(a) *Offload-Only*, and (b) *Non-Idling*.

The *Offload-Only* policy exclusively assigns jobs to SM1, offloading all computation to the cloud while idling the local server. In contrast, the *Non-Idling* policy adheres to the *cloud-first* structure but aggressively utilizes SM2, assigning it whenever it is admissible.

Both baselines are *cloud-first* and *switch-type* but represent opposite extremes in SM2 usage. The *Offload-Only* policy corresponds to the limiting case where switching thresholds are effectively infinite - SM2 is never used. On the other hand, the *Non-Idling* policy represents the case where switching thresholds are zero - SM2 is always used when possible.

It is important to note that the *Offload-Only* policy does not utilize local compute resources and is therefore not stabilizing at high arrival rates (specifically, for $\rho \geq K/(K+1)$). The *Non-Idling* policy, however, remains stabilizing across all arrival rates.

We simulate a system with $f = 0.6$ and $K = 10$ and measure the delay performance across a range of system utilization levels, ρ. The results are shown in Figure 5.

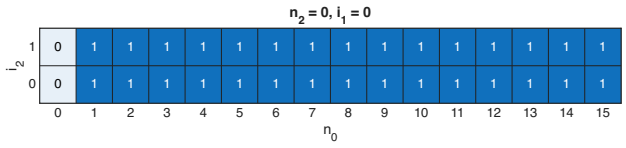
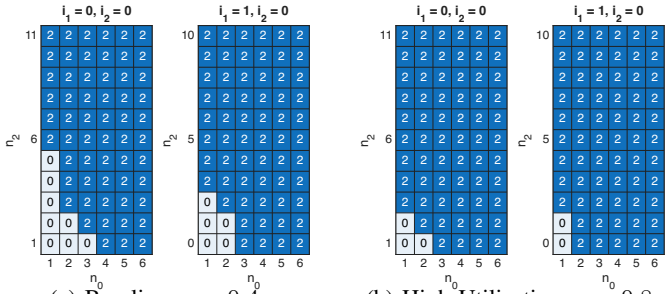
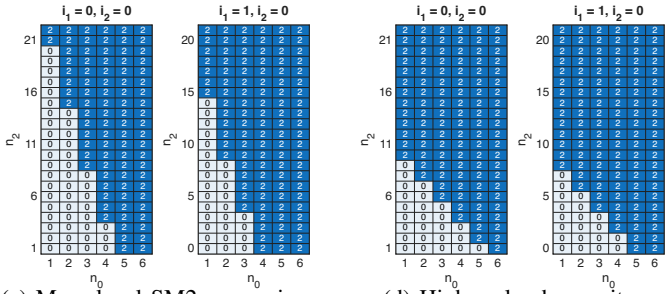


Fig. 3. *Cloud-First* structure: Optimal policy always assigns jobs to SM1 when the cloud is idle ($n_2 = i_1 = 0$), verified consistently across all system configurations (a)–(d).



(a) Baseline: $\rho = 0.4$, $f = 0.4, K = 8$

(b) High Utilization: $\rho = 0.8$, $f = 0.4, K = 8$



(c) More local SM2 processing: $\rho = 0.4, f = 0.8, K = 8$

(d) Higher cloud capacity: $\rho = 0.4, f = 0.4, K = 15$

Fig. 4. *Switch-Type* structure: The optimal policy assigns jobs to SM2 in directions where queue lengths increase, beyond configuration-specific switching points, and idles in the opposite direction. Switching-points vary across configurations (a)–(d)

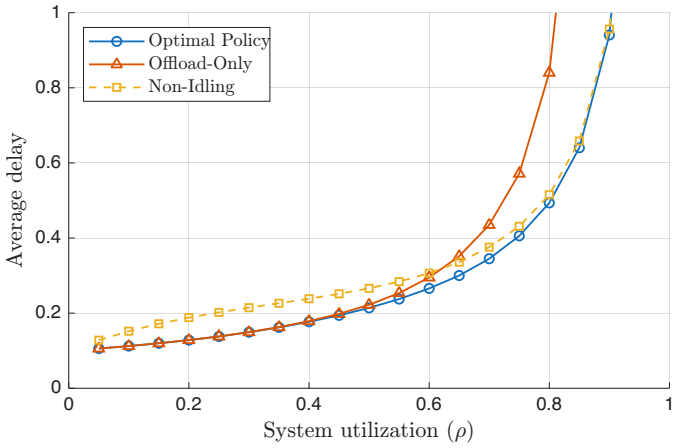


Fig. 5. Average delay under the optimal policy compared to two baseline policies — *Offload-Only* and *Non-Idling*, as a function of system utilization ρ, for a fixed system with $f = 0.6$ and $K = 10$.

As expected, the optimal policy consistently outperforms both baselines across all values of ρ. At low arrival rates, the *Offload-Only* policy performs close to optimal, while at high arrival rates, the *Non-Idling* policy performs comparably. This trend is consistent with our earlier observation: increasing system utilization lowers the optimal switching thresholds for SM2, making its use more favorable at higher utilization levels.

V. Conclusion

We studied the problem of service mode assignment in a two-mode offloading system with the goal of minimizing average delay. In our model, service mode 1 fully offloads jobs to the cloud, while service mode 2 performs a combination of local and cloud processing.

We establish that when the cloud is idle, the optimal action is to immediately offload an available job. When the cloud is busy, we show under mild conditions that the optimal policy exhibits a *switch-type* structure: assign a job to mixed service if the queue lengths exceed a threshold, and idle otherwise.

We verified this structural property through simulation and compared the delay performance of the optimal *switch-type* policy against two baseline *cloud-first* policies.

Future directions include establishing a formal proof of the urgency-monotonicity conjecture and extending the analysis to more general multi-stage, multi-mode offloading systems.

References

[1] L. Ballotta, L. Schenato, and L. Carlone, "Computation-communication trade-offs and sensor selection in real-time estimation for processing networks," *IEEE Transactions on Network Science and Engineering*, vol. 7, no. 4, pp. 2952–2965, 2020.

[2] V. Tripathi, L. Ballotta, L. Carlone, and E. Modiano, "Computation and communication co-design for real-time monitoring and control in multi-agent systems," in *2021 19th International Symposium on Modeling and Optimization in Mobile, Ad hoc, and Wireless Networks (WiOpt)*, 2021, pp. 1–8.

[3] H. Feng, J. Llorca, A. M. Tulino, and A. F. Molisch, "Optimal dynamic cloud network control," *IEEE/ACM Transactions on Networking*, vol. 26, no. 5, pp. 2118–2131, 2018.

[4] J. Zhang, A. Sinha, J. Llorca, A. M. Tulino, and E. Modiano, "Optimal control of distributed computing networks with mixed-cast traffic flows," *IEEE/ACM Transactions on Networking*, vol. 29, no. 4, pp. 1760–1773, 2021.

[5] R. Larsen, "Control of multiple exponential servers with application to computer systems," Doctoral Dissertation, University of Maryland, Jun. 1981.

[6] W. Lin and P. Kumar, "Optimal control of a queueing system with two heterogeneous servers," *IEEE Transactions on Automatic Control*, vol. 29, no. 8, pp. 696–703, 1984.

[7] J. Walrand, "A note on optimal control of a queuing system with two heterogeneous servers," *Systems, Control Letters*, vol. 4, no. 3, pp. 131–134, 1984.

[8] V. Rykov, "Monotone control of queueing systems with heterogeneous servers," *Queueing Systems*, vol. 37, no. 4, pp. 391–403, 2001.

[9] H. P. Luh and I. Viniotis, "Threshold control policies for heterogeneous server systems," *Mathematical Methods of Operations Research*, vol. 55, no. 1, pp. 121–142, 2002.

[10] F. de Véricourt and Y.-P. Zhou, "On the incomplete results for the heterogeneous server problem," *Queueing Systems*, vol. 52, no. 3, pp. 189–191, 2006.

[11] G. Koole, "The slow-server problem with multiple slow servers," *Queueing Systems*, vol. 100, no. 3, pp. 469–471, 2022.

[12] D. P. Bertsekas, *Dynamic Programming and Optimal Control, Volume II*, 4th ed. Athena Scientific, 2012.

[13] B. Hajek, "Optimal control of two interacting service stations," *IEEE Transactions on Automatic Control*, vol. 29, no. 6, pp. 491–499, 1984.

[14] D. Jeff and E. Modiano, "Optimal service mode assignment in a simple computation offloading system: Extended version," 2025. [Online]. Available: https://arxiv.org/abs/2509.18356

[15] S. A. Lippman, "Applying a new device in the optimization of exponential queuing systems," *Operations Research*, vol. 23, no. 4, pp. 687–710, Jul.–aug 1975.