network

MDPI

Article

Exploiting and Mitigating Staleness in Distributed Edge Offloading via Predictive Load Awareness

Sawsan Ali Hamid [1], Yassine Boujelben [2,*] and Faouzi Zarai [2]

[1] NTS'Com Research Laboratory, National School of Electronics and Telecoms of Sfax (ENET'Com), University of Sfax, Sfax 3018, Tunisia; sawsan.altikrity@enetcom.u-sfax.tn
[2] Department of Telecommunications, National School of Electronics and Telecoms of Sfax (ENET'Com), University of Sfax, Sfax 3018, Tunisia; faouzi.zerai@enetcom.usf.tn
* Correspondence: yassine.boujelben@enetcom.usf.tn

Abstract

Distributed offloading systems rely on periodic broadcasts to disseminate server state, yet propagation delays inevitably leave agents operating with stale information at decision time. Staleness is typically viewed as a performance limitation that should be minimized. This work revisits this assumption by studying its role in a distributed asynchronous offloading framework that operates under delayed and potentially stale load information. Players make independent server-selection decisions using locally available information and may optionally compensate for staleness through lightweight load-prediction mechanisms. A central finding is that staleness can act as an implicit coordination mechanism. By inducing heterogeneous and desynchronized perceptions of system state, it naturally diversifies agent decisions and mitigates collective migration oscillations that arise under perfect information sharing. These oscillations are shown to significantly delay convergence and can lead to unstable behavior in lightly loaded regimes. In contrast, stale yet diverse views prevent synchronized reactions and promote faster stabilization. The results further show that the value of prediction increases with communication staleness. As broadcast intervals grow and server-state information becomes increasingly outdated, prediction-based approaches substantially reduce performance degradation, improve load-balancing fairness, and lower migration activity relative to stale-information decisions. The numerical results show that increasing the broadcast interval from 0.5 s to 4 s causes a performance deterioration of approximately 95% for stale-information decisions, whereas prediction-based approaches limit this degradation to less than 15% over the same range. These findings suggest that the objective of distributed offloading should not be to eliminate staleness entirely, but rather to combine its stabilizing effects with lightweight predictive mechanisms that mitigate its negative impact on decision quality.

Keywords: cloud gaming; distributed decision making; edge computing; information age; load management; quality of experience

Check for updates

Academic Editor: Pingyi Fan

Received: 29 June 2026
Revised: 21 August 2026
Accepted: 25 August 2026
Published: 28 August 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

1. Introduction

The proliferation of mobile and Internet-of-Things (IoT) devices has fundamentally reshaped computing architectures. Processing workloads that once resided exclusively in centralized cloud data centers are increasingly offloaded to distributed edge infrastructure deployed closer to end users [1]. Multi-access edge computing (MEC) formalizes this shift: by co-locating computing resources at the network edge, MEC enables low-latency service

delivery without the propagation overhead of remote cloud servers. Yet edge servers are capacity-constrained. Cloud servers offer higher computational power but incur larger propagation delays. Server selection therefore involves a fundamental trade-off between latency and load that must be resolved dynamically and at scale.

Cloud gaming is one of the most demanding applications of this edge-cloud continuum [2]. In this paradigm, game execution, rendering, and processing are performed on remote servers, while the user device acts as a thin client: it sends control inputs upstream and receives a high-definition encoded video stream in return [3,4]. This architecture imposes strict and asymmetric network requirements. Control inputs are delay-sensitive but generate negligible traffic. The video stream, by contrast, demands high throughput and real-time delivery [5]. Any degradation in these conditions translates directly into a deterioration of the Quality of Experience (QoE) perceived by the player [2,6]. Supporting cloud gaming at scale therefore requires powerful server infrastructure, including high-performance CPUs, GPUs, and substantial storage, distributed across both edge and cloud tiers [7].

Within this architecture, the server selection problem is central: each player must be assigned to a computing resource capable of sustaining low latency and sufficient computational capacity. Edge servers minimize propagation delay but are quickly saturated under high player density. Cloud servers absorb demand elastically but reintroduce latency. We capture this trade-off through a weighted cost objective that jointly accounts for delay and server load, as illustrated in Figure 1.

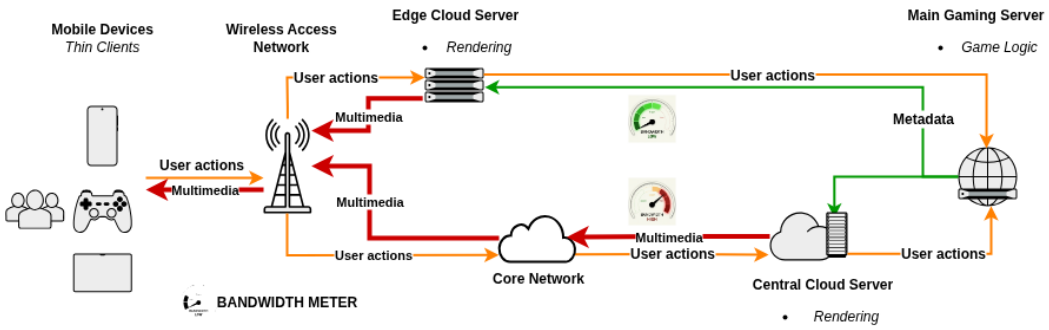


Figure 1. Cloud gaming architecture and offloading trade-off. Edge servers minimize delay but have limited capacity. Cloud servers provide higher capacity at the cost of larger propagation delay.

In large-scale MEC-enabled cloud gaming environments, server selection decisions are made locally by each player agent based on available system-state information. This information is inherently delayed. Server load is communicated through periodic broadcasts that traverse the network before reaching each player. During the interval between two successive broadcasts, the player's view of the system remains unchanged, it is stale. The severity of staleness is governed by the network delay distribution. Real network latency does not follow a uniform distribution; it is right-skewed and exhibits heavy tails arising from congestion, retransmission events, and routing anomalies [8]. Under heavy-tailed delays, a player may hold an outdated load estimate for significantly longer than average. This biases cost evaluation, suppresses beneficial server migrations, and degrades QoE.

Existing offloading frameworks largely assume either instantaneous information exchange or synchronized access to global state information. Consequently, the impact of communication latency, stale information, and concurrent decision making is often neglected despite being inherent to large-scale distributed edge-cloud infrastructures.

Game-theoretic approaches have been proposed for decentralized server selection, notably in cloud gaming environments [9], where resource competition among players

is formulated as a non-cooperative game. Although decision making is distributed among players, existing approaches generally assume that server load information is periodically and synchronously disseminated to all players before each decision round. In addition, strategy updates are typically serialized in order to satisfy the convergence requirements of potential-game dynamics, effectively preventing concurrent player actions. These assumptions neglect communication latency, stale information, and asynchronous decision making, which are inherent characteristics of large-scale edge and cloud infrastructures.

To address this gap, we propose a distributed asynchronous offloading framework that operates under delayed and potentially stale load information, explicitly models propagation latency, and allows players to make independent asynchronous server-selection decisions without centralized optimization or sequential update scheduling. Rather than acting on the last received, and potentially stale, broadcast, each agent maintains a local load predictor that ingests observations and estimates the current load at each candidate server.

Three predictor families are investigated: Exponential Weighted Moving Average (EWM), autoregressive (AR), and scalar Kalman predictors. The broadcast delay channel is modeled through a family of stochastic delay distributions, including LogNormal, Pareto, and Weibull models, allowing systematic analysis of the impact of variability and tail behavior on convergence and QoE.

This paper makes the following contributions.

- We propose the Predictive Distributed Asynchronous Server Selection (PDASS) framework, a distributed asynchronous offloading framework that operates under delayed and stale information. Players maintain local load predictors and base migration decisions on predicted rather than directly observed server states.
- We introduce a stochastic communication-staleness model that captures delayed dissemination of server-state information and the resulting information staleness. The framework supports multiple delay distributions with identical mean propagation delays but different variability characteristics.
- We conduct a comprehensive experimental study across workload regimes, communication conditions, and network scales, showing that the benefit of prediction increases with information staleness. Prediction improves performance, fairness, and migration stability, while EWM provides the most consistent overall trade-off.

The remainder of this paper is organized as follows. Section 2 discusses related work and identifies the research gap. Section 3 describes the system model. Section 3.5 provides a mathematical formulation of the offloading problem. Section 4 describes the proposed framework. Section 5 discusses the main numerical results. Section 6 summarizes the main contributions and outlines future research directions.

2. Related Work

2.1. Quality of Experience and Information Freshness

QoE is a central performance metric for latency-sensitive and interactive cloud and edge services. QoE captures how latency, jitter, packet loss, and system load collectively affect user-perceived performance [10]. Prior studies have shown that QoE degradation can originate from device limitations, network conditions, or server-side congestion [11–13].

The importance of information freshness in edge systems has recently been formalized through the Age-of-Information (AoI) framework, which characterizes the timeliness of state information delivered to decision-making entities. AoI has been studied in the context of mobile edge computing to analyze the trade-off between communication, computation delay, and system performance [14–16]. These works highlight that outdated system

information can significantly degrade decision quality, motivating explicit consideration of staleness in MEC environments.

2.2. Server Selection and Asynchronous Distributed Decisions

Server selection and resource allocation are fundamental problems in cloud and edge computing. In large-scale MEC systems, centralized optimization approaches often become impractical due to scalability limitations and delayed information dissemination [17,18]. As a result, distributed and asynchronous decision-making has received increasing attention.

Among existing decentralized server-selection approaches, the work of Han et al. [9] is most closely related to the present study. However, the two approaches differ fundamentally in their system assumptions. Han et al. assume concurrent migration decision rounds, periodic global dissemination of server-state information, and sequential strategy updates coordinated by a central dispatcher to preserve potential-game convergence. In contrast, PDASS considers asynchronous player decisions, delayed information propagation, stale load estimates, and concurrent updates. The objective of the proposed framework is therefore not to improve the game formulation itself, but to investigate whether predictive mechanisms can compensate for stale information and maintain efficient distributed asynchronous server selection under realistic communication constraints.

Auction-based methods have also been explored for distributed virtual machine placement [19], achieving near-optimal assignments at the cost of high convergence overhead. Theoretical foundations for asynchronous updates with delayed information have been developed in distributed optimization and game-theoretic settings.

Asynchronous best-response and Nash equilibrium seeking under delayed or partial information have been studied in [20,21], demonstrating that stale information can affect convergence behavior and equilibrium properties. These results provide a natural theoretical basis for modeling offloading decisions under delayed system views.

More recently, Aggarwal et al. [22] investigated distributed task offloading in large-scale MEC systems through a mean-field game formulation. Their work addresses scalability by allowing users to compute approximate equilibrium strategies from local information while accounting for shared-resource congestion. Unlike PDASS, however, the focus is on equilibrium computation under large populations rather than mitigating the effects of stale and delayed server-state information.

2.3. Computation Offloading and Predictive Load Estimation

Recent surveys highlight the growing use of game-theoretic, learning-based, and hybrid optimization strategies for computation offloading in MEC environments [23,24]. Existing work has explored optimization-based, game-theoretic, and learning-based offloading approaches [7,23–28], while identifying scalability, adaptability to dynamic network conditions, and distributed decision making as key open challenges. Recent research has therefore increasingly focused on distributed resource allocation, cloud-edge collaboration, and intelligent decision-making mechanisms capable of operating under dynamic conditions.

Lightweight predictive methods have long been used in distributed systems to improve resource allocation under partial observability. In particular, workload prediction techniques based on exponential smoothing and related models offer low computational overhead and robustness to noise [29]. Recent surveys further confirm that prediction-based resource management is a standard tool in cloud environments [30].

Our prior work [31] examined the impact of stale information by comparing idealized instantaneous system views with stale load observations in a distributed edge offloading framework. That work demonstrated that staleness alone can significantly degrade

convergence and system performance. In contrast, the present paper extends this line of research by introducing predictive load awareness under stale information, enabling agents to anticipate congestion rather than react to outdated states.

3. System Model

3.1. Network Topology

We consider a three-layer hierarchical edge–cloud architecture. The network is modeled as a weighted undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w)$, where $\mathcal{V}$ is the set of nodes, $\mathcal{E}$ the set of links, and $w : \mathcal{E} \to \mathbb{R}_{>0}$ the link-weight function encoding propagation cost.

The node set is partitioned into three disjoint layers.

- Player layer: $\mathcal{N} = \{1, \ldots, N\}$ is the set of player agents (mobile devices or thin clients). Each player $i \in \mathcal{N}$ generates a computational task with CPU demand $R_i \in \mathbb{Z}_{>0}$.
- Edge layer: $\mathcal{S}_e \subset \mathcal{S}$ is the set of edge servers. Edge servers are co-located near the players and offer low-latency access at the cost of limited computational capacity κ_e.
- Core layer: $\mathcal{S}_c \subset \mathcal{S}$ is the set of core (cloud) servers. Core servers provide higher computational capacity κ_c but are reached through longer network paths.

The full server set is $\mathcal{S} = \mathcal{S}_c \cup \mathcal{S}_e$, with $\mathcal{S}_c \cap \mathcal{S}_e = \varnothing$. Each server $s \in \mathcal{S}$ has a type $\text{type}(s) \in \{\text{core}, \text{edge}\}$ and a capacity $\kappa_s \triangleq \kappa_{\text{type}(s)}$.

3.2. Assignment and Server Load

At any time instant, each player i is assigned to exactly one server $\sigma(i) \in \mathcal{S}$. The assignment vector $\boldsymbol{\sigma} = (\sigma(1), \ldots, \sigma(N))$ determines how the computational demand is distributed across the server infrastructure. The total load on server s is

$$L_s = \sum_{i \in \mathcal{N}} R_i \cdot \mathbf{1}[\sigma(i) = s], \tag{1}$$

where $\mathbf{1}[\cdot]$ is the indicator function. The normalized load is $\rho_s = L_s/\kappa_s$. A server is overloaded when $\rho_s > 1$.

3.3. QoE and Experience-Loss Cost

3.3.1. Server-Load-Dependent QoE Model

The computational performance delivered by server s to its assigned players depends on the server load. We use the same QoE model commonly adopted in the literature, where this dependence is represented through a sigmoid function of the normalized load ρ_s:

$$g_s(\rho_s) = \frac{\omega_1}{1 + \exp(\omega_2 \rho_s + \omega_3)}, \tag{2}$$

where $\omega_1 > 0$, $\omega_2 > 0$, and $\omega_3 < 0$ are shape parameters.

Sigmoid functions have been extensively used to model saturation effects in resource-constrained systems, where performance remains relatively stable under light and moderate loads and then rapidly deteriorates as resource utilization approaches the capacity limit [9,32]. This behavior reflects the operating characteristics of practical edge and cloud infrastructures. When computational resources are abundant, additional load has only a limited impact on service quality. However, as utilization approaches saturation, queuing delays, resource contention, and scheduling overheads increase nonlinearly, leading to a sharp decline in the quality of service delivered to users. Consequently, the sigmoid function provides a realistic approximation of three distinct operating regions: a high-performance regime under low load, a transition regime near capacity saturation, and a degraded regime under server overload.

The function in (2) decreases monotonically with the normalized load. When $\rho_s \approx 0$, the achieved QoE approaches its maximum value $\omega_1/(1+e^{\omega_3})$. As the load increases toward the server capacity limit, QoE gradually declines and then drops rapidly around the sigmoid inflection region. For heavily overloaded servers ($\rho_s \gg 1$), $g_s(\rho_s)$ approaches zero, reflecting severe service degradation due to resource contention and congestion.

3.3.2. Experience-Loss Cost

Each player i incurs an experience-loss cost that penalizes propagation delay and rewards high server performance. Formally, the cost of player i being assigned to server s is

$$\ell_i(s, \rho_s) \;=\; \lambda_1 \cdot d_i(s) \;-\; \lambda_2 \cdot g_s(\rho_s) \cdot R_i, \tag{3}$$

where $d_i(s)$ is the end-to-end communication delay between player i and server s (defined in Section 3.4), and $\lambda_1, \lambda_2 > 0$ are trade-off weights.

The cost in (3) has a direct QoE interpretation. The first term penalizes latency: assigning player i to a distant server increases ℓ_i. The second term acts as a performance reward: it decreases ℓ_i when $g_s(\rho_s)$ is high, i.e., when the server has sufficient spare capacity to serve player i's demand R_i with low degradation. The weights λ_1 and λ_2 control the relative importance of latency versus load balance.

The system-level objective is to minimize the total experience loss

$$\mathcal{L}(\sigma) \;=\; \sum_{i \in \mathcal{N}} \ell_i\Big(\sigma(i),\, \rho_{\sigma(i)}\Big). \tag{4}$$

3.3.3. Calibration of the QoE–Delay Trade-Off Parameters

The parameters of Equation (3) were calibrated to obtain a meaningful trade-off between communication latency and server-side QoE degradation. The objective is to ensure that player assignments jointly depend on communication conditions and server utilization.

The sigmoid parameters were selected so that the QoE transition occurs near the server saturation point ($\rho_s = 1$). Specifically, we use $\omega_1 = 1$, $\omega_2 = 7$ and $\omega_3 = -7$. The parameter ω_1 determines the maximum QoE level, whereas ω_2 and ω_3 control the steepness and location of the sigmoid transition. The selected values place the inflection point of the sigmoid at $\rho_s = 1$, ensuring that the onset of rapid QoE degradation coincides with the server saturation threshold. With these values, QoE remains nearly constant under moderate loads and decreases rapidly once server utilization approaches capacity.

The relative influence of delay and QoE is controlled through $\lambda_1 = 0.4$ and $\lambda_2 = 0.6$. These weights were selected to keep the contributions of latency and server performance within a comparable numerical range over the operating conditions considered in the simulations. Consequently, communication delay differentiates between servers with similar utilization levels, while server congestion remains an important factor when utilization approaches saturation.

To verify this behavior, the cost function was evaluated for different delay levels while varying the normalized server load from $\rho_s = 0.2$ to $\rho_s = 2$. As shown in Figure 2, the cost increases slowly in the underloaded regime and rises sharply as the server approaches saturation, reflecting the rapid QoE degradation induced by congestion. Beyond $\rho_s = 1$, the performance term becomes increasingly influential, discouraging assignments to heavily loaded servers even when they offer lower communication delays. Because all stochastic delay models are calibrated to preserve the same expected delay for a given player–server pair, the weights λ_1 and λ_2 remain directly comparable across all delay distributions considered in the numerical evaluation.

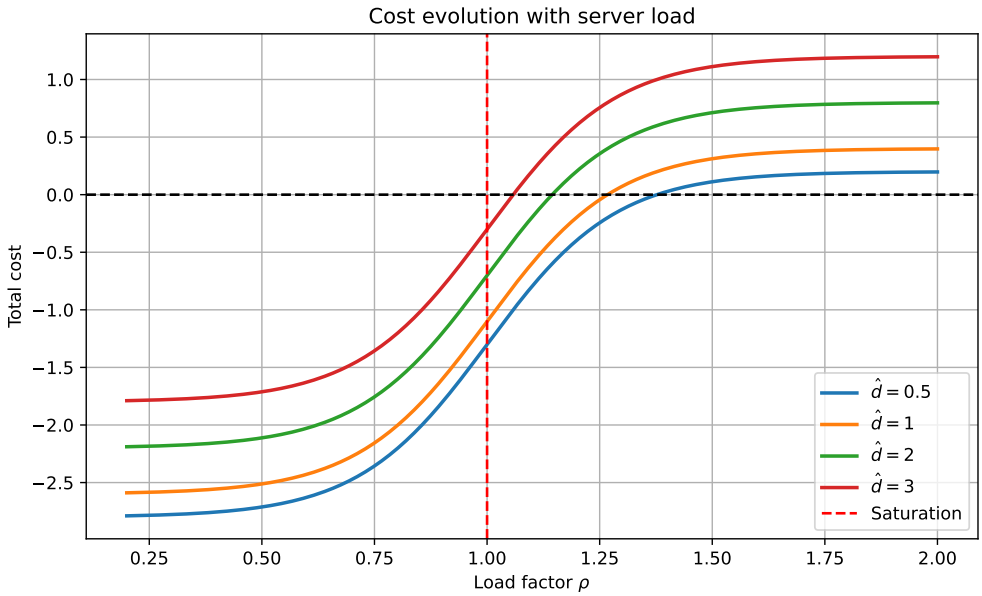


Figure 2. Evolution of the player cost as a function of server load for different normalized delay values. The saturation point ($\rho_s = 1$) corresponds to the center of the QoE transition region. The horizontal dashed line ($\ell_i = 0$) indicates the balance point between the communication-delay and QoE components of the cost function. Above this line, the delay term dominates the cost, whereas below it the QoE term becomes the primary contributor.

This calibration ensures that the proposed experience-loss model captures the intended balance between responsiveness and resource availability while preserving sensitivity to both network and computational conditions.

3.4. Delay Model and Staleness

3.4.1. Motivation

A realistic delay model is central to the analysis of information staleness. A deterministic or uniformly distributed delay would imply bounded and symmetric uncertainty, producing a staleness process that is both predictable and short-lived. Real network latency, however, is right-skewed: most packets arrive close to the minimum propagation time, but occasional congestion events, retransmission bursts, and routing anomalies produce large outliers. This heavy-tailed behavior directly determines how long a player holds a stale load estimate: under a heavy tail, the rare but large delays cause the staleness window to be significantly wider than the median would suggest.

3.4.2. Stochastic Communication Delay Model

Communication latency in practical edge and cloud infrastructures is inherently stochastic because packet transmission is affected by time-varying queuing, contention, routing, processing, and operating-system scheduling delays. Consequently, two packets exchanged between the same player and server may experience different end-to-end delays even though the physical path remains unchanged. Rather than assuming deterministic communication, we explicitly model this randomness to evaluate the robustness of distributed decision making under realistic network uncertainty.

The delay of player i reaching server s is decomposed as

$$d_i(s) \;=\; d_i^{\text{prop}}(s) \cdot Z_{i,s}, \tag{5}$$

where $d_i^{\text{prop}}(s)$ is the deterministic propagation component derived from the network graph,

$$d_i^{\text{prop}}(s) \;=\; \alpha \cdot d_{\text{SP}}(i,s), \tag{6}$$

with $d_{\text{SP}}(i,s)$ the weighted shortest-path distance in $\mathcal{G}$ and $\alpha > 0$ a unit-conversion constant.

The random factor $Z_{i,s}$ captures transient latency fluctuations arising from congestion, queuing, processing, and scheduling effects. A multiplicative representation is adopted because delay variability generally scales with the underlying propagation delay, allowing longer communication paths to exhibit larger fluctuations while preserving strictly positive delays.

The stochastic multiplier is modeled as

$$Z_{i,s} \sim \mathcal{D}(\theta), \tag{7}$$

where $\mathcal{D}$ denotes the selected delay distribution and θ its parameters. The same distribution parameters are shared by all player–server pairs. Consequently, spatial heterogeneity is determined by $d_i^{\text{prop}}(s)$, whereas the choice of distribution controls the magnitude and tail characteristics of latency fluctuations.

For all stochastic delay models considered in this work, the distribution parameters are calibrated such that

$$\mathbb{E}[d_i(s)] = d_i^{\text{prop}}(s). \tag{8}$$

Therefore, the deterministic shortest-path delay remains the expected communication latency for every player–server pair, while the stochastic models differ only in the variability, skewness, and tail behavior of the experienced delays. This calibration enables a fair comparison between delay models without introducing systematic differences in average latency.

3.4.3. From Delay to Staleness

Player agents receive server load information through periodic broadcasts emitted every T_b seconds. Each broadcast packet from server s reaches player i after a delay $d_i(s)$ sampled from (5). Between two successive arrivals, the player's perception of the load on server s remains frozen at the last received value.

Let $t_k = kT_b,\ k \in \mathbb{N}$, denote the broadcast emission times. At time t, player i only has access to the most recent broadcast that has already arrived. The *stale load* observed by player i is therefore

$$\tilde{L}_s^{(i)}(t) \;=\; L_s(t_k), \qquad t_k + d_i^{(k)}(s) \le t < t_{k+1} + d_i^{(k+1)}(s), \tag{9}$$

where $d_i^{(k)}(s)$ denotes the delay associated with the k-th broadcast packet.

Three properties of the staleness process follow directly from the log-normal delay model.

1. Asynchronous across players: Delays $d_i(s)$ differ across players, so no two players observe the same load snapshot simultaneously.
2. Time-varying: Each broadcast sample draws an independent delay, so the staleness window $d_i(s)$ fluctuates over time.
3. Occasionally large: The heavy right tail of the log-normal distribution produces intermittent large delays, during which $\tilde{L}_s^{(i)}(t)$ may deviate substantially from $L_s(t)$.

Players that rely on $\tilde{L}_s^{(i)}(t)$ in place of $L_s(t)$ may therefore evaluate (3) at incorrect load values, leading to suboptimal or suppressed migrations. The load predictor introduced in Section 4 is designed to mitigate this effect.

3.5. Exact Assignment Formulation

Let $x_{i,s} \in \{0,1\}$, for all $i \in \mathcal{N}$ and $s \in \mathcal{S}_i$, denote a binary assignment variable equal to 1 if player i is assigned to server s, and 0 otherwise. The load induced on server s, previously defined in (1), can then be reformulated in terms of the decision variables as

$$L_s = \sum_{i \in \mathcal{N}} R_i x_{i,s}. \tag{10}$$

Using the sigmoid performance model (2), the exact assignment problem is formulated as

$$\min_{x} \quad \sum_{i \in \mathcal{N}} \sum_{s \in \mathcal{S}_i} x_{i,s} \ell_i(s, \rho_s) \tag{11a}$$

$$\text{s.t.} \quad \sum_{s \in \mathcal{S}_i} x_{i,s} = 1, \qquad \forall i \in \mathcal{N}, \tag{11b}$$

$$x_{i,s} \in \{0,1\}, \qquad \forall i \in \mathcal{N},\ s \in \mathcal{S}_i. \tag{11c}$$

Constraint (11b) ensures that each player is assigned to exactly one server. The objective minimizes the total experience loss by jointly accounting for communication delay and server-side performance degradation. Note that no explicit capacity constraint is imposed on the servers. A server may therefore become overloaded, but this results in a degradation of the sigmoid-based QoE term and is consequently penalized directly in the objective function.

Problem (11) is a nonlinear binary constrained optimization problem. The nonlinearity arises from the sigmoid function $g_s(\rho_s)$, whose value depends on the aggregate load induced by all assignment variables through (10). The objective is therefore non-separable across players, since the assignment decision of one player affects the QoE experienced by all players sharing the same server.

Such nonlinear assignment formulations are known to be NP-hard in general [33]. The difficulty is further amplified by the non-separable nature of the objective function: the assignment decision of one player modifies the server load and therefore affects the QoE term experienced by all other players assigned to the same server. As discussed in [34], non-separable nonlinear optimization problems are generally significantly harder than separable formulations due to the coupling induced between decision variables.

4. Predictive Distributed Asynchronous Server Selection

4.1. Framework Overview

We introduce the Predictive Distributed Asynchronous Server Selection (PDASS) framework. PDASS models the server selection process as a stochastic asynchronous dynamical system in which player agents operate independently and periodically search for beneficial server migrations using locally available network information.

Each player maintains a local estimate of server load derived from delayed broadcast updates. Because these updates may be stale due to communication latency, players rely on predictive mechanisms to estimate the current server state before evaluating the experience-loss cost (3). Based on this estimated load state, agents asynchronously decide whether migrating to another server would reduce their expected experience loss.

The framework, illustrated in Figure 3, employs distributed asynchronous decision making. Each player acts autonomously using only its local observations and predicted server states, without a centralized optimizer, assignment controller, or concurrent migration decision phase. Server-load broadcasting and convergence monitoring are supporting services used for state dissemination and convergence detection, and do not participate in the migration-decision process.

We refer to the baseline non-predictive version of the system as the DASS (Distributed Asynchronous Server Selection) [31], where agents directly use the most recently received stale load estimate without applying any prediction mechanism.

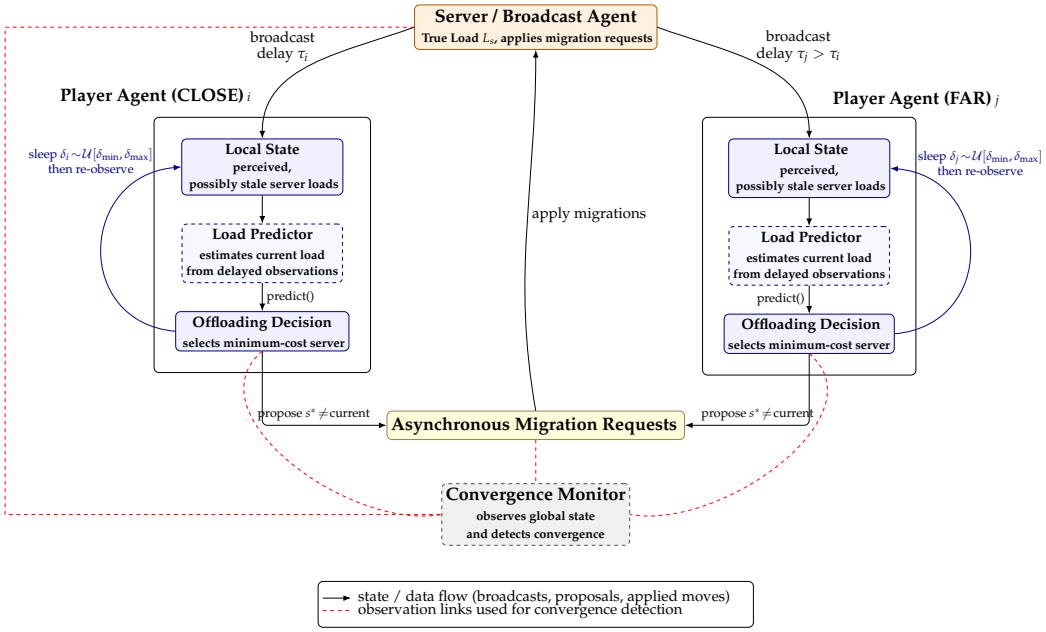


Figure 3. PDASS framework architecture. Players maintain local stale load perceptions and use prediction mechanisms to estimate current server states before evaluating migration opportunities. Server agents disseminate load information and apply migration requests, while a separate convergence monitor observes system evolution. Two player agents are shown for illustration; the framework generalizes to $\mathcal{N}$ asynchronous player agents.

4.2. System Entities

4.2.1. Server Agent

A single server agent maintains the global load vector $\{L_s\}_{s \in \mathcal{S}}$ and broadcasts a snapshot of it to all player agents at regular intervals T_b. The broadcast is synchronous in origin: all players receive the same load snapshot at each emission instant. Each packet, however, traverses the network and arrives at player i after a delay $d_i(s)$ sampled from the log-normal model (5), so arrivals are asynchronous in practice. Consequently, different players may observe different perceptions of the same server state at the same instant, resulting in heterogeneous and potentially stale views of the system load.

4.2.2. Player Agents

Each player agent $i \in \mathcal{N}$ maintains a local state and executes an independent decision loop. Player agents act asynchronously: offloading decisions are taken at uncoordinated times, with inter-decision intervals $\delta_i \sim \mathcal{U}[\delta_{\min}, \delta_{\max}]$. Each agent periodically evaluates candidate servers and selects the destination $s^* \in \mathcal{S}$ that minimizes its experience-loss cost (3), computed using the best available estimate of the server loads.

4.3. Internal Structure of a Player Agent

Each player agent maintains two internal components, depicted in Figure 3:

- A local state;
- A load predictor.

Convergence detection is performed by a separate monitoring service described below.

4.3.1. Local State

The local state of player i comprises its current server assignment $s^*(i)$ and its stale load perception vector $\{\tilde{L}_s^{(i)}\}_{s\in\mathcal{S}}$. The perception $\tilde{L}_s^{(i)}$ is updated upon each broadcast arrival and remains frozen between arrivals, as formalized in (9).

4.3.2. Load Predictor

The load predictor ingests the sequence of stale observations $\tilde{L}_s^{(i)}(t_1), \tilde{L}_s^{(i)}(t_2), \ldots, \tilde{L}_s^{(i)}(t_k)$ received at broadcast arrival instants $t_1 < t_2 < \cdots < t_k$ and produces an estimate of the current load:

$$\hat{L}_s^{(i)}(t) \;=\; \mathcal{P}\Big(\tilde{L}_s^{(i)}(t_1),\; \tilde{L}_s^{(i)}(t_2),\; \ldots,\; \tilde{L}_s^{(i)}(t_k)\Big), \tag{12}$$

where $\mathcal{P}$ denotes the predictor operator. Three instantiations of $\mathcal{P}$ are studied.

EWM Predictor (PDASS-EWM)

The Exponential Weighted Moving (EWM) Average maintains a scalar running estimate $\hat{L}_s^{(i)}$ updated on each new observation $\tilde{L}_s^{(i)}$:

$$\hat{L}_s^{(i)} \;\leftarrow\; \alpha\,\tilde{L}_s^{(i)} \;+\; (1-\alpha)\,\hat{L}_s^{(i)}, \qquad \alpha \in (0,1]. \tag{13}$$

The parameter α controls the memory horizon. A large α favors recent observations; a small α provides greater smoothing.

Kalman Predictor (PDASS-Kalman)

The scalar Kalman filter models server load as a random-walk hidden state:

$$L_s(t) \;=\; L_s(t-1) + w_t, \quad w_t \sim \mathcal{N}(0, Q), \tag{14}$$

$$\tilde{L}_s^{(i)} \;=\; L_s + v_t, \quad v_t \sim \mathcal{N}(0, R_K), \tag{15}$$

where $Q > 0$ is the process noise variance and $R_K > 0$ the observation noise variance. The posterior mean $\hat{L}_s^{(i)}$ and variance P_s are updated via the standard Kalman gain:

$$\begin{aligned} K &= \frac{P_s + Q}{P_s + Q + R_K}, \\ \hat{L}_s^{(i)} &\leftarrow \hat{L}_s^{(i)} + K\big(\tilde{L}_s^{(i)} - \hat{L}_s^{(i)}\big), \\ P_s &\leftarrow (1-K)(P_s + Q). \end{aligned} \tag{16}$$

The posterior variance P_s quantifies estimation uncertainty and is used to widen the migration threshold, as described in Section 4.5.

AR Predictor (PDASS-AR)

The autoregressive (AR) predictor of order p fits a linear model to the last W observations via ordinary least squares:

$$\hat{L}_s^{(i)}(t_k) \;=\; a_0 + \sum_{m=1}^{p} a_m\,\tilde{L}_s^{(i)}(t_{k-m}), \tag{17}$$

where $(a_0, a_1, \ldots, a_p)$ are ordinary least squares (OLS) coefficients estimated on a rolling window of the W most recent observations. The AR predictor captures load trends and oscillations that the EWM and Kalman predictors cannot represent.

4.4. Convergence Monitor

The convergence monitor observes the global state snapshot

$$\Phi(t) \;=\; \big(\{s^*(i)\}_{i\in\mathcal{N}},\; \{L_s\}_{s\in\mathcal{S}}\big) \tag{18}$$

and triggers a `ShutdownEvent` once Φ remains unchanged for τ consecutive broadcast intervals. The monitor passively observes player assignments and server-load states without interacting with the player decision loops. It does not participate in server-selection or migration decisions and therefore has no influence on the distributed dynamics of PDASS.

4.5. Offloading Decision Rule

At each decision epoch, player i computes the predicted experience-loss cost for every candidate server $s \in \mathcal{S}$:

$$\hat{\ell}_i(s,t) \;=\; \lambda_1\, d_i(s) \;-\; \lambda_2\, g_s\Big(\hat{L}_s^{(i)}(t)/\kappa_s\Big) R_i, \tag{19}$$

where $\hat{L}_s^{(i)}(t)$ is obtained from (12) (or equals $\tilde{L}_s^{(i)}(t)$ in the DASS baseline). For migration evaluation, player i considers the counterfactual load state that would result if the migration were executed. Specifically, the player's workload R_i is removed from the current server and added to the candidate destination server before computing the experience-loss cost. The decision therefore reflects the anticipated post-migration server loads rather than the currently predicted assignment.

The agent migrates from its current server s^* to $s' = \arg\min_{s\neq s^*} \hat{\ell}_i(s,t)$ if and only if the improvement exceeds a threshold ε':

$$\hat{\ell}_i(s',t) \;<\; \hat{\ell}_i(s^*,t) - \varepsilon'. \tag{20}$$

For PDASS-Kalman, the threshold is widened in proportion to the posterior uncertainty P_{s^*}:

$$\varepsilon' \;=\; \varepsilon\left(1 + \min\left(\frac{P_{s^*}}{\eta},\, 1\right)\right), \tag{21}$$

where $\eta > 0$ is a scaling constant. This prevents migrations driven by low-confidence load estimates. For PDASS-EWM, PDASS-AR, and DASS, $\varepsilon' = \varepsilon$.

4.6. Formal Description

Algorithms 1–3 jointly provide the complete formal description of PDASS. Algorithm 2 constitutes the core distributed decision-making procedure that solves the player-assignment problem formulated in Section 3. Algorithms 1 and 3 are supporting sub-algorithms that, respectively, provide asynchronous dissemination of load-state information and global convergence detection. Together, the three algorithms form the complete PDASS framework. The novelty of PDASS lies in combining distributed asynchronous server selection with predictive compensation of stale load information through the predictor $\mathcal{P}$, enabling players to make decisions based on anticipated server states rather than outdated observations.

Algorithm 1 Server agent: periodic load broadcast (Algorithm 1 of PDASS)

Require: $\mathcal{N}$, $\mathcal{S}$, broadcast interval T_b, delay model $\mathcal{D}(\sigma)$
Initialize: $L_s \leftarrow 0, \forall s \in \mathcal{S}$
while $\neg$ `ShutdownEvent` **do**
  for all incoming offloading request from i targeting s **do**
    $L_{s_{\text{prev}}} \leftarrow L_{s_{\text{prev}}} - R_i$
    $L_s \leftarrow L_s + R_i$
  end for
  for all $i \in \mathcal{N}$ **do**
    Sample $d_i(s) \sim \mathcal{D}(\sigma)$ {see (5)}
    Schedule delivery of $\{L_s\}_{s\in\mathcal{S}}$ to player i at time $t + d_i(s)$
  end for
  wait T_b
end while

Algorithm 1 describes the server agent. It maintains the global load vector, accepts asynchronous offloading requests, and emits load snapshots every T_b seconds. Delivery is scheduled per player using the stochastic delay model (5), inducing per-player information asymmetry.

Algorithm 2 describes the player-agent decision loop and represents the core optimization mechanism of PDASS. DASS is recovered as the special case where $\mathcal{P}$ is the identity operator, i.e., $\hat{L}_s^{(i)} = \tilde{L}_s^{(i)}$. In contrast, PDASS introduces an explicit prediction stage before each server-selection decision. The predictor $\mathcal{P}$ continuously updates its internal state from received broadcasts and produces estimates of the current server loads through $\mathcal{P}.\textsc{Predict}(\cdot)$. Migration decisions are then computed using the predicted costs $\hat{\ell}_i(s,t)$ rather than the stale observations $\tilde{L}_s^{(i)}$.

For each candidate destination server, the player evaluates a counterfactual post-migration load state by removing its workload from the current server and adding it to the candidate server before computing the corresponding experience-loss cost. This ensures that decisions reflect the anticipated impact of the player's own resource demand on both the source and destination servers.

For PDASS variants, the predictor (12) produces a refined load estimate before each decision, with the Kalman variant additionally widening the migration threshold via (21).

Unlike conventional distributed server-selection schemes that optimize directly using the most recently received load information, PDASS decouples state estimation from decision making through the predictor abstraction $\mathcal{P}$. Consequently, AR, EWM, Kalman, or future prediction models can be incorporated without modifying the migration logic itself. This modular design preserves the distributed asynchronous nature of DASS while enabling decisions based on anticipated server states rather than outdated observations.

Algorithm 3 runs independently. It detects convergence by monitoring the global state snapshot Φ and halts all agents simultaneously via a `ShutdownEvent` once Φ has been stable for τ consecutive seconds.

Algorithm 2 Player agent i: PDASS decision loop (Algorithm 2 of PDASS)

Require: $i, \mathcal{S}, R_i, \mathcal{P} \in \{\text{Identity, AR}(p)\text{, EWM, Kalman}\}, \varepsilon, \eta, K, \delta_{\min}, \delta_{\max}$

Initialize: $s^* \leftarrow s_0$, $\tilde{L}_s^{(i)} \leftarrow 0$, $\forall s \in \mathcal{S}$, $\mathcal{P}$.RESET(), $k \leftarrow 0$
while $\neg$ `ShutdownEvent` **and** $k < K$ **do**
  $k \leftarrow k + 1$
  if new broadcast $\{L_s^{\text{rcv}}\}$ received **then**
    $\tilde{L}_s^{(i)} \leftarrow L_s^{\text{rcv}}$, $\forall s \in \mathcal{S}$
    $\mathcal{P}$.OBSERVE$(\tilde{L}_{s^*}^{(i)})$
  end if
  $\hat{L}_s^{(i)} \leftarrow \mathcal{P}$.PREDICT$(s)$, $\forall s \in \mathcal{S}$
  $P_{s^*} \leftarrow \mathcal{P}$.UNCERTAINTY() $\{P_{s^*} = 0$ for Identity, AR and EWM$\}$
  Compute ε' via (21)
  Compute current cost $\hat{\ell}_i(s^*, t)$
  for all $s \in \mathcal{S}$, $s \neq s^*$ **do**
    $\bar{L}^{(i)} \leftarrow \hat{L}^{(i)}$ {counterfactual load view}
    $\bar{L}_{s^*}^{(i)} \leftarrow \max(0, \bar{L}_{s^*}^{(i)} - R_i)$
    $\bar{L}_s^{(i)} \leftarrow \bar{L}_s^{(i)} + R_i$
    Compute candidate cost $\hat{\ell}_i(s, t)$ using $\bar{L}^{(i)}$
  end for
  $s' \leftarrow \arg\min_{s \neq s^*} \hat{\ell}_i(s, t)$
  if $\hat{\ell}_i(s', t) < \hat{\ell}_i(s^*, t) - \varepsilon'$ **then**
    Send offloading request for s' to server agent
  end if
  Draw $\delta_i \sim \mathcal{U}[\delta_{\min}, \delta_{\max}]$
  Wait δ_i
end while

Algorithm 3 Convergence monitor: global shutdown detection (Algorithm 3 of PDASS)

Require: $\{s^*(i)\}_{i \in \mathcal{N}}, \{L_s\}_{s \in \mathcal{S}}, \tau$

Initialize: $\Phi_{\text{prev}} \leftarrow \varnothing$, $n \leftarrow 0$
while true do
  Wait 1 s
  $\Phi \leftarrow \left(\{s^*(i)\}_{i \in \mathcal{N}}, \{L_s\}_{s \in \mathcal{S}}\right)$ {see (18)}
  if $\Phi = \Phi_{\text{prev}}$ **then**
    $n \leftarrow n + 1$
    if $n \geq \tau$ **then**
      Trigger `ShutdownEvent`
      break
    end if
  else
    $n \leftarrow 0$
  end if
  $\Phi_{\text{prev}} \leftarrow \Phi$
end while

5. Numerical Results

5.1. Global Setup

We evaluate PDASS and its variants against the DASS and Ideal baselines across all combination of load regimes, delay models, and predictor configurations described below.

The evaluation was conducted using a custom discrete-event simulator implemented in Python 3.12.9 and developed specifically for this study. The simulator models asynchronous player decisions, server-state broadcasts, communication delays, migration events, and convergence dynamics through an event-driven execution engine. Communi-

cation staleness arises naturally from delayed state dissemination and is controlled through configurable broadcast intervals and delay distributions.

The simulated environment consists of a three-layer hierarchical network with $|\mathcal{S}_c|$ core servers and $|\mathcal{S}_e|$ edge servers, serving $\mathcal{N}$ autonomous player agents that make asynchronous offloading decisions independently, as illustrated in Figure 4. The considered topology reflects a typical cloud-edge gaming architecture, where players access nearby edge servers that are connected to a backbone of core servers. Since the objective of this work is to study the impact of information staleness, load prediction, and decision dynamics, all experiments are conducted on this representative deployment model.

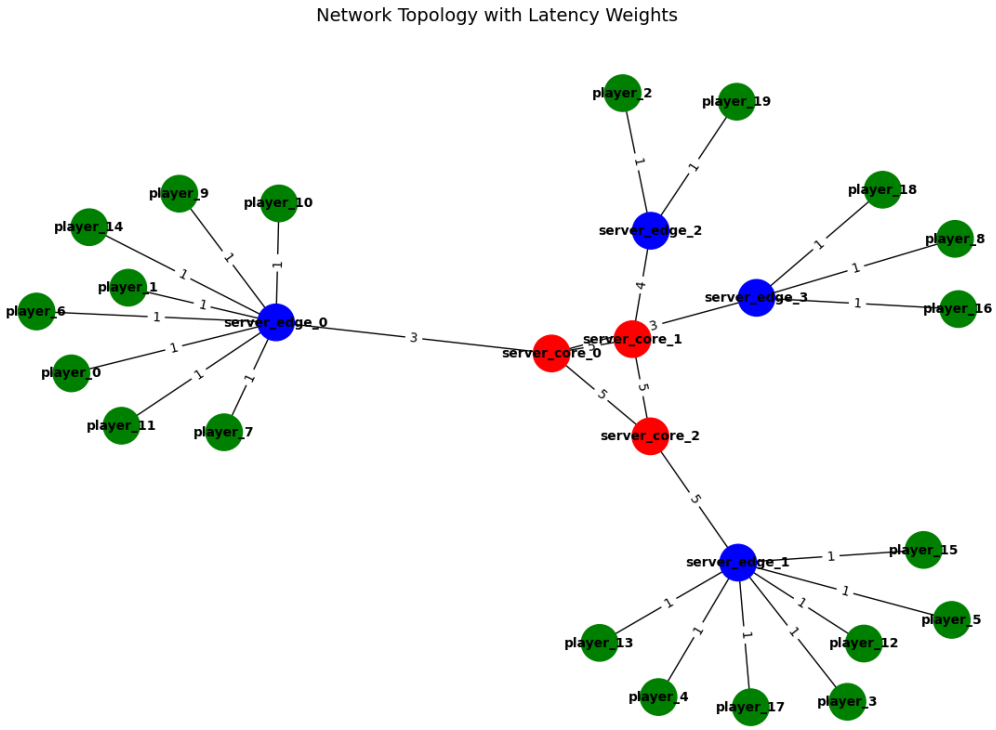


Figure 4. Hierarchical cloud–edge network.

The baseline scenario consists of $\mathcal{N} = 60$ players served by $|\mathcal{S}_c| = 3$ core servers and $|\mathcal{S}_e| = 6$ edge servers. The hierarchical topology was scaled proportionally in order to preserve approximately the same player-to-server ratio as the baseline scenario.

The CPU demand of each player is drawn uniformly at random from $\{1, \ldots, 10\}$ CPU units, yielding a total system demand for the baseline network of approximately $\sum_{i \in \mathcal{N}} R_i = 300$ CPU units. Each experiment is repeated $M = 20$ independent runs with different random seeds to account for stochastic variability.

The cost-function parameters were calibrated as described in Section 3.3.3. Unless otherwise stated, all experiments use $\omega_1 = 1$, $\omega_2 = 7$, $\omega_3 = -7$, $\lambda_1 = 0.4$, and $\lambda_2 = 0.6$.

The key simulation parameters are summarized in Table 1.

Several experiments investigate the sensitivity of the predictors to their hyperparameters. Table 1 therefore reports the set of evaluated parameter values rather than a single operating point.

Table 1. Simulation parameters.

Category	Parameter	Value
Network Baseline	Players N	60
	Core servers $\lvert\mathcal{S}_c\rvert$	3
	Edge servers $\lvert\mathcal{S}_e\rvert$	6
	Player CPU demand R_i	$\mathcal{U}\{1,\ldots,10\}$
	Total CPU demand $\sum R_i$	$\approx$300
Load regimes (κ_c/κ_e)	Severe overload	20/10
	Critical overload	30/15
	Over capacity	40/20
	Saturation	50/25
	Near saturation	60/30
	Moderate load	80/40
	Light load	100/50
Agent timing	Broadcast interval T_b	1 s (default)
	Decision delay $[\delta_{\min}, \delta_{\max}]$	$[0.1, 0.3]$ s
	Max iterations K	500
	Migration threshold ε	0.05
	Stability window τ	10 s
Cost weights	$[\lambda_1, \lambda_2]$	$[0.4, 0.6]$
Sigmoid (g_s)	$[\omega_1, \omega_2, \omega_3]$	$[1, 7, -7]$
Predictor hyperparameter ranges	EWM smoothing α	$\{0.3, 0.6\}$
	Kalman (Q, R_K)	$\{(0.5, 2.0), (2.0, 4.0)\}$
	AR order p/window W	$\{(2, 20), (3, 30)\}$

5.1.1. Load Regimes

Seven capacity configurations are evaluated, ranging from severe overload to light load (Table 1). Core and edge server capacities are coupled at a fixed 2:1 ratio ($\kappa_c = 2\kappa_e$), reflecting the higher provisioning of backbone infrastructure. At saturation load regime, the total available CPU capacity for the baseline network is $= 3 \times 50 + 6 \times 25 = 300$, which corresponds to the total CPU demand.

5.1.2. Communication Delay Models

Five delay configurations are considered to evaluate robustness under different communication conditions:

- LogNormal $\sigma = 0.3$;
- LogNormal $\sigma = 0.6$;
- LogNormal $\sigma = 1.0$;
- Weibull $k = 0.75$;
- Pareto $\alpha = 2.5$.

The three LogNormal settings represent increasing levels of delay variability, whereas Pareto and Weibull capture heavy-tailed and sub-exponential latency behavior, respectively. All models are calibrated such that $\mathbb{E}[d_i(s)] = d_i^{\text{prop}}(s)$. Consequently, differences in performance are attributable to variability and tail behavior rather than changes in average delay.

Together, these distributions provide a controlled comparison framework in which the mean communication latency is held constant while higher-order statistical properties of the delay process vary. This allows the sensitivity of the proposed predictors to latency variability and tail events to be evaluated independently of average network distance. This collection of delay models enables evaluating the proposed predictors under a broad spectrum of latency characteristics, from moderate variability to severe heavy-tailed communication environments.

5.1.3. Evaluation Metrics

The evaluation considers four complementary groups of metrics: system performance, resource allocation quality, convergence behavior, and decision dynamics.

Performance Metrics

The primary objective metric is the global experience loss

$$L(t) = \sum_i \ell_i(s_i(t), \rho_{s_i(t)}),$$

which combines communication delay and QoE degradation caused by server congestion.

The final loss is measured at the end of the simulation:

$$L_{\text{final}} = L(t_{\text{stop}}).$$

Because some scenarios may still exhibit small oscillations close to termination, we also report the steady-state loss, computed as the average loss over the final observation window of length W:

$$L_{\text{ss}} = \frac{1}{W} \sum_{t=t_{\text{stop}}-W+1}^{t_{\text{stop}}} L(t).$$

For experiments involving different network sizes, the normalized loss

$$L_{\text{norm}} = \frac{L}{\sum_i R_i}$$

is additionally reported to enable fair comparisons across workloads.

Resource Allocation Metrics

To quantify the quality of resource utilization, the average overload is reported both over overloaded servers only and over all servers.

Load-balancing fairness is measured through the variance of normalized server utilization:

$$F_{\text{var}} = \text{Var}(\rho_s),$$

where smaller values indicate a more balanced distribution of workload across servers.

Convergence Metrics

The convergence time is measured by the number of broadcast rounds required to reach the termination condition:

$$T_{\text{conv}}.$$

We also report the convergence rate, defined as the percentage of runs that converge before reaching the maximum iteration limit K. This distinguishes genuine convergence from runs that terminate due to the stopping constraint.

To characterize the stability of the final state, we additionally record the number of broadcast rounds elapsed since the last migration event.

Decision Dynamics Metrics

The amount of adaptation activity is measured through the total number of offloading operations:

$$N_{\text{off}} = \sum_{t} M(t),$$

where $M(t)$ denotes the number of migrations occurring during broadcast round t.

The average number of migrations per broadcast round,

$$\frac{N_{\text{off}}}{T_{\text{conv}}},$$

is used as an indicator of collective oscillatory behavior. Lower values correspond to more stable decision dynamics and reduced adaptation overhead.

5.2. Impact of Load Prediction on Staleness

5.2.1. Objective

The objective of this study is to quantify the impact of load prediction on mitigating the performance degradation caused by stale information. In the proposed distributed framework, players make server-selection decisions using server-load information received through delayed broadcasts. Consequently, local decisions are based on potentially outdated views of the system state.

We investigate whether prediction enables players to compensate for this information staleness and approach the performance obtained under perfect information. In addition, we assess whether improved load estimation translates into better convergence behavior and lower experience-loss cost.

5.2.2. Compared Scenarios

Nine scenarios are considered.

The first scenario, denoted Ideal, assumes instantaneous delivery of server-load information. Players therefore observe the current global state before each decision and no information staleness is present.

The second scenario, denoted Stale, uses delayed broadcasts but does not apply any prediction mechanism. Decisions are therefore based solely on the most recently received load snapshot.

The remaining six scenarios augment the stale-information baseline with a local predictor. Three predictor families are investigated:

- Exponential Weighted Moving Average (EWM);
- Autoregressive (AR);
- Kalman filtering.

For each predictor family, two parameter configurations are evaluated, yielding a total of six predictive variants.

Finally, a Sequential Oracle is introduced as a reference benchmark representing fully informed and conflict-free decision making.

All compared methods employ the same asynchronous decision-making process, migration threshold, initialization, and server-selection rule. The only difference among the Ideal, Stale, and prediction-based scenarios lies in the load information available during migration evaluation. The Ideal scenario uses exact current server loads, the Stale scenario

uses the most recently received delayed load information, and the predictive variants use load estimates generated from the same delayed observations.

5.2.3. Sequential Oracle Benchmark

Defining a meaningful reference solution for distributed server selection is non-trivial. An intuitive benchmark is the ideal information scenario, where players observe the exact current server loads. However, perfect information alone does not eliminate the effects of concurrent decision making.

In practice, multiple players may react to the same favorable server conditions within a short time interval, producing concurrent migrations and persistent oscillations. Consequently, the ideal-information scenario does not necessarily converge to a stable allocation despite having access to exact system-state information.

To isolate the effect of information staleness from the effect of concurrent decision conflicts, we introduce a Sequential Oracle. Unlike all distributed scenarios, the oracle allows only one player to update its assignment at a time while providing access to the exact current server loads.

The oracle executes a sequential best-response process. During each pass, players are visited one after another, and every beneficial migration is applied immediately before the next player makes its decision. The process terminates when no player can further reduce its experience-loss cost.

Because different update orders may lead to different equilibria, the oracle employs a multi-start strategy. After convergence, a fraction of players is randomly perturbed and the search is restarted. The lowest-loss equilibrium discovered across all restart trajectories is retained as the oracle solution.

The Sequential Oracle is not intended as a practical implementation. Instead, it provides a strong reference benchmark based on exact load information and sequential best-response updates. While the multi-start procedure attempts to identify high-quality equilibria, the benchmark does not solve a global optimization problem and therefore does not necessarily represent the globally optimal allocation.

5.2.4. Simulation Setup

Unless otherwise stated, the experiments use the baseline network configuration described in Section 5. The study considers $N = 60$ players, six edge servers, and three core servers. The system operates under the saturation load regime, where total demand approximately matches total available computational capacity. The broadcast interval is fixed to $T_b = 1$ s. Communication delays follow a LogNormal distribution with $\sigma = 0.6$, representing the baseline delay condition used throughout this study.

Each scenario is evaluated over 20 independent simulation runs with different random seeds. Results are reported using the performance and convergence metrics introduced previously.

5.2.5. Results and Main Findings

This subsection presents the comparative evaluation of the stale, predictive, ideal, and Sequential Oracle scenarios. The discussion focuses on:

- Final and steady-state experience loss;
- Convergence time and convergence rate;
- Migration activity;
- Load-balancing fairness;
- Performance gap relative to the Sequential Oracle.

The Sequential Oracle Establishes a Stronger Benchmark

The Sequential Oracle consistently achieved the lowest experience loss, with a mean final loss of −73.29, substantially outperforming every concurrent strategy (Table 2). Furthermore, it exhibited near-perfect load balancing, with a fairness variance of only 0.002, compared with values ranging from 0.22 to 0.95 for the predictive and stale-information regimes.

Table 2. Performance comparison using final and steady-state experience loss. Lower values indicate better overall utility.

Method	Final Loss	Steady-State Loss
Sequential Oracle	−73.29	−73.29
EWM ($\alpha = 0.6$)	−36.46	−36.62
Kalman ($Q = 2.0$)	−26.36	−26.52
Stale	−25.37	−25.86
EWM ($\alpha = 0.3$)	−25.00	−24.35
AR ($p = 3$)	−22.09	−22.16
Kalman ($Q = 0.5$)	−19.95	−19.86
AR ($p = 2$)	−10.14	−10.67
Ideal	−69.51	−31.61

These results indicate that a large fraction of the performance degradation observed in distributed execution is not caused by information staleness alone but also by synchronization conflicts arising from concurrent decision-making. The Sequential Oracle therefore provides a meaningful upper-bound benchmark for the performance achievable by selfish agents operating with accurate information.

Perfect Information Does Not Guarantee Convergence

Contrary to intuition, the ideal-information scenario was the only evaluated configuration that failed to converge. As shown in Table 3, the convergence rate of the ideal regime was 0%, whereas all predictor-based approaches, the stale baseline, and the Sequential Oracle achieved a convergence rate of 100%.

The ideal configuration exhibited extreme oscillatory behavior. On average, approximately 96% of the players migrated during each broadcast round (57.85 offloads per round for a 60-player system), resulting in more than 6100 migrations per run. Furthermore, all runs reached the maximum iteration limit before a stable assignment could be identified. These observations confirm that perfect information combined with concurrent best-response updates creates persistent herding effects and prevents convergence.

Table 3. Convergence and migration dynamics under LogNormal delay ($\sigma = 0.6$).

Method	Convergence Rate	Offloads/Round	Total Offloads
Sequential Oracle	100%	3.90	1169.56
Stale	100%	22.80	500.10
AR ($p = 2$)	100%	22.86	504.40
AR ($p = 3$)	100%	14.61	346.35
EWM ($\alpha = 0.3$)	100%	13.01	305.05
EWM ($\alpha = 0.6$)	100%	20.95	473.80
Kalman ($Q = 0.5$)	100%	18.63	448.05
Kalman ($Q = 2.0$)	100%	19.95	465.45
Ideal	0%	57.85	6100.05

These results indicate that the ideal-information scenario is not an appropriate primary benchmark for evaluating the effects of information staleness. Consequently, the Sequential

Oracle provides a more meaningful reference point because it removes decision conflicts while preserving exact load information.

Prediction Primarily Improves Stability

All predictive approaches reduced migration activity relative to the stale baseline. The strongest reductions were observed for EWM ($\alpha = 0.3$) and AR($p = 3$), which lowered total migrations from approximately 500 (stale case) to 305 and 346, respectively (Table 3).

However, lower migration activity did not necessarily imply lower experience loss. Figure 5 shows that several predictors achieved significant reductions in churn while simultaneously producing worse final loss values than the stale baseline. For example, AR ($p = 2$) increased the final loss from -25.37 to -10.14 despite generating a similar number of migrations. Likewise, EWM ($\alpha = 0.3$) reduced migration activity by almost 40% while providing virtually no performance improvement.

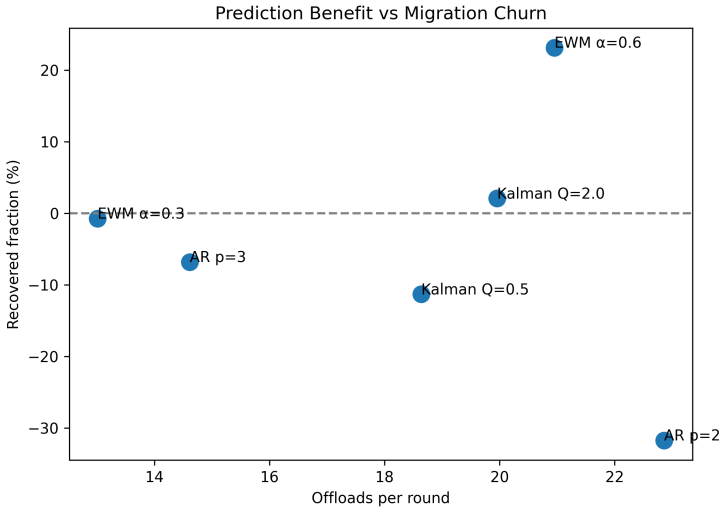


Figure 5. Recovered fraction of the Sequential Oracle performance gap versus migration activity for the evaluated prediction models. Positive values indicate improvement over the stale-information baseline.

These results suggest that prediction primarily affects the stability of the system rather than directly improving the quality of the final equilibrium.

EWM ($\alpha = 0.6$) Provides the Strongest Performance Recovery

The most important result of the study is summarized in Table 4. Relative to the stale baseline, only EWM ($\alpha = 0.6$) achieved a substantial improvement in experience loss, reducing the final loss from -25.37 to -36.46.

Table 4. Fraction of the Sequential Oracle versus Stale performance gap recovered by each prediction strategy. Positive values indicate improvement relative to the stale baseline, while negative values indicate degradation.

Predictor	Recovery (%)
AR ($p = 2$)	-31.76
AR ($p = 3$)	-6.83
EWM ($\alpha = 0.3$)	-0.77
EWM ($\alpha = 0.6$)	23.15
Kalman ($Q = 0.5$)	-11.30
Kalman ($Q = 2.0$)	2.08

Using the Sequential Oracle as reference, EWM ($\alpha = 0.6$) recovered approximately 23% of the performance gap between the stale and oracle regimes. In contrast, Kalman ($Q = 2.0$) recovered only about 2%, while AR ($p = 3$) failed to provide measurable gains. Several predictors actually degraded performance, yielding negative recovery values despite reducing migration activity.

Figure 6 clearly illustrates this ranking. Although EWM ($\alpha = 0.6$) remains far from the Sequential Oracle benchmark, it is the only predictor that consistently moves the system toward the oracle regime while preserving convergence.

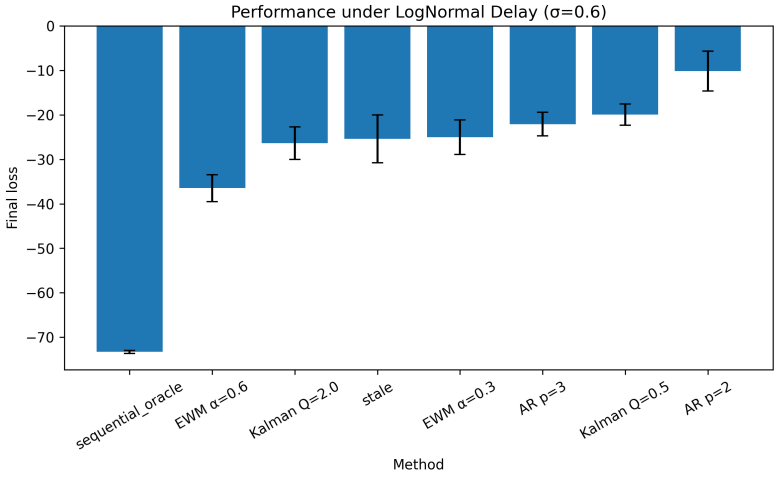


Figure 6. Final experience loss for stale, predictive, ideal, and Sequential Oracle strategies. Error bars denote 95% confidence intervals.

Implications for Predictive Offloading

Overall, the results reveal that prediction affects system dynamics more strongly than equilibrium quality. The largest improvements appear in migration stability, whereas improvements in final utility are limited and highly predictor-dependent. The Sequential Oracle confirms that a substantial performance gap remains, even when prediction is applied. Among the evaluated methods, only EWM ($\alpha = 0.6$) provides a meaningful reduction of that gap.

Load prediction mainly improves stability rather than efficiency; among all tested predictors, only EWM ($\alpha = 0.6$) achieved a meaningful reduction in the performance gap induced by stale information, recovering approximately 23% of the Sequential Oracle benchmark gap while all methods substantially reduced migration churn relative to the unstable ideal-information regime.

5.3. Impact of Broadcast Interval and Communication Delay

5.3.1. Study Utility and Setup

The broadcast interval represents a fundamental trade-off in distributed edge offloading systems. A short broadcast interval provides fresher information about server states but increases communication overhead and may synchronize the decisions of multiple agents. Conversely, a longer broadcast interval reduces communication frequency and introduces staleness, which can potentially prevent undesirable collective oscillations.

This study evaluates the sensitivity of distributed server selection to communication delays and broadcast frequencies. The considered delay models include LogNormal, Pareto, and Weibull distributions with varying parameters, while the broadcast interval controls how frequently server-load information is disseminated to players.

The results show that communication staleness has a pronounced impact on server-selection quality. For the stale-information baseline, performance generally deteriorates as the broadcast interval increases. For example, under the LogNormal delay model with $\sigma = 0.3$, the steady-state loss increases from -37.30 at a broadcast interval of 0.5 s to 0.78 at 2 s and -2.18 at 4 s. Similar trends are observed across all considered delay distributions, indicating that infrequent dissemination of load information can lead to significantly poorer assignment decisions.

5.3.2. Assessment of the Contribution of Prediction to Mitigating Information Staleness

The primary objective of this study is to evaluate whether predictive load estimation can compensate for the degradation caused by stale server-state information. While communication delays and broadcast intervals jointly determine the freshness of the information available to players, increasing the broadcast interval directly amplifies information staleness. Therefore, the key question is not simply whether prediction performs better than stale-information decisions, but whether the benefit of prediction increases as information staleness grows.

To answer this question, we compare the stale-information baseline against the EWM and Kalman predictors under progressively larger broadcast intervals and multiple communication-delay models. The resulting evidence consistently supports the same conclusion: prediction becomes increasingly valuable as information freshness decreases.

Performance Degradation Under Increasing Staleness

Table 5 summarizes the average steady-state loss across all delay models. For the stale baseline, increasing the broadcast interval produces a severe deterioration in performance. In contrast, both predictive approaches remain relatively stable.

Table 5. Average steady-state loss across all communication-delay models. Lower values are better.

Predictor	0.5 s	1 s	2 s	4 s
Stale	−37.42	−27.00	−5.03	−1.82
EWM	−35.59	−33.30	−31.13	−30.49
Kalman	−29.49	−30.40	−29.38	−26.01

Figure 7 makes this trend particularly clear. The figure reports the relative change in steady-state loss with respect to the 0.5 s broadcast interval. The stale-information baseline exhibits a rapid deterioration as information becomes older, with the loss increasing by approximately 28%, 87%, and 95% at broadcast intervals of 1 s, 2 s, and 4 s, respectively. In contrast, EWM experiences only a modest degradation, remaining below 15% across the entire range of broadcast intervals. Kalman performs even more robustly, exhibiting a slight improvement at 1 s and a total degradation of only 12% at 4 s.

This result demonstrates that prediction substantially reduces the sensitivity of distributed server selection to communication staleness. While stale-information decisions become increasingly unreliable as broadcasts become less frequent, predictive approaches maintain nearly constant performance despite operating under the same communication constraints.

This result demonstrates that prediction effectively absorbs the impact of communication-induced staleness and prevents the dramatic collapse observed with stale information.

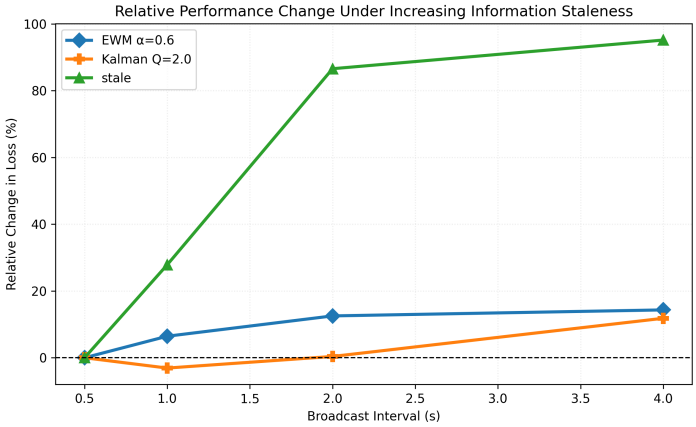


Figure 7. Relative change in steady-state loss with respect to the 0.5 s broadcast interval. Positive values indicate performance degradation, whereas negative values indicate a slight improvement relative to the baseline configuration. The stale-information baseline deteriorates rapidly as communication becomes less frequent, while EWM and Kalman remain comparatively stable, demonstrating that prediction substantially reduces sensitivity to information staleness.

The Value of Prediction Increases with Staleness

While Figure 7 shows how each method degrades with staleness, Figure 8 directly quantifies the benefit of prediction relative to stale-information decisions.

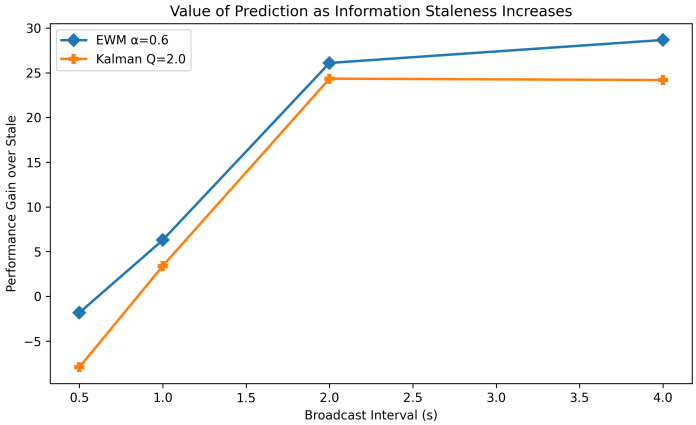


Figure 8. Performance gain of prediction-based decision making relative to the stale-information baseline. The benefit of prediction increases rapidly with the broadcast interval, indicating that predictive load estimation becomes increasingly valuable as server-state information grows older and less reliable.

At short broadcast intervals, stale information remains reasonably accurate and the advantage of prediction is limited. In fact, both predictive approaches perform slightly worse than the stale baseline at 0.5 s, indicating that prediction provides little value when fresh information is already available.

However, the benefit of prediction grows rapidly as communication becomes less frequent. At a broadcast interval of 1 s, EWM and Kalman improve the average steady-state loss by 6.29 and 3.40 units, respectively. At 2 s, the gain increases sharply to more than 24 loss units for both predictors and remains similarly large at 4 s. These results indicate that prediction becomes increasingly valuable precisely when stale information starts to lose its decision-making usefulness.

This observation constitutes the principal finding of the study and provides direct evidence for the central thesis of the paper: predictive load estimation is most beneficial under highly stale communication conditions, where the quality of reactive decisions deteriorates most severely.

Prediction Preserves Fairness Under Stale Information

Performance improvements are accompanied by substantial improvements in load-balancing fairness. Table 6 shows the average fairness variance for all predictors, while Figure 9 reports the relative reduction in fairness variance compared with stale information.

Table 6. Average fairness variance across all communication-delay models. Lower values indicate fairer load distribution.

Predictor	0.5 s	1 s	2 s	4 s
Stale	0.604	0.730	0.981	1.157
EWM	0.572	0.511	0.554	0.876
Kalman	0.462	0.463	0.502	0.955

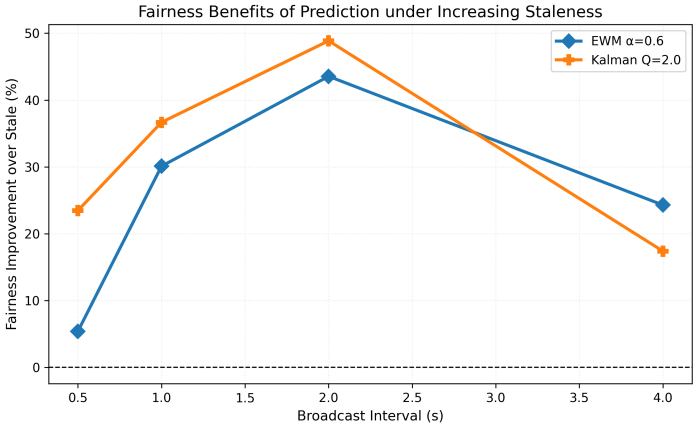


Figure 9. Reduction in fairness variance achieved by predictive load estimation relative to stale-information decisions. The fairness benefit of prediction increases as communication staleness grows, reaching its highest values around a 2 s broadcast interval. These results show that prediction not only improves aggregate performance but also helps maintain a more balanced distribution of workload across edge servers.

For both predictive methods, the fairness benefit grows as the broadcast interval increases from 0.5 s to 2 s. The largest observed reductions exceed 40% of the fairness variance produced by stale-information decisions. This indicates that predictive load estimation not only improves aggregate system performance but also maintains a more balanced distribution of workload across servers.

Interestingly, the fairness improvement follows the same qualitative behavior observed in the performance results. The benefit is modest when information remains fresh but becomes substantial once communication staleness starts dominating the decision process.

Prediction Improves Performance While Reducing Migration Activity

The improvements obtained through predictive load estimation are not achieved by increasing reassignment activity. On the contrary, Table 7 shows that both predictive approaches consistently perform fewer task offloads than the stale-information baseline.

Across all broadcast intervals, Kalman produces the lowest migration counts, followed closely by EWM. For example, at a broadcast interval of 2 s, the average number of offloads decreases from 485.7 for the stale baseline to 464.0 for EWM and 442.6 for Kalman. Similar reductions are observed throughout the studied range of communication conditions.

These results indicate that prediction not only improves assignment quality but also reduces reconfiguration overhead. By anticipating future load conditions, predictive approaches can avoid many of the reactive migrations triggered by outdated server-state

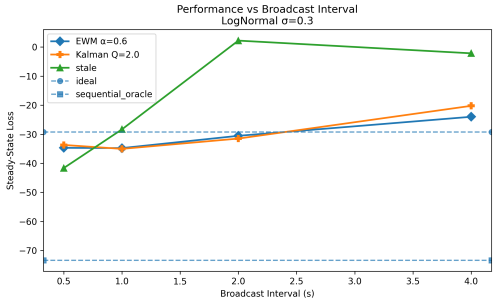
information. Consequently, the performance and fairness gains reported in the previous sections are accompanied by lower migration activity rather than increased system churn.

Table 7. Average number of task offloads across all communication-delay models.

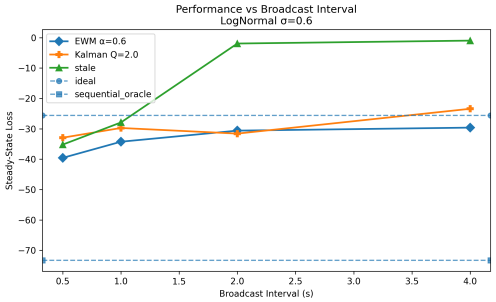
Predictor	0.5 s	1 s	2 s	4 s
Stale	523.0	497.7	485.7	522.2
EWM	507.0	470.5	464.0	500.6
Kalman	502.4	457.1	442.6	477.1

Robustness Across Communication-Delay Distributions

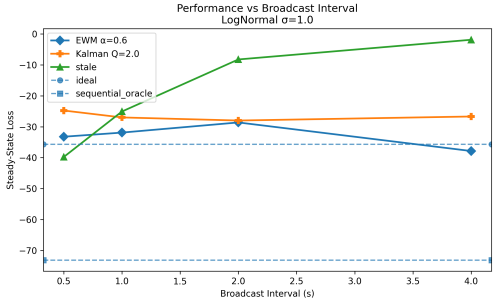
Figure 10 presents representative performance curves for the considered LogNormal, Pareto, and Weibull delay models. Although the exact values differ across distributions, the qualitative behavior remains remarkably consistent.



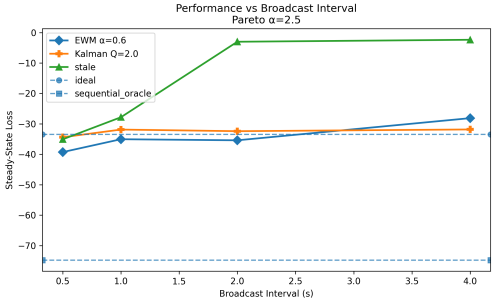
(**a**) LogNormal ($\sigma = 0.3$)



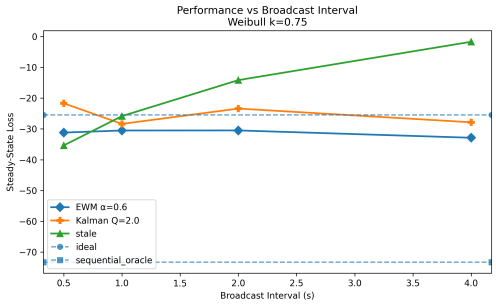
(**b**) LogNormal ($\sigma = 0.6$)



(**c**) LogNormal ($\sigma = 1.0$)



(**d**) Pareto ($\alpha = 2.5$)



(**e**) Weibull ($k = 0.75$)

Figure 10. Steady-state loss as a function of the broadcast interval for the considered communication-delay models. Despite differences in the underlying delay distributions, the same qualitative behavior is observed in all cases: performance of the stale-information baseline deteriorates substantially as broadcasts become less frequent, whereas prediction-based approaches remain considerably more robust. The consistency of this trend demonstrates that the benefits of predictive load estimation are largely independent of the specific communication-delay model.

For all considered communication-delay models, increasing the broadcast interval significantly harms stale-information decisions, whereas prediction-based approaches remain considerably more robust. The persistence of this trend across heterogeneous delay characteristics suggests that the effectiveness of prediction is not tied to a particular communication-delay distribution.

The convergence analysis further confirms that all practical configurations achieve a convergence rate of 100%, indicating that prediction improves decision quality without compromising the stability of the asynchronous distributed decision process.

Key Insight

The results reveal a clear relationship between information staleness and predictor utility. As communication becomes less frequent and server-state information becomes increasingly outdated, the quality of stale-information decisions deteriorates rapidly. Prediction-based approaches substantially reduce this degradation, leading to improved performance, improved fairness, and reduced sensitivity to communication delays.

Most importantly, the benefit of prediction grows with increasing staleness. This observation is consistently supported by all delay models and constitutes the principal finding of this study. The results therefore provide strong evidence for the central thesis of the paper: predictive load estimation is most valuable precisely when communication limitations make fresh system-state information unavailable.

5.4. Impact of Workload Regimes

5.4.1. Experimental Objectives and Setup

This experiment evaluates the sensitivity of the proposed load-balancing mechanism to workload intensity. While the previous study focused on the impact of stale information and predictive load estimation under a fixed operating point, the objective here is to determine whether the effectiveness of prediction varies across the different workload regimes introduced in Section 5.1.1.

To isolate the effect of workload, all simulations use a fixed broadcast interval of 1 s and a LogNormal communication-delay model with $\sigma = 0.6$. Based on the findings of the predictor-screening study, only the most representative configurations are retained: the stale-information baseline, EWM ($\alpha = 0.6$), Kalman ($Q = 2.0$), the ideal-information reference, and the Sequential Oracle benchmark.

The analysis focuses on three complementary aspects: (i) the evolution of experience loss across load regimes, (ii) the distance from the Sequential Oracle benchmark, and (iii) the fraction of the stale-to-oracle performance gap recovered by predictive load estimation. Together, these metrics provide insight into when prediction is beneficial and whether its primary impact is on equilibrium quality, system stability, or both.

5.4.2. Performance Across Load Regimes

The experimental results show that the effectiveness of predictive load estimation varies significantly across workload regimes, affecting both equilibrium quality and migration dynamics.

Operating Regions for Prediction

The effectiveness of prediction varies significantly with the workload regime. As shown in Table 8 and Figure 11, both predictive approaches provide little benefit under light workloads and become ineffective under extreme overload.

Under Light Load, EWM ($\alpha = 0.6$) and Kalman ($Q = 2.0$) actually degrade performance relative to the stale baseline, yielding recovered fractions of -13.0% and -85.5%, respectively. In contrast, prediction becomes useful in the neighborhood of saturation.

EWM ($\alpha = 0.6$) recovers approximately 19.1% of the stale-to-oracle gap under the Saturation regime, while both predictors provide measurable gains under Near Saturation.

The results therefore indicate that prediction is most valuable when the system operates close to its capacity limits but has not yet entered a resource-starved state. When resources are abundant, prediction is unnecessary; when resources are severely constrained, prediction cannot compensate for the lack of available capacity.

Table 8. Recovered fraction of the Sequential Oracle versus Stale performance gap using steady-state loss. Positive values indicate improvement over the stale-information baseline.

Load Regime	EWM ($\alpha = 0.6$)	Kalman ($Q = 2.0$)
Light Load	−12.99%	−85.50%
Moderate Load	8.78%	3.46%
Near Saturation	8.26%	12.14%
Saturation	19.15%	0.39%
Over Capacity	0.87%	−2.32%
Critical Overload	−81.32%	−136.75%
Severe Overload	−48.95%	−54.63%

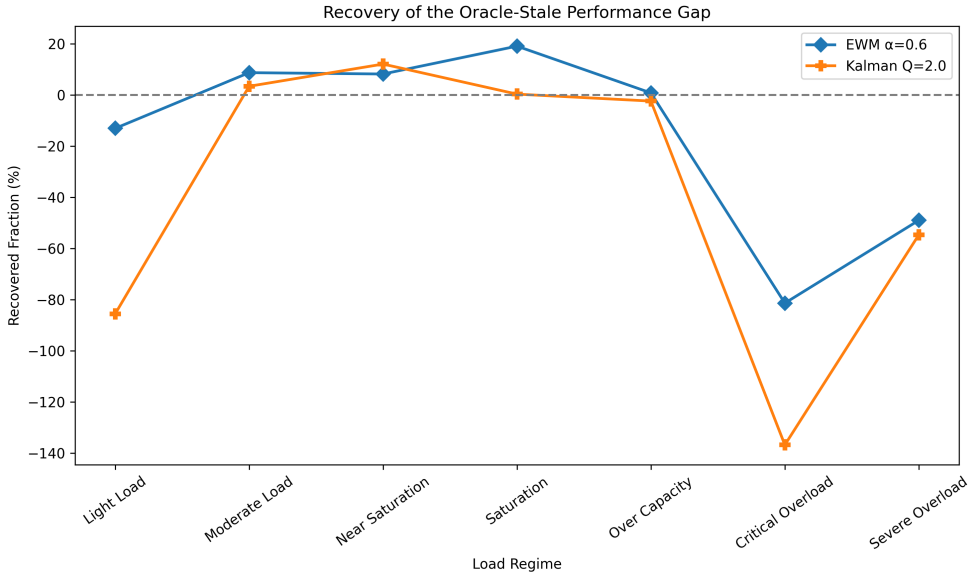


Figure 11. Recovery of the stale-to-oracle performance gap. Prediction is most effective near saturation and provides little or no benefit under extreme overload.

Migration Stability Versus Equilibrium Quality

Table 9 shows that both predictive approaches reduce migration activity across all load regimes. This effect is most pronounced under Light and Moderate Loads. For example, under Light Load, total migrations decrease from approximately 428 for the stale baseline to 262 for EWM ($\alpha = 0.6$) and only 191 for Kalman ($Q = 2.0$).

However, the reduction in migration activity does not always translate into improved utility. Figure 12 shows several workload regimes where predictive methods produce fewer migrations while simultaneously yielding worse experience loss than the stale baseline. This discrepancy is particularly visible for Kalman ($Q = 2.0$), which consistently suppresses player movement yet rarely improves equilibrium quality.

These observations confirm that prediction primarily acts as a stabilization mechanism by damping migration oscillations and reducing churn. Improvements in utility require not only fewer migrations but also sufficiently accurate forecasts that steer players toward better equilibria.

Table 9. Convergence behavior and migration activity across workload regimes.

Load Regime	Predictor	Convergence Rate	Offloads per Round	Total Offloads
Light Load	Stale	100%	19.89	427.75
	EWM ($\alpha = 0.6$)	100%	12.58	262.15
	Kalman ($Q = 2.0$)	100%	9.60	191.30
Moderate Load	Stale	100%	21.27	466.80
	EWM ($\alpha = 0.6$)	100%	17.55	382.10
	Kalman ($Q = 2.0$)	100%	14.22	313.45
Near Saturation	Stale	100%	22.47	499.15
	EWM ($\alpha = 0.6$)	100%	19.88	453.20
	Kalman ($Q = 2.0$)	100%	18.46	422.95
Saturation	Stale	100%	22.93	498.20
	EWM ($\alpha = 0.6$)	100%	21.96	472.95
	Kalman ($Q = 2.0$)	100%	20.13	461.75
Over Capacity	Stale	100%	22.53	490.55
	EWM ($\alpha = 0.6$)	100%	22.05	488.90
	Kalman ($Q = 2.0$)	100%	21.02	474.90
Critical Overload	Stale	100%	22.70	502.80
	EWM ($\alpha = 0.6$)	100%	22.18	490.90
	Kalman ($Q = 2.0$)	100%	21.43	484.45
Severe Overload	Stale	100%	23.59	501.95
	EWM ($\alpha = 0.6$)	100%	23.67	496.45
	Kalman ($Q = 2.0$)	100%	22.28	481.05

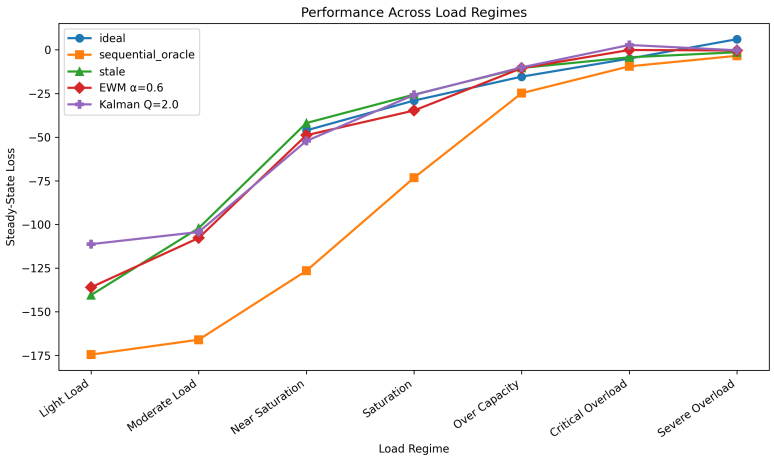


Figure 12. Steady-state experience loss across workload regimes for the stale-information baseline, EWM ($\alpha = 0.6$), Kalman ($Q = 2.0$), and the Sequential Oracle benchmark.

Consistent Gains from EWM ($\alpha = 0.6$)

Across all workload conditions, EWM ($\alpha = 0.6$) delivers the most consistent prediction gains and generally achieves the best trade-off between equilibrium quality and stability.

Near Saturation, both predictors improve upon the stale baseline. EWM ($\alpha = 0.6$) reduces the steady-state loss from -41.93 to -48.92, while Kalman ($Q = 2.0$) achieves an even lower loss of -52.21. Under Saturation, EWM ($\alpha = 0.6$) delivers the strongest observed improvement, reducing the steady-state loss from -25.64 to -34.77 and recovering approximately 19.1% of the stale-to-oracle gap. In contrast, Kalman ($Q = 2.0$) recovers less than 1% of the same gap.

The corresponding oracle-gap results in Figure 13 provide additional insight. Although Kalman occasionally achieves competitive steady-state losses, EWM ($\alpha = 0.6$) consistently remains closer to the Sequential Oracle in the near saturation and saturation regimes. This indicates that EWM more reliably converts stale information into higher-quality equilibrium outcomes while maintaining stable system behavior.

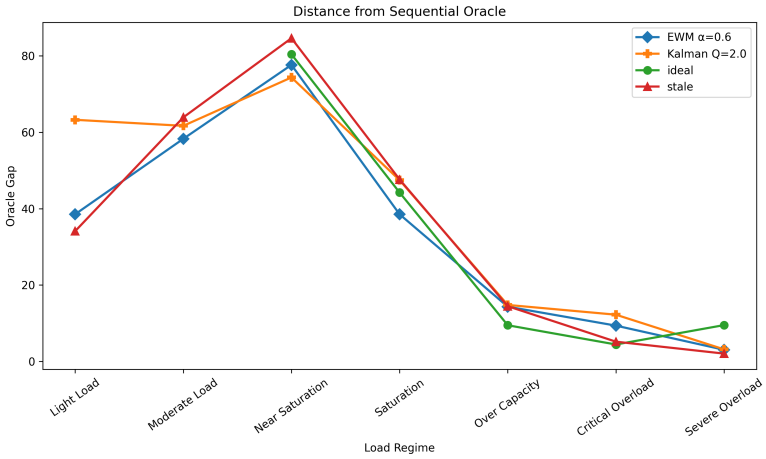


Figure 13. Oracle gap across workload regimes. The metric measures the difference between the achieved steady-state loss and the Sequential Oracle loss.

Oracle Performance and Instability Under Perfect Information

The Sequential Oracle continues to provide the best performance across all workload conditions. As shown in Table 10 and Figure 13, every concurrent strategy remains substantially above the oracle benchmark, especially near saturation where the oracle advantage is greatest.

Table 10. Steady-state experience loss across workload regimes. Lower values indicate better performance.

Load Regime	Sequential Oracle	Stale	EWM ($\alpha = 0.6$)	Kalman ($Q = 2.0$)
Light Load	−174.57	−140.46	−136.03	−111.30
Moderate Load	−166.04	−102.13	−107.75	−104.35
Near Saturation	−126.57	−41.93	−48.92	−52.21
Saturation	−73.29	−25.64	−34.77	−25.83
Over Capacity	−24.83	−10.38	−10.51	−10.04
Critical Overload	−9.46	−4.29	−0.09	2.78
Severe Overload	−3.42	−1.37	−0.37	−0.26

For example, under Saturation, the stale baseline exhibits an oracle gap of approximately 47.6, whereas EWM ($\alpha = 0.6$) reduces that gap only to 38.5. Although prediction improves performance, a large fraction of the gap remains unexplained by information staleness alone.

At the same time, the ideal-information scenario again fails to converge in any workload regime. The convergence rate remains 0%, while migration counts exceed 5500–6100 moves per run and approximately 50–58 players migrate during each broadcast round. This result confirms that perfect information combined with concurrent best-response updates induces persistent herding effects and prevents convergence.

Together, these findings indicate that synchronization effects and concurrent decision-making remain the dominant sources of inefficiency, even when accurate future load estimates are available.

Operating Regions for Prediction

Overall, the load-regime study reveals a clear operating region in which prediction is useful. Predictive models provide little benefit under light workloads, become most effective near saturation, and lose effectiveness again once the system enters severe overload. Prediction consistently reduces migration activity across all regimes, but these stability improvements do not necessarily translate into improved utility. Among the evaluated methods, EWM ($\alpha = 0.6$) is the only predictor that repeatedly improves equilibrium quality and meaningfully reduces the stale-to-oracle performance gap. Nevertheless, a substantial gap to the Sequential Oracle remains, suggesting that concurrent decision conflicts constitute a more important source of inefficiency than information staleness itself.

Load prediction mainly improves stability rather than efficiency; among the evaluated models, EWM ($\alpha = 0.6$) is the only predictor that consistently improves equilibrium quality, with its largest benefit occurring near saturation where it recovers approximately 19% of the stale-to-oracle performance gap while significantly reducing migration churn.

5.5. Scalability Analysis

5.5.1. Objectives and Setup

This study evaluates the scalability of the proposed framework as the number of players increases. The experiments consider network sizes ranging from 20 to 100 players while maintaining the Saturation load regime, a broadcast interval of 1 s, and a Log-Normal communication-delay model with $\sigma = 0.6$. Following the predictor-screening study, the analysis focuses on the stale-information baseline, EWM ($\alpha = 0.6$), Kalman ($Q = 2.0$), the ideal-information reference, and the Sequential Oracle benchmark. The objective is to determine whether the benefits of predictive load estimation persist as the system grows and to quantify their impact on equilibrium quality, migration activity, and convergence behavior.

5.5.2. Performance Across Network Sizes

The results reveal distinct scalability characteristics among the considered approaches. While predictive methods continue to improve upon the stale-information baseline at larger scales, the ideal-information scenario remains unable to achieve stable convergence despite attaining near-oracle performance.

Prediction Remains Effective as the System Scales

From 60 players onward, predictive load estimation consistently improves equilibrium quality relative to the stale-information baseline. EWM ($\alpha = 0.6$) reduces the oracle gap across all large-scale configurations and recovers up to 22.97% of the stale-to-oracle performance gap at 80 players (Table 11, Figure 14). These results indicate that the benefits of prediction persist as the number of players grows.

Table 11. Recovered fraction of the Sequential Oracle versus Stale performance gap across network sizes. Positive values indicate improvement over the stale-information baseline.

Players	EWM ($\alpha = 0.6$)	Kalman ($Q = 2.0$)
20	0.57%	−0.18%
40	164.86%	283.06%
60	10.74%	−2.71%
80	22.97%	14.22%
100	7.66%	−7.73%

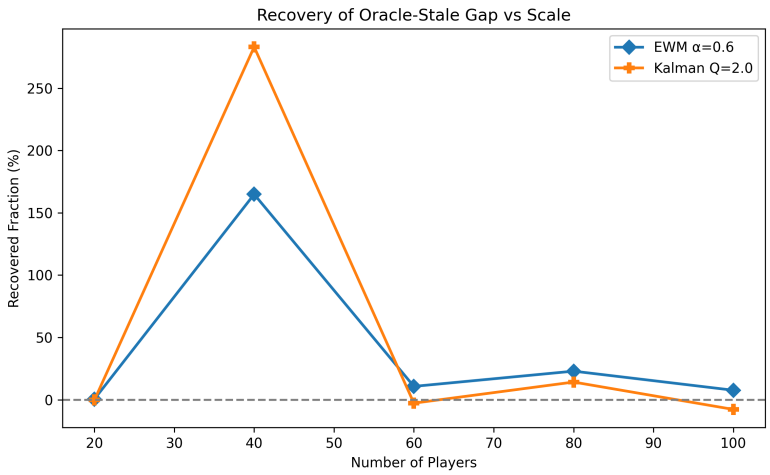


Figure 14. Recovered fraction of the stale-to-oracle performance gap across network sizes. Positive values indicate improvement over the stale-information baseline.

Scalability Trade-Offs Between Predictors

As network size increases, EWM ($\alpha = 0.6$) delivers the strongest utility improvements among the distributed approaches. While the differences between predictors remain small for lightly loaded instances, EWM increasingly outperforms both the stale-information baseline and Kalman ($Q = 2.0$) as the number of players grows, as shown in Table 12. At 80 and 100 players, EWM achieves steady-state losses of -96.91 and -176.88, compared with -86.03 and -159.80 for Kalman, respectively.

Kalman further suppresses migration activity but generally achieves smaller utility gains, suggesting a more conservative decision policy. The widening gap between EWM and Kalman for large networks indicates that prediction accuracy becomes increasingly important as system congestion and decision interactions intensify.

Table 12. Steady-state experience loss across network sizes. Lower values indicate better performance.

Players	Sequential Oracle	Ideal	Stale	EWM ($\alpha = 0.6$)	Kalman ($Q = 2.0$)
20	-2.19	1.63	0.80	0.78	0.81
40	-7.65	-7.24	-9.42	-6.50	-4.40
60	-73.29	-35.77	-26.59	-31.61	-25.32
80	-192.77	-90.10	-68.33	-96.91	-86.03
100	-279.36	-250.77	-168.38	-176.88	-159.80

Migration Activity at Scale

Migration activity increases with network size for all distributed methods, but the increase is significantly slower when prediction is used. At 100 players, EWM and Kalman reduce total offloads by approximately 27% and 37%, respectively, compared with the stale baseline (Table 13, Figure 15). Prediction therefore improves not only equilibrium quality but also operational stability.

Instability Under Perfect Information

Despite achieving near-oracle utility, the ideal-information scenario fails to converge for all network sizes examined. Its convergence rate remains 0% while migration activity grows almost linearly with the number of players, reaching 9 566.95 offloads at 100 players (Table 13). In contrast, the Sequential Oracle achieves both stable convergence and the best overall performance. These results reinforce the conclusion that accurate information

alone does not guarantee convergence; the dominant source of instability is the presence of concurrent decision conflicts rather than information staleness.

Table 13. Convergence behavior and migration activity as a function of network size.

Players	Predictor	Convergence Rate	Offloads per Round	Total Offloads
20	Ideal	0%	19.25	2029.35
	Stale	100%	5.81	60.00
	EWM ($\alpha = 0.6$)	100%	5.97	60.00
	Kalman ($Q = 2.0$)	100%	5.97	60.00
40	Ideal	0%	38.18	4066.70
	Stale	100%	14.33	238.30
	EWM ($\alpha = 0.6$)	100%	14.05	233.95
	Kalman ($Q = 2.0$)	100%	13.78	231.25
60	Ideal	0%	57.36	6101.30
	Stale	100%	23.10	501.15
	EWM ($\alpha = 0.6$)	100%	21.46	477.60
	Kalman ($Q = 2.0$)	100%	20.19	460.50
80	Ideal	0%	76.64	8129.40
	Stale	100%	26.81	750.45
	EWM ($\alpha = 0.6$)	100%	23.00	646.25
	Kalman ($Q = 2.0$)	100%	19.69	547.70
100	Ideal	0%	86.46	9566.95
	Stale	100%	29.10	952.10
	EWM ($\alpha = 0.6$)	100%	21.68	692.85
	Kalman ($Q = 2.0$)	100%	19.25	601.95

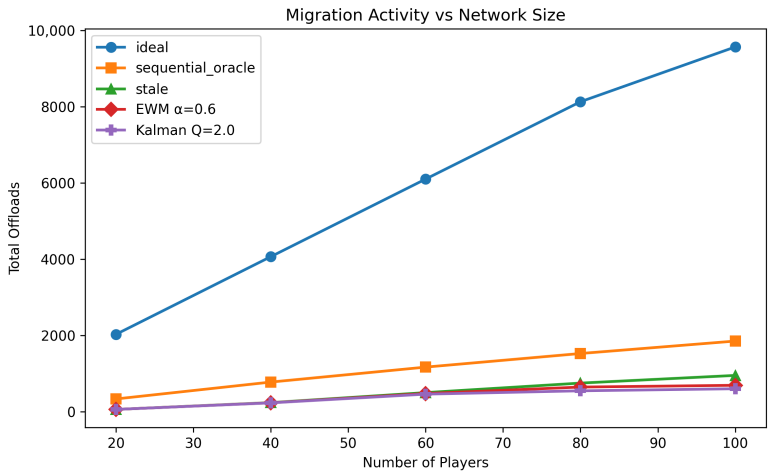


Figure 15. Migration activity versus network size. Predictive methods substantially reduce migration churn relative to both the stale-information baseline and the ideal-information reference.

Behavior at Scale

The advantages of predictive load estimation persist as the system scales. Both predictive approaches consistently outperform the stale-information baseline while reducing migration activity, with EWM ($\alpha = 0.6$) providing the best overall trade-off between utility and stability. Although prediction mitigates part of the performance degradation caused by stale information, a substantial gap to the Sequential Oracle remains at large scales. Moreover, the ideal-information scenario continues to exhibit severe instability despite achieving near-oracle utility, indicating that concurrent migration decisions based on fresh

load information, rather than information staleness alone, remain the primary limiting factor in large-scale systems.

6. Conclusions

This work studied the impact of information staleness on distributed offloading under delayed state dissemination, comparing reactive and predictive strategies across multiple workload regimes, communication conditions, and network scales. The results show that staleness should not be viewed solely as a degradation factor. While stale information reduces decision quality, it also acts as an implicit stabilizer by desynchronizing agent actions and mitigating the collective migration oscillations that emerge under perfectly synchronized information.

A central finding of this work is that the utility of prediction increases with information staleness. As communication becomes less frequent and server-state information becomes increasingly outdated, the quality of reactive stale-information decisions degrades rapidly. Prediction-based approaches substantially reduce this degradation, preserve fairness, and lower migration activity, with the largest benefits consistently observed under the most challenging staleness conditions.

Among the evaluated methods, EWM ($\alpha = 0.6$) provides the most consistent improvements across workload regimes, communication-delay models, and network scales. Although prediction substantially narrows the performance gap caused by stale information, a significant gap to the Sequential Oracle remains, suggesting that concurrent reactions to the same load information have a greater impact on performance than information staleness itself.

The present study focuses on a representative hierarchical cloud–edge deployment and a specific workload-generation model. Although the results are consistent across a broad range of system loads, delay models, broadcast intervals, and network sizes, further evaluation under alternative network topologies and more diverse workload dynamics would help assess the generality of the observed behavior. Exploring these scenarios constitutes an important direction for future research.

Overall, the findings support a reframing of staleness as a potential design lever for distributed offloading systems. Rather than eliminating staleness entirely, practical systems may benefit from combining moderate information freshness with lightweight predictive mechanisms that compensate for outdated observations while preserving the stabilizing effects of asynchronous decision making.

Author Contributions: Methodology: S.A.H. and Y.B.; formal analysis and investigation: S.A.H. and Y.B.; writing—original draft preparation: S.A.H. and Y.B.; writing—review and editing: S.A.H., Y.B. and F.Z.; supervision: Y.B. and F.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Xia, X.; Chen, F.; He, Q.; Cui, G.; Grundy, J.C.; Abdelrazek, M.; Xu, X.; Jin, H. Data, user and power allocations for caching in multi-access edge computing. *IEEE Trans. Parallel Distrib. Syst.* **2021**, *33*, 1144–1155. [CrossRef]
2. Baena, C.; Fortes, S.; Peñaherrera-Pulla, O.S.; Baena, E.; Barco, R. Gaming in the cloud: 5G as the pillar for future gaming approaches. *IEEE Commun. Mag.* **2024**, *62*, 76–82. [CrossRef]
3. Di Domenico, A.; Perna, G.; Trevisan, M.; Vassio, L.; Giordano, D. A network analysis on cloud gaming: Stadia, geforce now and psnow. *Network* **2021**, *1*, 247–260. [CrossRef]

4. Peñaherrera-Pulla, O.S.; Baena, C.; Fortes, S.; Baena, E.; Barco, R. Measuring key quality indicators in cloud gaming: Framework and assessment over wireless networks. *Sensors* **2021**, *21*, 1387. [CrossRef] [PubMed]
5. Monaco, D.; Sacco, A.; Spina, D.; Strada, F.; Bottino, A.; Cerquitelli, T.; Marchetto, G. Real-time latency prediction for cloud gaming applications. *Comput. Netw.* **2025**, *264*, 111235. [CrossRef]
6. Jumani, A.K.; Shi, J.; Laghari, A.A.; Amin, M.A.; Nabi, A.U.; Narwani, K.; Zhang, Y. Quality of experience (qoe) in cloud gaming: A comparative analysis of deep learning techniques via facial emotions in a virtual reality environment. *Sensors* **2025**, *25*, 1594. [CrossRef] [PubMed]
7. Zhang, X.; Chen, H.; Zhao, Y.; Ma, Z.; Xu, Y.; Huang, H.; Yin, H.; Wu, D.O. Improving cloud gaming experience through mobile edge computing. *IEEE Wirel. Commun.* **2019**, *26*, 178–183. [CrossRef]
8. Graff, P.; Marchal, X.; Cholez, T.; Tuffin, S.; Mathieu, B.; Festor, O. An analysis of cloud gaming platforms behavior under different network constraints. In *Proceedings of the 2021 17th International Conference on Network and Service Management (CNSM)*; IEEE: Izmir, Turkey, 2021; pp. 551–557.
9. Han, Y.; Guo, D.; Cai, W.; Wang, X.; Leung, V.C. Virtual machine placement optimization in mobile cloud gaming through QoE-oriented resource competition. *IEEE Trans. Cloud Comput.* **2020**, *10*, 2204–2218. [CrossRef]
10. Hamid, S.A.; Boujelben, Y.; Zarai, F. Enhancing cloud gaming experience through optimized virtual machine placement: A comprehensive review. *J. Netw. Syst. Manag.* **2024**, *32*, 80. [CrossRef]
11. Jumani, A.K.; Shi, J.; Laghari, A.A.; Nabi, A.U. A proposed model for security of quality of experience data in the cloud gaming environment. *Int. J. Electron. Secur. Digit. Forensics* **2025**, *17*, 327–350. [CrossRef]
12. Laghari, A.A.; Zhang, X.; Shaikh, Z.A.; Khan, A.; Estrela, V.V.; Izadi, S. A review on quality of experience (QoE) in cloud computing. *J. Reliab. Intell. Environ.* **2024**, *10*, 107–121. [CrossRef]
13. Amiri, M.; Osman, H.A.; Shirmohammadi, S.; Abdallah, M. Toward delay-efficient game-aware data centers for cloud gaming. *ACM Trans. Multimed. Comput. Commun. Appl. (TOMM)* **2016**, *12*, 1–19. [CrossRef]
14. Jiang, Y.; Liu, J.; Humar, I.; Chen, M.; Alqahtani, S.A.; Hossain, M.S. Age-of-Information-Based Computation Offloading and Transmission Scheduling in Mobile Edge Computing. *IEEE Internet Things J.* **2023**, *10*, 19138–19150. [CrossRef]
15. He, X.; You, C.; Quek, T.Q. Age-based scheduling for mobile edge computing: A deep reinforcement learning approach. *IEEE Trans. Mob. Comput.* **2024**, *23*, 9881–9897. [CrossRef]
16. Yang, N.; Liu, Y.; Chen, S.; Zhang, M.; Zhang, H. Minimizing AoI in Mobile Edge Computing: Nested Index Policy with Preemptive and Non-preemptive Structure. *IEEE Trans. Mob. Comput.* **2026**, *25*, 12641–12652. [CrossRef]
17. Dinaki, H.E.; Shirmohammadi, S.; Hashemi, M.R. Boosted metaheuristic algorithms for QoE-aware server selection in multiplayer cloud gaming. *IEEE Access* **2020**, *8*, 60468–60483. [CrossRef]
18. Deng, Y.; Li, Y.; Seet, R.; Tang, X.; Cai, W. The server allocation problem for session-based multiplayer cloud gaming. *IEEE Trans. Multimed.* **2017**, *20*, 1233–1245. [CrossRef]
19. Boujelben, Y.; Fourati, H. A distributed auction-based algorithm for virtual machine placement in multiplayer cloud gaming infrastructures. *Int. J. Cloud Comput.* **2024**, *13*, 80–98. [CrossRef]
20. Lei, J.; Shanbhag, U.V. Asynchronous Schemes for Stochastic and Misspecified Potential Games. *Oper. Res.* **2020**, *68*, 1742–1766. [CrossRef]
21. Yi, P.; Pavel, L. Asynchronous Distributed Algorithms for Seeking Generalized Nash Equilibria under Partial Information. *IEEE Trans. Cybern.* **2020**, *50*, 2514–2526. [CrossRef] [PubMed]
22. Aggarwal, S.; uz Zaman, M.A.; Bastopcu, M.; Ulukus, S.; Başar, T. Distributed offloading in multi-access edge computing systems: A mean-field perspective. *IEEE Trans. Mob. Comput.* **2025**, *25*, 7592–7610. [CrossRef]
23. Pirhosseini, H.; Zarei, E.; Rezvani, M.H.; Pourkabirian, A.; Shankar, A.; Bacanin, N.; Maple, C.; Viriyasitavat, W. Game-theoretic methods for edge/fog/cloud computation offloading: A systematic review. *Computing* **2025**, *107*, 166. [CrossRef]
24. Miriyala, S.; Chirra, V.R. Deep learning approaches for computation offloading in edge computing: A critical review: S. Miriyala, VR Chirra. *Telecommun. Syst.* **2026**, *89*, 42.
25. Wang, C.; Zhang, S.; Chen, Y.; Qian, Z.; Wu, J.; Xiao, M. Joint configuration adaptation and bandwidth allocation for edge-based real-time video analytics. In *Proceedings of the IEEE INFOCOM 2020-IEEE Conference on Computer Communications*; IEEE: Piscataway, NJ, USA, 2020; pp. 257–266.
26. Gu, H.; Zhao, L.; Han, Z.; Zheng, G.; Song, S. AI-enhanced cloud-edge-terminal collaborative network: Survey, applications, and future directions. *IEEE Commun. Surv. Tutor.* **2023**, *26*, 1322–1385. [CrossRef]
27. Rahmati, I.; Shah-Mansouri, H.; Movaghar, A. QECO: A QoE-oriented computation offloading algorithm based on deep reinforcement learning for mobile edge computing. *IEEE Trans. Netw. Sci. Eng.* **2025**, *12*, 3118–3130. [CrossRef]
28. Tong, Z.; Zhang, Y.; Mei, J.; Chen, C.; Li, K. Game-theoretic bandwidth allocation and task offloading in cloud-edge collaboration. *IEEE Internet Things J.* **2026**, *13*, 22425–22439. [CrossRef]
29. Prangchumpol, D.; Sophatsathit, P.; Lursinsap, C.; Tantatsanawong, P. Resource Allocation with Exponential Model Prediction for Server Virtualization. *J. Digit. Inf. Manag.* **2015**, *13*, 75–83.

30. Saxena, D.; Kumar, J.; Singh, A.K.; Schmid, S. Performance Analysis of Machine-Learning-Based Cloud Workload Prediction Models. *IEEE Trans. Parallel Distrib. Syst.* **2023**, *34*, 1461–1476. [CrossRef]
31. Boujelben, Y.; Hamid, S.A.; Zerai, F. Distributed Asynchronous Server Selection in Cloud Gaming with Delayed Load Information. In *Proceedings of the 2026 IEEE Wireless Communications and Networking Conference (WCNC)*; IEEE: Kuala Lumpur, Malaysia, 2026; pp. 1–6. [CrossRef]
32. Hong, H.J.; Chen, D.Y.; Huang, C.Y.; Chen, K.T.; Hsu, C.H. Placing Virtual Machines to Optimize Cloud Gaming Experience. *IEEE Trans. Cloud Comput.* **2015**, *3*, 42–53. [CrossRef]
33. Pardalos, P.M.; Pitsoulis, L.S.; Mavridou, T. Nonlinear assignment problems. In *Large Scale Computations in Air Pollution Modelling*; Springer: Dordrecht, The Netherlands, 1999; pp. 261–273.
34. Hochbaum, D.S. Complexity and algorithms for nonlinear optimization problems. *Ann. Oper. Res.* **2007**, *153*, 257–296. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.