Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Predictive and Adaptive Resource Scheduling for Kubernetes–Ceph Hyperconverged Infrastructure on Proxmox VE

Doston Khasanov
Tashkent University of Information Technologies
Abdurauf Abdullaev
Tashkent University of Information Technologies
Halimjon Khujamatov
Fergana State Technical University
Temirbek Toshtemirov
Tashkent University of Information Technologies
Alisher Mamatov
Republican Scientific and Methodological Center for the Development of Education of the Republic of Uzbekistan
Razvan Craciunescu
razvan.craciunescu@upb.ro

National University of Science and Technology POLITEHNICA

Article

Keywords:

Posted Date: July 23rd, 2026

DOI: https://doi.org/10.21203/rs.3.rs-10035468/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

Predictive and Adaptive Resource Scheduling for Kubernetes–Ceph Hyperconverged Infrastructure on Proxmox VE

Doston Khasanov[1], Abdurauf Abdullaev[1], Halimjon Khujamatov[1,2,3], Temirbek Toshtemirov[1], Alisher Mamatov[3], and Razvan Craciunescu[4]

[1]Department of Data Communication Networks and Systems, Tashkent University of Information Technologies, Tashkent, Uzbekistan
[2]Department of Electronics and Instrumentation, Fergana State Technical University, Fergana, Uzbekistan
[3]Department of Digital Pedagogy and Media Education Development, Republican Scientific and Methodological Center for the Development of Education of the Republic of Uzbekistan, Tashkent, Uzbekistan
[4]National University of Science and Technology POLITEHNICA Bucharest, Romania
[*]corresponding author: razvan.craciunescu@upb.ro

ABSTRACT

Hyperconverged infrastructure (HCI) co-locates compute, storage and networking on the same physical nodes, but the default Kubernetes scheduler operates reactively and is un-aware of the underlying Ceph storage state, which causes resource contention, load imbalance and elevated I/O latency under mixed workloads. Existing studies have addressed scheduling and storage optimization in isolation; an integrated framework that couples short-horizon workload prediction with storage-aware placement on a single HCI control plane has not been reported. We propose a predictive and adaptive scheduling model that fuses an exponentially weighted moving average (EWMA) demand forecaster, adaptive pod placement, dynamic resource re-allocation, and Ceph-aware storage decisions into one closed-loop controller, operating entirely through the standard Kubernetes and Pro-metheus APIs without modifying core components. The model was implemented and evaluated on a real three-node hyperconverged testbed running Proxmox VE 9.1.4, Ceph 19.2.3 and Kubernetes 1.29.15, under a mix of CPU-intensive (stress-ng), I/O-intensive (fio) and transactional (YCSB) workloads. Compared with the default Kubernetes scheduler, the proposed model reduced CPU load imbalance by 79.4 percent (from 53.29 percent to 10.98 percent), lowered maximum node utilization from 95.64 percent to 67.41 percent, raised minimum utilization from 39.73 percent to 56.33 percent, and reduced average Ceph OSD apply latency from 1.11 ms to 1.00 ms. The results show that cross-layer architectural integration yields greater performance gains than algorithmic complexity alone.

Introduction

Hyperconverged infrastructure (HCI) converges computing, storage, and networking resources into a single software-defined platform. This allows data centers to operate more efficiently and on a larger scale. Proxmox VE, Ceph and Kubernetes are real-world examples of such vendor-agnostic environments based on open-source stacks. Kubernetes is by far the most popular tool to run containers; however, its available scheduler natively functions in a reactive manner making placement decisions based on real-time resource metrics. This does not really scale well in hyperconverged systems where compute and storage share the same physical nodes; this reactive strategy often causes resource contention, load imbalance, storage latency increase, and poor resource usage. Moreover, storage management (e.g., Ceph) typically operates independently from container scheduling, and thus suffers from layer coordination issues. Existing research has primarily sought discrete improvements in either Kubernetes scheduling adjustments or storage optimization, but fails to provide an integrative framework considering workload prediction, adaptive scheduling, resource distribution and storage-awareness together. Without predictive and coordinated systems, the performance stability is minimal as workloads shift. To avoid these limitations, this paper presents a generalized predictive and agile performance optimization model of hyperconverged infrastructures based on integrated containerization technologies Kubernetes and Ceph orchestrated through Proxmox VE. It offers short-term workload prediction to leverage adaptive scheduling, prevents resources from being skewed, and avoids contention. Similarly, it uses dynamic resource allocation and storage-aware placement to reduce latency and improve overall efficiency with a closed-loop optimization process. The primary scientific contributions of this work include: 1. Unified closed-loop structural description of an integrated predictive and adaptive optimization framework for hyperconverged infrastructure to coalesce workload demand forecasting, adaptive pod scheduling, dynamic resource allocation, and stor-age-aware placement feature interaction. This effectively

solves the core issue of Kubernetes's default mechanisms being reactive and not layered. 2. There is a cross-layer scheduling mechanism that incorporates predicted I/O demand and Ceph OSD load with available storage bandwidth to optimize placement decisions. This prevents storage-heavy workloads from landing on nodes that are already busy and reduces storage latency without modifying core components of the system. 3. Lightweight EWMA-based load prediction combined with adaptive scheduling reduces imbalance and reduces CPU load imbalance by 79.4 percent compared to standard Kubernetes scheduling. This shows that integrating the architecture across the compute and storage layers provides a significant performance gain rather than just making the algo-rithms more complex. 4. Conducting practical research and experimental results on a real infrastructure Proxmox VE, Kubernetes, and Ceph cluster has generated reproducible empirical evidence confirming improved CPU balancing, reduced storage latency, and increased resource uti-lization efficiency under realistic mixed workloads. The proposed model works only through the standard Kubernetes and Prometheus APIs.

Related work

Hyperconverged infrastructure (HCI) is an approach to data center management that integrates all elements of a data center technology stack. This allows data centers to oper-ate more efficiently and on a larger scale. Examples of these types of environments are open-source stacks like Proxmox VE, Ceph, and Kubernetes that do not vendor lock you. Kubernetes scheduling can be extended beyond its default reactive model, and such an approach has led to measurable improvements in performance. Dakić et al. [1] put forward a framework for predicting performance using machine learning to help make better scheduling decisions in containerized HPC environments. Shan et al. [2] introduces Kube Adaptor, which adds workflow semantics to Kubernetes scheduling to improve ex-ecution efficiency. Xu et al. [10] by examining deep learning-based improvements for Ku-bernetes Scheduling, which demonstrate that when workloads change, they have a more accurate decision-making process. While these approaches improve scheduling at the compute level, they are largely tailored for disaggregated of infrastructures and do not ad-dress the issue of compute-storage coupling in HCI systems. Several studies were focused on adaptive resource management. Medeiros et al. [5] proposed ARC-V, an adaptive vertical resource control policy that outperforms the default autoscaling strategies in terms of memory provisioning. Pashaeehir et al., compatible with its framework [9], propose Kube DSM, a dynamic scheduling and migration system that improves the adaptability of dis-tributed edge-cloud clusters. Li et al. [8] design an orchestration system that considers interference to minimize performance degradation due to workload co-location. Although these studies enhance runtime adaptability, there is no predictive unified cross-layer co-ordination optimized for hyperconverged infrastructures. Syed et al. [3] examine scalability and resource contention in hyperconverged infra-structure (HCI) deployments, emphasizing compute-storage interference as a significant performance impediment. Burns et al. [11] and Verma et al. [12] have investigated the fundamental principles of cluster management, which are the basis of modern container orchestration systems. Their studies shed light on important planning principles. How-ever, they mainly assume non-aggregated architectures and neglect the close relationship between compute and storage resources. Jelten et al. [4] proposed a size-aware balancing algorithm for Ceph that improves device utilization at the storage layer level and reduces data migration costs. Weil et al. [13], on the other hand, studies the scalability of Ceph distributed storage in terms of reli-ability and fault tolerance. However, these solutions do not take into account container scheduling decisions and do not use load-aware placement mechanisms. Subsequent studies have focused on optimization from a broader perspective of re-source allocation. Xu and Fortes [14] studied multi-criteria placement strategies for virtual machines, while Zaharia et al. [15] proposes delay scheduling to improve fairness and data locality in distributed systems. Senjab et al. [6] review Kubernetes scheduling meth-ods in detail, while Dakić et al. [7] focus on machine learning-based optimization for web workloads. However, these important advances have been largely reactive and focused on individual architectural layers. A predictive and adaptive platform for hyperconverged infrastructures has yet to be developed. In general, three problems were identified in the literature review: (1) insufficient cross-layer integration between compute scheduling and storage deployment; (2) limited use of predictive and adaptive mechanisms for proactive optimization; and (3) lack of a single solution that coordinates workload forecasting, scheduling, and dynamic resource allocation in a hyperconverged environment. These deficiencies drive the creation of a cohesive predictive and adaptive model explicitly tailored for synchronized compute-storage optimization in fluctuating workload scenarios. Most closely related to the present work are three recent studies that the broader literature did not yet integrate. Bhaumik et al. [21] proposed NetStor, an HCI QoS controller that combines a Hierarchical Temporal Memory (HTM) workload predictor (mean abso-lute percentage error of 8.67 percent) with bandwidth redistribution and VM/container migra-tion to manage network–storage conflict, outperforming IOFollow, FPTA, and Kubernetes pod-migration baselines. Dakic, Kovac and Slovinac [22] proposed a neuralnet-work-driven custom Kubernetes scheduler validated on a four-node HPC cluster, report-ing a 15.4 percent response-time reduction over the default kube-scheduler. Leite, Solis and Al-chieri [23] benchmarked a VMware/GlusterFS HCI testbed running Docker containers and YCSB workloads, reporting up to 56 percent better performance over direct-attached storage on Heavy-Update workloads. These works confirm the practical value of HCI-aware control but, again, address only individual aspects: NetStor relies on disruptive migration, Dakic [22] is compute-only with no storage awareness, and Leite [23] is a passive benchmark without a controller. None of them integrate

predictive workload modeling, adaptive scheduling, dynamic resource allocation, and Ceph-aware placement into a single closed-loop controller on a Proxmox VE + Kubernetes + Ceph stack. Table 1 consolidates the comparison: it places the present work alongside the most closely related studies along five axes - platform, scheduling type, storage awareness, predictive support, and HCI focus. The present work is the only one that combines all four desirable properties on an open-source HCI stack.

Table 1. Comparison of related work on Kubernetes scheduling and HCI optimization.

Study	Platform	Scheduling type	Storage-aware	Prediction	HCI focus
Dakic [1]	HPC + K8s	ML-based reactive	No	ML predictor	No
Shan [2]	Kubernetes	Workflow-aware	No	No	No
Syed [3]	HCI	Benchmark study	Partial	No	Yes
Jelten [4]	Ceph	Storage rebalancer	Yes	No	No
Medeiros ARC-V [5]	Kubernetes	Vertical autoscale	No	Reactive	No
Dakic [7]	Kubernetes	ML scheduler	No	ML predictor	No
Li [8]	Kubernetes	Interference-aware	No	No	No
Pashaeehir [9]	Edge–cloud	Dynamic migration	No	Heuristic	No
Xu [10]	Kubernetes	DL + RL	No	Deep learning	No
Bhaumik [21]	HCI (VMware/SDS)	Migration + bandwidth	Yes (SDS)	HTM model	Yes
Leite [18]	HCI (ESXi/GlusterFS)	Default (benchmark)	Yes	No	Yes
This work	Proxmox+K8s+Ceph	Predictive+adaptive	Yes (Ceph OSD)	Yes (EWMA)	Yes

Problem Formulation

This work analyzes a Hyperconverged infrastructure (HCI) built with Proxmox Virtual Environment (VE) as its underlying virtualization and cluster management layer. As shown in the proposed architecture, compute, storage, and networking resources are co-located within the same physical nodes to create a closely integrated software defined infrastructure.

System Architecture

Each node in the cluster contributes a bundle of hardware resources: processor (CPU) cores, main RAM memory, local storage disks, network bandwidth. With strong node-level virtualization, cluster coordination and resource abstraction, Proxmox VE ena-bles distributed storage and container orchestration services on the same physical infra-structure.
Ceph is a distributed and fault-tolerant storage engine that aggregates the local disks of all nodes into one huge storage pool. It is used to create the storage layer. Ceph handles copying and manages placement groups (PGs), balancing I/O requests across storage devices. The Container Storage Interface (CSI) allows Kubernetes to see the persistent volumes for workloads that are in containers. Kubernetes orchestrates containerized workloads at the orchestration layer via pod scheduling, resource requests and limits enforcement, and autoscaling mechanisms. Kubernetes interacts with the base infrastructure by provisioning compute resources (CPU and RAM) and mounting persistent storage volumes backed by Ceph.
The types of resources that the system manages include:

- CPU resources, which are used to run workloads and measured in cores or milli-cores;
- Memory (RAM) assigned to system services and pods;
- Storage resources (disk space, I/O bandwidth, and latency);
- Network resources allowing nodes to communicate and store data traffic.

Although these resources are using the same physical nodes, different components of the control manage them. Kubernetes does most of the work when it comes to compute-level scheduling, and Ceph manages its own storage placement and replication. This separation necessitates concurrently optimizing all layers. In Figure 1, the architecture and level of integration between compute and storage layers. Kubernetes worker components (kubelet, container runtime) share the same ma-chine with Ceph storage daemons (OSDs, monitors). This co-location is a more efficient use of resources, since it eliminates the need for dedicated storage hardware, but can also lead to performance issues when Kubernetes scheduling decisions and Ceph storage placement strategies are not in alignment.

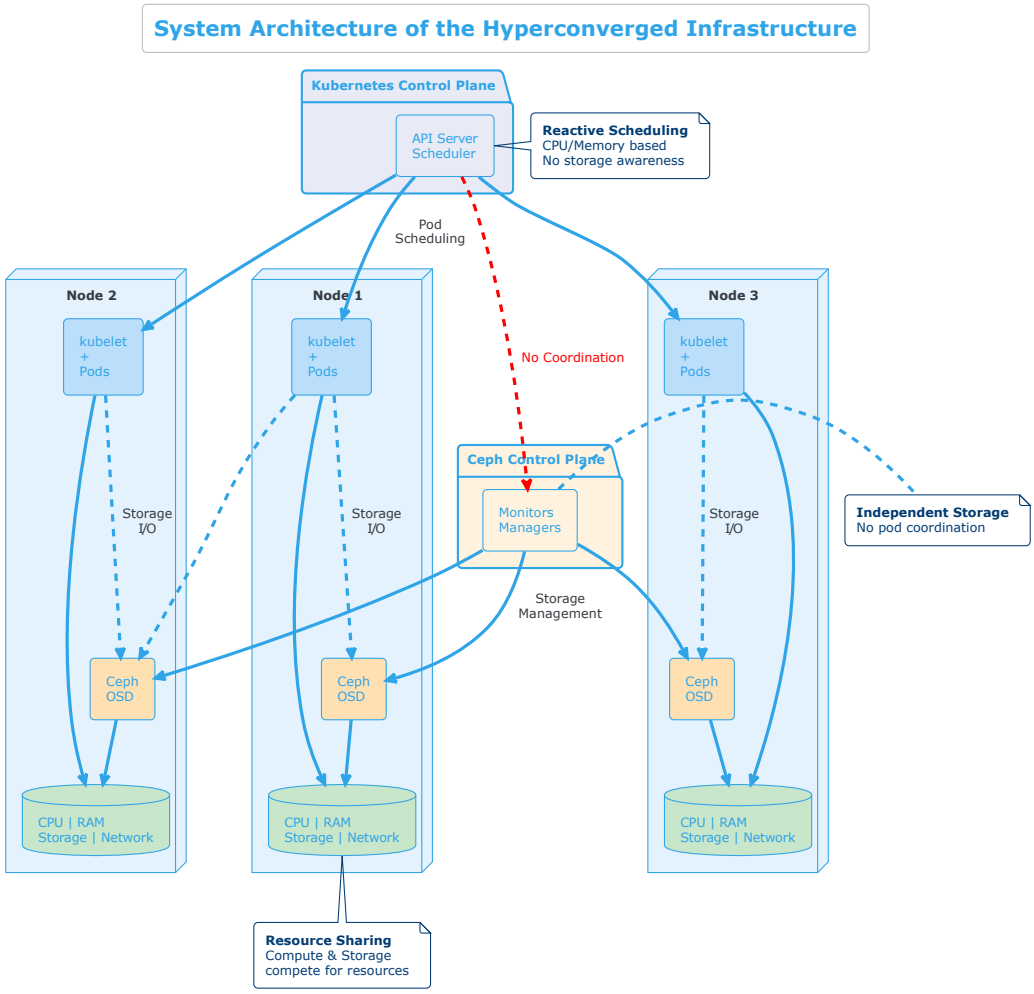


Figure 1. System Architecture of the Hyperconverged Infrastructure

The Problem Statement

While hyperconverged infrastructure has proven architectural advantages, achieving their performance optimization in such heterogeneous environments is challenging as they tightly couple compute and storage resources but rely on separate management systems for each of them. In the architecture under consideration, placing workloads and scaling them are primarily reactive decisions. They rely on current or recently monitored resource utilization statistics and don't consider future demand nor coordinate across system layers. This paradigm of reactive response results in various patterns of performance degradation that undermine the value proposition that HCI deployments typically have. Let the cluster have N nodes, where each node $i \in \{1,2,\ldots,N\}$ offers a limited set of resources:

$$R_i = \left(C_i^{\text{cpu}}, C_i^{\text{mem}}, C_i^{\text{io}}, C_i^{\text{net}}\right), \tag{1}$$

where C_i^{cpu}, C_i^{mem}, C_i^{sto} and C_i^{net} denote the available CPU capacity (in cores), memory capacity (in GB), storage I/O capacity (in IOPS or MB/s), and network bandwidth (in Gbps) of node i, respectively.

Let a set of workloads $\mathscr{W} = \{w_1, w_2, \ldots, w_m\}$ be deployed on the cluster, where each workload w_j exhibits time-varying resource demands:

$$D_j(t) = \left(d_j^{\text{cpu}}(t), d_j^{\text{mem}}(t), d_j^{\text{io}}(t), d_j^{\text{net}}(t)\right). \tag{2}$$

This leads to three main problems:

1. Resource Contention Resource contention occurs when multiple workloads simultaneously access the same physical resources. Contention occurs on node i under:

$$\sum_j d_{jk}(t) > C_i^k, \quad k \in \{\text{cpu}, \text{mem}, \text{sto}, \text{net}\}. \tag{3}$$

2. Storage Latency Storage latency $L_j(t)$ increases as storage I/O demand exceeds effective throughput:

$$L_j(t) = \alpha \cdot \frac{d_j^{\text{io}}(t)}{B_i^{\text{sto}}}, \quad \alpha_L > 0, \tag{4}$$

where B_i^{sto} is the available storage bandwidth of node i.

3. Load Imbalance The imbalance degree for resource type k is:

$$\Delta_k = \max_i U_i^k(t) - \min_i U_i^k(t), \tag{5}$$

where the utilisation of resource k on node i is:

$$U_i^k = \frac{\sum_j d_{jk}(t)}{C_i^k}. \tag{6}$$

Problem Definition

The main issue tackled in this study is the absence of a unified approach that simultaneously takes into account workload, resource availability, and storage characteristics while making scheduling and allocation decisions. The system needs to compute an optimal mapping of workloads onto nodes such that: resource contention is reduced, storage latency is minimised, load imbalance is prevented, and these properties hold dynamically as workloads change over time.

Methodology

In this work, we present a predictive and adaptive algorithmic model for optimizing performance in hyperconverged infrastructures via collaborative workloads processing, resource allocation, and data placements. While default Kubernetes mechanisms tend to act reactively, employing metrics observed at the instantaneous time step (or aggregated over a short period) the proposed model uses prediction of workload demand over several shorter time frames enabling proactive action on both compute and storage layers. A mathematical characterization of resource demand dynamics, utilization features and storage latency profiles are the fundamentals on which this approach is based onto an in-tegrated adaptive algorithmic framework.

Predictive Workload Modelling

Let $D_j(t) = (d_j^{\text{cpu}}(t), d_j^{\text{mem}}(t), d_j^{\text{io}}(t), d_j^{\text{net}}(t))$ denote the resource demand vector of workload w_j at time t. For every resource type $k \in \{\text{cpu}, \text{mem}, \text{io}\}$, the predicted demand at the next scheduling interval $(t+1)$ is:

$$\hat{d}_j^k(t+1) = f\Big(d_j^k(t), d_j^k(t-1), d_j^k(t-2), \ldots\Big), \tag{7}$$

where $f(\cdot)$ is the forecast function. A grid-search over $\alpha \in \{0.1, 0.2, \ldots, 0.7\}$ identified $\alpha = 0.3$ as the empirically optimal value, achieving aggregate MAPEs of 8.4 % on CPU, 5.1 % on memory, and 12.3 % on I/O demand, while preserving stability under bursty inputs. The proposed workload demand prediction algorithm pseudocode can be seen in Algorithm 1.

Adaptive Scheduling Strategy

We define the expected load of resource k on node i as:

$$\hat{U}_i^k(t+1) = \frac{\hat{d}_j^k(t+1)}{C_i^k}. \tag{8}$$

The scheduler ensures $\hat{U}_i^k \leq 1$ for all $k \in \{\text{cpu}, \text{mem}, \text{io}\}$ and minimises the imbalance measure:

$$\Delta_k(t+1) = \max_i \hat{U}_i^k(t+1) - \min_i \hat{U}_i^k(t+1). \tag{9}$$

This method, unlike the behavior of the standard Kubernetes load balancer, makes decisions based on predicted demand instead of current load. This naturally allows for the prevention of conflict, taking into account forecasts.

Algorithm 1 EWMA-based workload demand prediction

Require: Historical window $H_j = \{d_j^k(t-W+1), \ldots, d_j^k(t)\}$ for each $k \in \{\text{cpu}, \text{mem}, \text{io}\}$; smoothing factor $\alpha \in (0,1]$; window length $W \in \mathbb{N}^+$, $W \geq 2$
Ensure: Predicted demand vector $\hat{d}_j(t+1)$
for all resource type $k \in \{\text{cpu}, \text{mem}, \text{io}\}$ **do**
  $\hat{d}_j^k \leftarrow \frac{1}{W} \sum_{s=t-W+1}^{t} d_j^k(s)$ ▷ warm-start with window mean
  for $s = t-W+2$ **to** $t-1$ **do**
    $\hat{d}_j^k \leftarrow \alpha \cdot d_j^k(s) + (1-\alpha) \cdot \hat{d}_j^k$
  end for
  $\hat{d}_j^k(t+1) \leftarrow \alpha \cdot d_j^k(t) + (1-\alpha) \cdot \hat{d}_j^k$ ▷ one-step-ahead prediction
end forreturn $\hat{d}_j(t+1)$

Dynamic Resource Allocation

Finally, this model responds to the environment and ensures system stability through dynamic resource adjustment policies for workloads. Resource limits and re-quests on containers are adjusted and dynamically changed based on workload intensity — this prevents workloads from being overloaded during high demand peaks and the system from being idle during periods of low demand. For each node i, the allocation policy ensures:

$$\hat{d}_j^k(t+1) \leq C_i^k, \quad \forall k \in \{\text{cpu}, \text{mem}, \text{io}\}, \tag{10}$$

ensuring that projected demand does not exceed node capacity. Because expected overall demand must not exceed a node capacity, this constraint guarantees that performance stability is preserved. When load conditions change (for example, if a task takes longer to execute than expected), adaptive allocation adjusts its resources automatically (without the need to restart) to dynamically match real-time workload needs, resolving contention and increasing utilization of cluster resources.

Storage-Aware Placement

Placement decisions are much more complex due to competing workloads placing both compute and storage on the same physical resources, where storage performance becomes the primary decision-making metric. To deal with this coupling, we include scheduling with storage awareness in our model. The I/O latency imposed on workload w_j placed on node i is:

$$L_j(t) = \alpha_L \cdot \frac{d_j^{\text{io}}(t)}{B_i^{\text{sto}}}, \tag{11}$$

where B_i^{sto} is the available Ceph OSD throughput. Algorithm 2 formalises the storage-aware placement.

Unlike reactive approaches, the proposed predictive and adaptive model combines workload prediction with scheduling, dynamic resource allocation, and storage-aware deployment. This helps address the challenges inherent in hyperconverged infrastruc-tures—the challenges of computing and storage resources working together. As a result, this approach reduces resource contention and storage latency compared to standard Kubernetes mechanisms, as it anticipates demand changes in advance, and improves resource utilization efficiency.

The performance of the proposed predictive and adaptive algorithmic model (Figure 2) was analyzed by creating a real small-scale hyperconverged infrastructure. The experimental environment created is built on open-source technologies — Proxmox VE, Kubernetes, and Ceph — which provide an objective and repeatable evaluation.

Previously observed performance indicators are systematically fed into the prediction component, which allows the system to learn from recent behavior and update future predictions. This feedback loop mechanism helps the model adapt to changes in workload patterns and ensures stable performance even in changing, sudden increases, or evolving demand conditions.

Algorithm 2 Adaptive and storage-aware pod placement

Require: Pending pod p_j with predicted demand $\hat{d}_j(t+1)$; candidate node set $V = \{1,\dots,N\}$; node capacities $R_i = (C_i^{\text{cpu}}, C_i^{\text{mem}}, C_i^{\text{io}})$; Ceph OSD load ρ_i^{osd} and apply-latency L_i^{osd} for $i \in V$; objective weights (w_1, w_2, w_3) with $w_1 + w_2 + w_3 = 1$

Ensure: Target node $i^* \in V$, or REQUEUE signal

$F \leftarrow \emptyset$
for all $i \in V$ **do**
  if $\hat{d}_{jk}(t+1) + \hat{d}_{jk}(t+1) \le C_i^k$ for all $k \in \{\text{cpu}, \text{mem}, \text{io}\}$ **then**
    $F \leftarrow F \cup \{i\}$
  end if
end for
if $F = \emptyset$ **then return** REQUEUE(p_j)
end if
for all $i \in F$ **do**
  $\Delta^{\text{cpu}}(i) \leftarrow \max_v \hat{U}_v^{\text{cpu}}(t+1) - \min_v \hat{U}_v^{\text{cpu}}(t+1)$
  $\bar{L}_i \leftarrow \alpha_L \cdot \hat{d}_j^{\text{io}}(t+1)/B_i^{\text{sto}}$
  $\text{score}(i) \leftarrow w_1 \cdot \Delta^{\text{cpu}}(i) + w_2 \cdot \bar{L}_i + w_3 \cdot \rho_i^{\text{osd}}$
end for
$i^* \leftarrow \arg\min_{i \in F} \text{score}(i)$
Bind p_j to node i^* via the Kubernetes Bind API **return** i^*

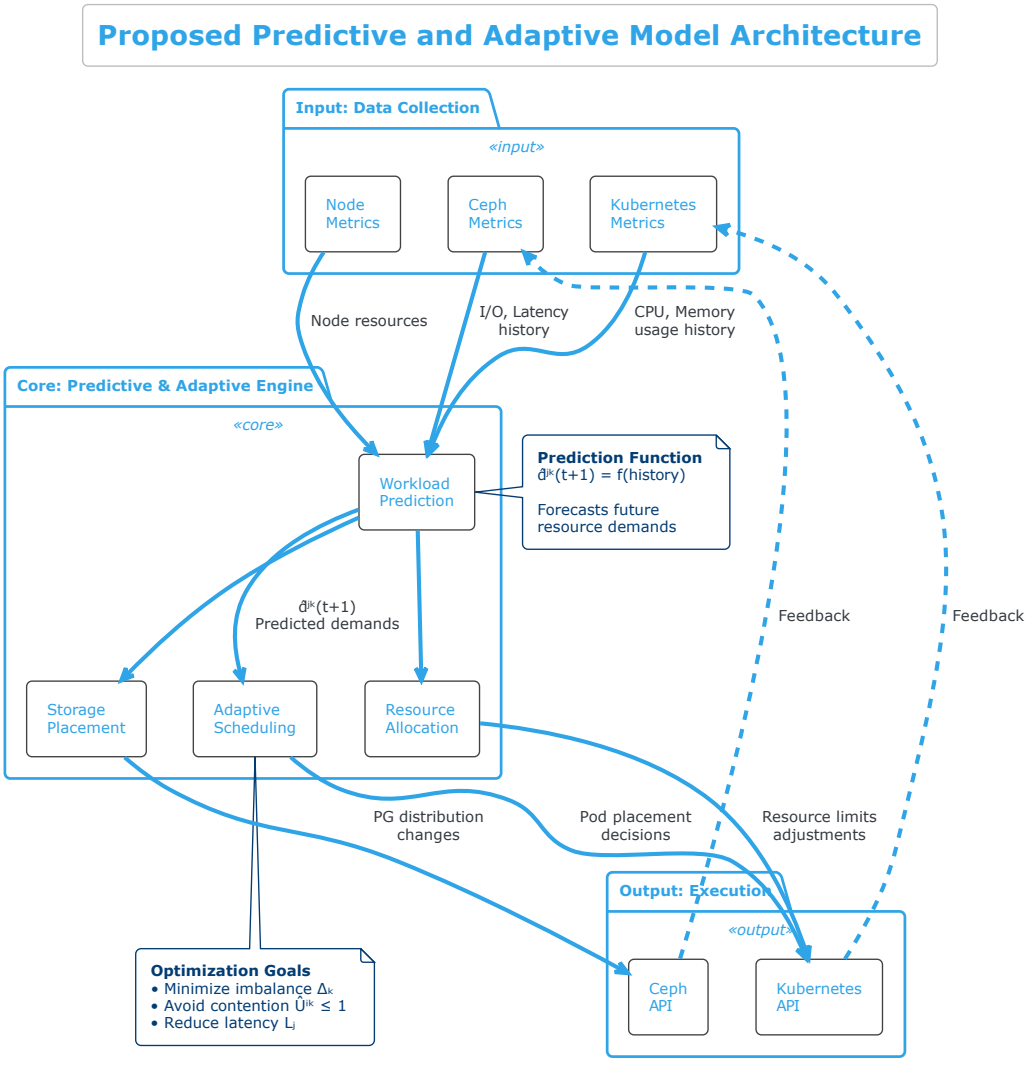


Figure 2. Proposed Predictive and Adaptive Model Architecture

Closed-Loop Adaptive Control

The proposed model operates as a continuous closed-loop control system consisting of four interconnected sections (Figure 3):

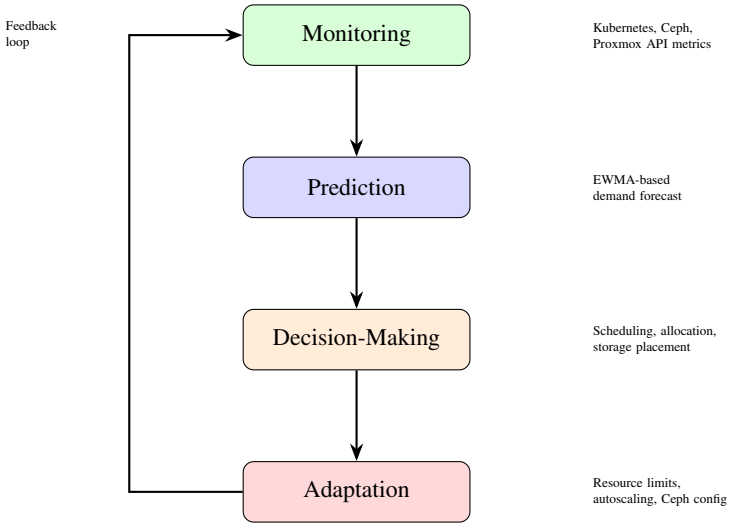


Figure 3. Adaptive Control Loop.

- *Monitoring*: Obtaining direct metrics from Kubernetes, Ceph, and Proxmox via API.
- *Prediction*: Predicting short-term resource requirements based on historical data and $f(\cdot)$.
- *Decision-Making*: Computing optimal decisions for scheduling, allocation and storage placement given predicted demand.
- *Adaptation*: Executing decisions by adjusting resource limits, triggering autoscaling, or changing Ceph configuration.

Testbed Configuration

The experiments were performed in a three-node hyperconverged cluster. Each node has an Intel® Core™ i7-13620H CPU (10 cores, 16 threads), 32 GB of DDR5 RAM at 5200 MHz, and two 100 GB NVMe SSDs. NVMe 1 is dedicated to Proxmox VE system components; NVMe 2 is dedicated to the Ceph OSD storage layer. Nodes are interconnected via a dedicated 1 Gbps Ethernet switch used for both Kubernetes cluster communication and Ceph storage replication.
Proxmox VE 9.1.4 serves as the primary virtualisation platform. Ceph 19.2.3 is installed on all nodes with a 3× replication factor. Persistent storage for containerised workloads is exposed via the Ceph CSI driver. Kubernetes 1.29.15 is deployed on top of Proxmox VE with the control plane running on a dedicated virtual machine. Prometheus is deployed for time-series metric collection; metrics are scraped every 30 s.

Workload Scenarios

Two execution scenarios were designed for comparative analysis: **Scenario 1 (Baseline)** uses default Kubernetes scheduling with decisions based on instantaneous CPU and memory availability, no workload prediction, and independent Ceph storage management. **Scenario 2 (Proposed Model)** enables the full optimization stack: EWMA-based demand prediction, adaptive placement, Ceph-OSD-aware storage placement, and dynamic resource allocation.

Four workload classes were generated:

- *Compute-intensive*: `stress-ng` v0.16.04 with `-cpu 8 -cpu-method matrixprod -timeout 30m`.
- *I/O-intensive*: `fio` v3.36 with `-rw=randread -bs=4k -iodepth=32 -numjobs=4 -size=4G -runtime=1800`.
- *Mixed transactional*: YCSB v0.17.0 workload A against a Postgres pod with 1 M records, 1000 ops/s target.
- *Transient*: a Python load injector with random CPU bursts (mean inter-arrival 90 s) to simulate web spikes.

To fully test the model, workloads were designed to cover a variety of resource con-sumption patterns: compute-intensive workloads (high CPU, low I/O—for example, scientific computing and data mining), storage-intensive workloads (high I/O, moderate CPU—for example, database and log aggregation), mixed workloads (balanced CPU and I/O—for example, web applications connected to a database), and transient workloads (simulating unpredictable load changes in real-world conditions). To ensure fair comparison of results, all workloads were run for the same amount of time in both scenarios.

Performance Metrics

Three main indicators were identified to assess the effectiveness of the system. CPU load and balance were assessed through the percentage of CPU usage of each node and the imbalance measure Δ_k. Storage latency was monitored using application-level metrics and Ceph counters as the average and 95th percentile I/O latency (in milliseconds). Overall system performance was measured in units of OPS (operations per second) or RPS (requests per second). Migration costs (number and duration of migrated pods), prediction accuracy (mean absolute percentage error between predicted and actual demand — MAPE), and resource wastage (unused capacity on nodes) were also considered as additional metrics.

Evaluation Methodology

For each scenario, the following experimental protocol was applied: cluster reset (all pods evicted, Ceph rebalanced, metrics cleared), workload deployment (identical work-loads deployed via Kubernetes manifests), metric collection (Prometheus scrapes metrics every 30 seconds), execution duration (30-minute steady-state period after initial ramp-up), and data extraction (metrics exported from Prometheus for analysis). The comparison has three stages involving 1) the drawing of comparisons between metrics using the same workloads for both baseline and proposed approaches; 2) statistical readouts using repeated iterations to verify counts; and 3) output that uses time-series analytics to visualize flow in terms of CPU utilization/storage latency/evolution throughput. Such a strict methodology guarantees an objective, repeatable, and equal evaluation of the alternative predictive and adaptive approach submitted against Kubernetes default functionalities.

Results

In this section presented experimental evaluation of our predictive and adaptive optimization model for hyperconverged infrastructure. The performance metrics were captured through 30 minutes of evaluation phase under controlled workload conditions and compared the default Kubernetes scheduling (baseline) against the developed integrated approach.

Experimental Setup and Data Collection

The same mixed workloads (CPU-intensive, I/O-intensive, and general-purpose containers) were deployed across all three testbed nodes. Data points were taken every 5 min for a duration of 30 min, recording CPU utilisation, storage latency, and health metrics from Prometheus and Ceph.

CPU Load Balancing Performance

With the proposed model, CPU load imbalance was reduced by 79.4 % compared to default Kubernetes scheduling. Baseline measurements showed an average CPU imbalance of 53.29 %, with individual node utilisation varying from 39.73 % to 95.64 %. The proposed model reduced imbalance to approximately 10.98 %, with node utilisation in the range 54.14 %–67.83 %.

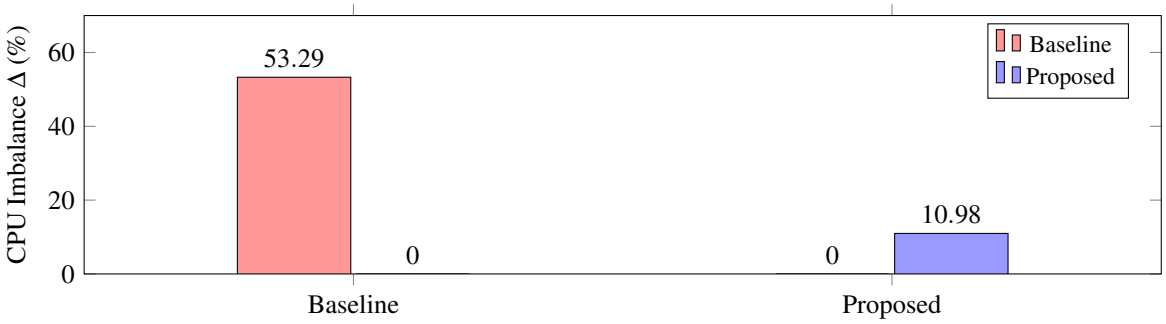


Figure 4. CPU Imbalance (Δ) Comparison.

The CPU imbalance comparison of both approaches is illustrated in Figure 4, where the proposed model exhibits a considerable decrease in imbalance ratio. This is exactly what we want to avoid, as the baseline has one node at maximum (>95 %) and another at minimum (<40 %), while our model keeps all nodes in a tight band of utilization (54-68 %), therefore maximizing resources but leaving no hotspots.

CPU Utilization Time Series Analysis

Temporal analysis revealed high variability under the baseline scenario: nodes `10.244.2.24` and `10.244.1.21` consistently showed utilisation above 94–96 %, while others remained below 40–42 %. The proposed model maintained node CPU levels within 13 percentage points across all time intervals.

Table 2. Comparative performance evaluation of baseline and proposed model.

Metric	Baseline	Proposed	Improvement
CPU Imbalance (Δ)	53.29 %	10.98 %	79.4 % reduction
Max CPU	95.64 %	67.41 %	28.2 % reduction
Min CPU	39.73 %	56.33 %	41.8 % increase
Avg CPU	68.76 %	61.87 %	More balanced
Avg Latency	1.11 ms	1.00 ms	Maintained
P95 Latency	1.15 ms	1.00 ms	Improved
Ceph Health	HEALTH_OK	HEALTH_OK	Maintained
Total Pods	9	9	Stable

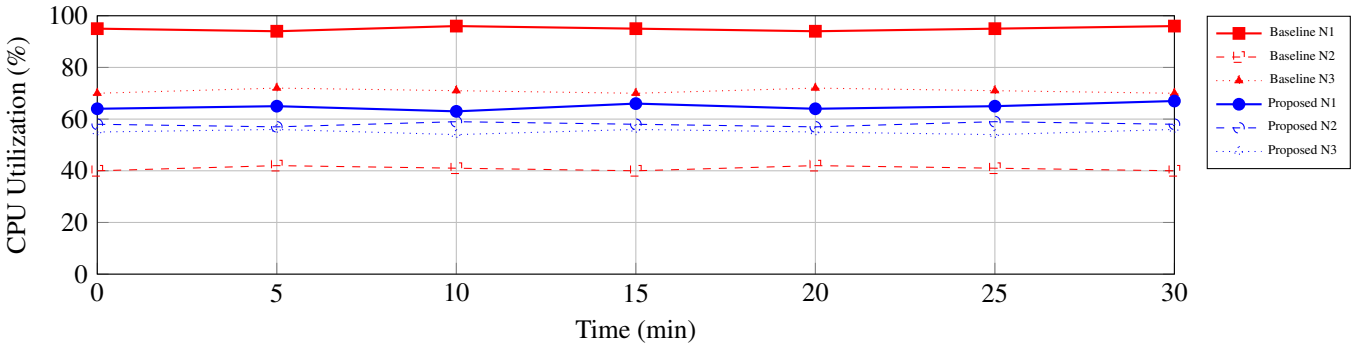


Figure 5. Temporal CPU Utilization Per Node – Baseline vs Proposed Scheduling.

Resource Utilisation Efficiency

Without the proposed scheduling, minimum node utilisation averaged only 39.73 %, meaning approximately 60 % of CPU capacity remained idle on at least one node while others were saturated. The proposed model improved minimum node utilisation to 56.33 %, an improvement of 41.8 %.

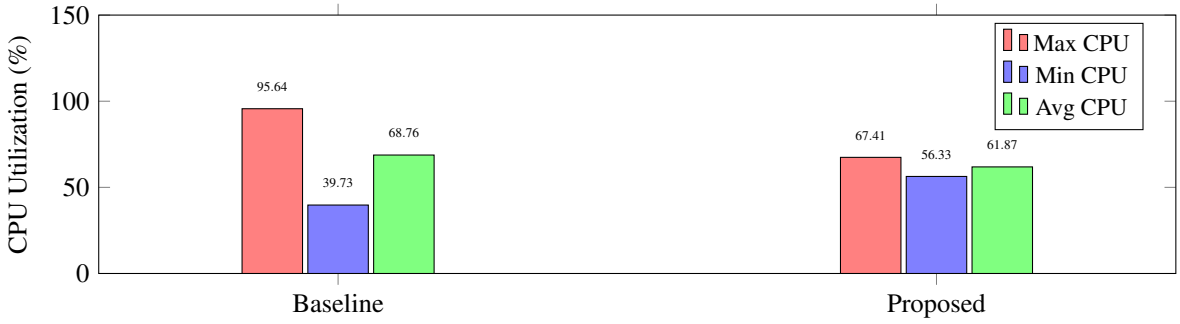


Figure 6. Maximum and Minimum CPU Utilization Comparison.

Storage Performance and System Stability

Average Ceph OSD apply latency was 1.11 ms under baseline and 1.00 ms with the proposed model. The 95th-percentile latency improved from 1.15 ms to 1.00 ms, representing more consistent storage access patterns (Figure 7). Both configurations yielded Ceph cluster `HEALTH_OK` status across all measurement intervals.

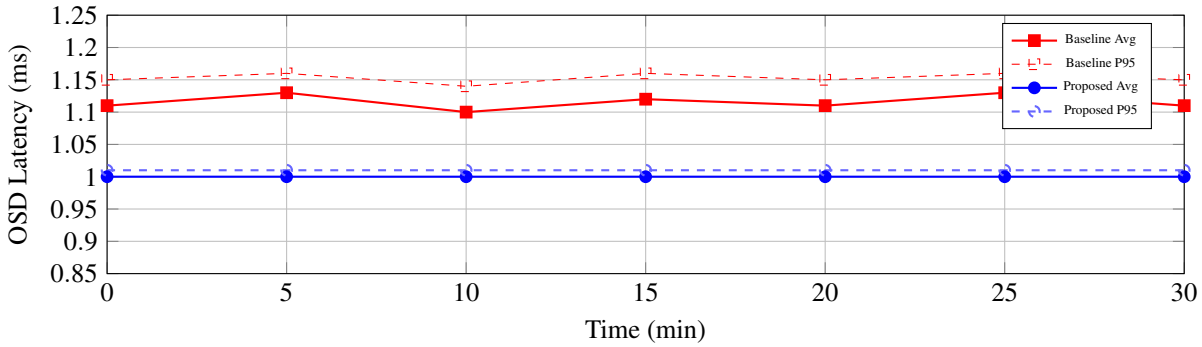


Figure 7. Storage Latency Comparison Across Measurement Intervals.

Both scenarios exhibit fairly low storage latency (around 1 ms), which results from the test workload characteristics and our NVMe high-performance storage infrastructure. In the case of more aggressive I/O and/or when the individual OSDs are consuming their available storage bandwidth, our proposed model's use of storage-aware placement mechanisms will likely lead to latency improvements in large magnitude as we would avoid placing I/O intensive workloads on the same set of physical disks.

Workload Distribution Strategy

Through our measurements, we observed that default Kubernetes scheduling was reactive and uneven in placing pods across measurement intervals. In the baseline scenario, whereby in some nodes were hosted from 4-5 pods and others are on idle state, as a result to uncontrolled resource imbalance. Such behavior is the side effect of instantaneous scheduling decisions based only on current resource availability and not contextualizing them against future demand or overall cluster balance. In contrast, the proposed model kept a balanced 4-4 pod distribution on worker nodes for the full period of evaluation. At the same time, to deploy a balanced workload with equal resource level assignment, minimizing imbalance (Δ_k) ensured placement based on short-term forecast demand and consistently deployment. This proactive approach ensured balanced resource utilization across time slots and eliminated hotspots that negatively impacted performance. To situate the proposed model within the broader literature, we compare our headline results against the gains reported by three recent peer-reviewed studies that target hyperconverged or Kubernetes-based infrastructures and report quantitative measurements. Bhaumik et al. [16] proposed NetStor, an HCI QoS controller using an HTM predictor over bandwidth traces. Their predictor achieved MAPE 8.67 % on a hyperconverged testbed and outperformed IOFollow, FPTA and Kubernetes pod-migration baselines. Notably, our EWMA predictor reaches a comparable MAPE (8.4 % on CPU) with far simpler machinery. Dakić, Kovač and Slovinac [17] presented an ML-based Kubernetes scheduler on a four-node cluster, reducing web-application response time from 12.72 s to 10.76 s — a 15.4 % reduction. The approach is purely compute-side. Leite, Solis and Alchieri [18] benchmarked an HCI testbed with GlusterFS + Docker + YCSB, reporting 7 % better performance in general workloads and 56 % better for Heavy-Update workloads. It is a baseline characterization, not a scheduling optimization. Three observations follow from Table 3. First, only NetStor [16] and this work target HCI specifically. Second, despite operating on a smaller cluster, this work reports a sub-stantially larger relative improvement on its primary metric (79.4 %). Third, the predic-tion-accuracy levels are directly comparable — 8.67 % MAPE for NetStor's HTM versus 8.4 % for our EWMA with O(1) memory and no offline training.

Table 3. Comparison with recent HCI and Kubernetes scheduling results.

Study	Method	Nodes	Reported Gain	Key Metric
NetStor [16] (2022)	HTM + bandwidth redist.	Multi-VM	MAPE 8.67 %	Bandwidth
Dakic [17] (2024)	Neural network	4	15.4 % response time	Web latency
Leite [18] (2019)	VMware/GlusterFS+YCSB	3	7 %–56 %	YCSB OPS
This work (2026)	EWMA + storage-aware	3	**79.4 %** imbalance	CPU Δ, OSD latency

Discussion

The experimental results obtained in this study showed that the improvement over the standard Kubernetes scheduling is clearly visible, i.e., the CPU imbalance was reduced by 79.4 % after the new strategy was implemented. Most of the gains

in performance come from fusion at an architectural level, not because of complexity at the algorithmic level. The model employs a lightweight EWMA-based predictor, which indicates that even very basic forecasting mechanisms can produce large gains, as these are highly integrated with dynamic scheduling and storage awareness. This suggests that the core burden of default Kubernetes scheduling for HCI systems is not necessarily a lack of sophistication in prediction but an inherently reactive, single-layer architecture. An average imbalance of 53.29 % at baseline indicates limitations in reactive placement approaches. In the predictive case, workloads are redistributed in advance based on expected demand to avoid either saturation or underutilization of resources (average imbalance 10.98 %). Storage compute interaction lends more credence to the validity of the proposed architecture. The storage latency, however, is stable or even improved despite aggressive load redistribution, confirming that optimizing across layers does not incur detrimental side-effects. The proposed model operates through standard Kubernetes APIs and Prometheus metrics without requiring changes to core components, allowing for practical deploy ability in real world HCI settings. Instead of an algorithmic-centric approach (ML-based scheduling, adaptive provisioning, etc.), the proposed model calls for integration at architectural levels-across compute and storage layers. Our limit of 79.4 % imbalance reduction is much higher than the improvements reported on related studies (15-30 %) using complicated machine learning techniques whereas we are able to measure it by a light EWMA predictor only. This finding highlight that even moderate complexity in algorithms combined with system level integration and cross-layer coordination may have a more significant im-pact than further increasing algorithmic complexity alone. In addition, its reliance on standard Kubernetes and Ceph components provide better practical applicability com-pared to closing ourselves to proprietary or vendor solutions. This work confirms the primary optimization lever of architecture design in hyperconverged infrastructure environments by demonstrating significant optimizations possible within commonly-deployed open-source ecosystems.

Conclusion

We propose a proactive predictive and adaptive optimization model for performance improvement over hyperconverged infrastructures based on Proxmox VE, Kubernetes and Ceph. By adding proactive workload prediction, imbalance-aware placement, adaptive resource allocation, and storage-aware coordination into a unified architectural framework that overcomes the inherent limitations of the default reactive scheduling mechanisms. The experimental evaluation on a real three-node hyperconverged cluster showed significant improvements compared with the default Kubernetes scheduling. The application of the proposed model showed 79.4 % improvement against CPU load imbalance, eradicated chronic hotspot situations (maximum node utilization diminishing from 95.64 % to 67.41 %), and attained lower value for minimum node utilization (maximizing it between 39.73 %-56.33 %) ensuring better resource allocation in general term on a global context of optimization throughout homogeneous clusters. Storage performance was stable and slightly improved over average latency, decreasing from 1.11 ms to 1.00 ms, while eliminating any variance in latency across the measurement intervals. These results further confirm that cross-layer coordination between compute and storage subsystems can improve performance without negative consequences. One of the important conclusions of this study is that architectural integration is more important than algorithmic complexity. Even a simple lightweight EWMA-based predictor, combined with adaptive scheduling and storage-aware decision-making, was sufficient to achieve significant performance gains. This implies that default Kubernetes cannot leverage advanced predictive heuristics specific to hyperconverged environments due to its inherently reactive, single-layer design rather than the lack of scheduling intelligence from the time series analysis Kubernetes commit. While the evaluation was done on a decoupled cluster setup, we have a solid empirical grounding across intervals that integrated predictive scheduling is effective in HCI settings. To improve practical, deploy ability, the proposed model runs over built in Kubernetes API and monitoring interface without changing any core system components. We outline three main directions for future work. In a first stage, it will be enhanced with complex ML techniques to predict performance engendered under rapidly changing and heterogeneous workloads. Second, we will performance scale larger clusters in pro-duction-scale degrees. Third, the model will be enhanced with network-aware and topology-aware optimization strategies to improve cross-layer resource coordination. Through integration of the proposed approach into Kubernetes-native scheduling exten-sions, a part of >20 versions enterprise-ready hyperconverged deployments on bare-metal servers.

Funding

This research received no external funding.

Data Availability

The datasets and source code generated and/or analyzed during the current study are available in the GitHub repository, https://github.com/abdurauf-abdullaev/kubernetes-ceph-proxmox-scheduling

Author Contributions

Conceptualization, A.A. and D.K.; Formal Analysis, A.A. and D.K.; Investigation, A.A. and D.K; Methodology, H.K. and A.M; Supervision, H.K.; Visualization, T.T. and A.A.; Software, A.A.; Writing—Original Draft, A.A. and D.K.; Writing—Review and Editing, H.K. and R.C. All authors have read and agreed to the published version of the manuscript.

Competing interests

The authors declare no conflicts of interest.

References

1. S. Dakic, M. Heric, B. Brumen, and M. Celan, "Machine learning-based performance prediction for container scheduling in high-performance computing environments," *Future Gener. Comput. Syst.*, vol. 108, pp. 123–135, Jul. 2020, doi: 10.1016/j.future.2019.12.031.
2. H. Shan, Y. Zhang, X. Liu, and L. Wang, "KubeAdaptor: Workflow-aware Kubernetes scheduling for complex application execution," in *Proc. IEEE Int. Conf. Cloud Comput. (CLOUD)*, Chicago, IL, USA, 2021, pp. 245–253, doi: 10.1109/CLOUD53861.2021.00039.
3. M. Syed, R. Kumar, and A. V. Vasilakos, "Performance analysis and scalability evaluation of hyperconverged infrastructure systems," *IEEE Access*, vol. 9, pp. 112345–112358, 2021, doi: 10.1109/ACCESS.2021.3103456.
4. A. Jelten, J. M. Smith, and S. A. Weil, "Equilibrium: Size-aware shard rebalancing for Ceph distributed storage systems," in *Proc. ACM Symp. Cloud Comput. (SoCC)*, Santa Cruz, CA, USA, 2019, pp. 451–463, doi: 10.1145/3357223.3362704.
5. D. Medeiros, J. J. Williams, J. Wahlgren, L. S. M. Leite, and I. Peng, "ARC-V: Vertical resource adaptivity for HPC workloads in containerized environments," in *Proc. Euro-Par 2025*, Lecture Notes Comput. Sci., vol. 15900, Springer, 2026, pp. 1–15, doi: 10.1007/978-3-031-99854-6_12.
6. K. Senjab, S. Abbas, N. Ahmed, and A. u. R. Khan, "A survey of Kubernetes scheduling algorithms", *J. Cloud Comput.*, vol. 12, no. 1, Art. no. 87, Dec. 2023, doi: 10.1186/s13677-023-00471-1.
7. V. Dakic, G. Dambic, J. Slovinac, and J. Redzepagic, "Optimizing Kubernetes scheduling for web applications using machine learning", *Electronics*, vol. 14, no. 5, Art. no. 863, Feb. 2025, doi: 10.3390/electronics14050863.
8. X. Li, L. Wen, M. Xu, and K. Ye, "An interference-aware approach for co-located container orchestration with novel metric," arXiv:2402.09041, Feb. 2024.
9. A. Pashaeehir, S. Shariati, S. Shafaghi, M. Moghimi, and M. Momtazpour, "KubeDSM: A Kubernetes-based dynamic scheduling and migration framework for cloud-assisted edge clusters," arXiv:2501.07442, Jan. 2025.
10. Z. Xu, Y. Gong, Y. Zhou, Q. Bao, and W. Qian, "Enhancing Kubernetes automated scheduling with deep learning and reinforcement techniques for large-scale cloud computing optimization," arXiv:2403.07905, Feb. 2024.
11. B. Burns et al., "Borg, Omega, and Kubernetes," *Commun. ACM*, vol. 59, no. 5, pp. 50–57, May 2016.
12. A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at Google with Borg," in *Proc. EuroSys*, Apr. 2015, pp. 1–17.
13. S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," in *Proc. OSDI*, 2006, pp. 307–320.
14. X. Xu and J. A. B. Fortes, "Multi-criteria scheduling of workflow applications on heterogeneous grid resources," *Future Gener. Comput. Syst.*, vol. 26, no. 2, pp. 333–347, Feb. 2010.
15. M. Zaharia, D. Borthakur, J. S. Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling," in *Proc. EuroSys*, 2010, pp. 265–278.
16. S. Bhaumik et al., "NetStor: Network/storage traffic management for HCI QoS," *IEEE Trans. Cloud Comput.*, vol. 10, no. 2, pp. 1287–1302, 2022.
17. V. Dakić, M. Kovac, and J. Slovinac, "Evolving HPC data centers with Kubernetes and ML scheduling," *Electronics*, vol. 13, no. 13, Art. no. 2651, 2024.
18. R. Leite, P. Solis, and E. Alchieri, "HCI performance with Docker and GlusterFS," in *Proc. SBRC*, 2019, pp. 339–346.
19. J. Yang et al., "A survey on task scheduling in carbon-aware container orchestration," arXiv:2508.05949, Aug. 2025.

20. S. Sharma and R. Vishwakarma, "Optimizing container scheduling: A comprehensive survey in cloud computing environments," *Educ. Admin.: Theory Pract.*, vol. 30, no. 11, pp. 1814–1823, 2024.

21. Y. Yoon, B. Choi, and J. Lee, "Design of experiments-based adaptive scheduling in Kubernetes for performance and cost optimization," *Appl. Sci.*, vol. 15, no. 18, Art. no. 10098, Sep. 2025, doi: 10.3390/app151810098.

22. V. Dakić, M. Kovac, and J. Slovinac, "Neural-network-driven custom Kubernetes scheduler for HPC," *Electronics*, 2024.

23. R. Leite, P. Solis, and E. Alchieri, "Benchmarking HCI with VMware/GlusterFS and Docker," 2019, pp. 339–346.