CloudSafe-TP: Hybrid AI-Based SLO-Aware Predictive Thread-Pool Control for Distributed Cloud Backends

Faisal Bahadur, Arooba Akhtar, Nadir Hussain, Abdul Qadir
Department of CS&IT, Hazara University, Mansehra, Pakistan.
*Correspondence: faisal@hu.edu.pk

Citation | Bahadur. F, Akhtar. A, Hussain. N, Qadir. A, "CloudSafe-TP: Hybrid AI-Based SLO-Aware Predictive Thread-Pool Control for Distributed Cloud Backends", IJIST, Vol. 8 Issue. 5 pp 2177-2210, August 2026
DOI | https://doi.org/10.33411/IJIST/1989
Received | July 29, 2026 **Revised |** Aug 18, 2026 **Accepted |** Aug 20, 2026 **Published |** Aug 22, 2026.

Cloud-based distributed services use backend thread-pool controllers to process changing request workloads. Fixed and reactive controllers may allocate too few threads during bursts or maintain unnecessary threads during normal operation. Existing schemes often depend on current measurements, fixed rules, or predictions without calibrated uncertainty, which may cause queue growth and P99 latency SLO violations. This paper presents CloudSafe-TP, a hybrid AI-based and SLO-aware predictive thread-pool control framework for distributed cloud backends. The novelty of CloudSafe-TP lies in combining a sparse Gaussian-process state-space model, conformal calibration, and model predictive control within an independent local controller that evaluates candidate pool sizes before they are applied. At each control interval, CloudSafe-TP predicts P99 latency, queue length, CPU usage, and memory usage over a short horizon, removes candidates that may violate the P99 SLO or resource limits, and selects a lower-cost safe configuration. The framework was evaluated on Oracle Cloud Infrastructure against analytical, neural-network-based, and reinforcement-learning controllers through five independent runs using random seeds 0–4. Under the bursty workload, CloudSafe-TP reduced mean P99 latency by 7.17–23.01%, queue length by 39.74–55.01%, and SLO violation rate by 77.11–89.57% compared with the baselines. Under moderate load, it used 20.70–26.60% fewer threads, reduced CPU usage by 3.90–7.64%, and reduced heap usage by 2.68–23.01%, while maintaining a P99 latency comparable to that of the strongest baseline. Paired t-tests further confirmed statistically significant improvements in P99 latency, queue length, and SLO violation rate under the bursty workload. Overall, CloudSafe-TP provides predictive and uncertainty-aware pool-size adjustment that improves P99 SLO protection during bursty workloads while avoiding unnecessary concurrency and resource usage under moderate load.

Keywords: Cloud-Based Distributed Services; Thread-Pool Control; P99 Latency SLO; Gaussian Process; Conformal Calibration; Model Predictive Control.

Introduction:

Cloud-based distributed services use multiple backend servers to process large and continuously changing request workloads [1][2]. At each backend, a thread pool controls the number of requests that can be executed concurrently [3]. The selected pool size directly affects service performance and resource usage [4]. A small pool may increase queueing delay and tail latency, while an unnecessarily large pool may increase CPU pressure, memory consumption, context switching, and thread-management overhead. Runtime thread-pool adjustment is therefore required to keep each backend close to its useful processing capacity.

The problem becomes more difficult in a heterogeneous distributed environment [5][6]. Backend nodes may differ in processing power, memory capacity, and workload conditions. A pool size suitable for one node may be inefficient for another, while a configuration selected under stable load may become unsuitable during a sudden burst or workload transition. Load balancers and distributed executor services can distribute requests among backend nodes [7], but request routing alone does not ensure that the local thread pool of each backend is correctly sized. Each node still requires a local decision mechanism that can respond to its own queue, latency, workload, and resource conditions.

Recent Studies:

Recent studies have addressed thread and concurrency management through machine-learning-based thread selection [8][9][10][11][12][13], reinforcement learning and online search [14][15][16][17], and latency-aware analytical adaptation [18]. These studies show that runtime measurements can support better parallelism decisions. However, existing approaches are generally based on offline tuning, analytical approximation, point prediction, or a scalar reward. They do not jointly predict the multi-step evolution of P99 latency, queue length, CPU usage, and memory usage while also considering the uncertainty associated with the predicted latency. The detailed characteristics and limitations of these methods are discussed in the next section.

Recent thread-pool controllers use different runtime decision mechanisms. ANN [13] predicts candidate outcomes from current measurements, but its point prediction does not capture future queue, latency, and resource changes over several intervals. PPO [15] learns increase, hold, or decrease actions through a reward function, while SLO protection remains indirect. SORA [18] estimates the goodput–concurrency relationship and considers a latency deadline, but it does not model multi-step backend behaviour or prediction uncertainty during bursts. Hence, these approaches do not jointly provide multi-step prediction, calibrated P99 uncertainty, and explicit filtering of unsafe pool-size candidates.

This limitation is important for tail-latency control. A pool size may appear suitable according to its expected P99 latency but still carry a high prediction error under a burst or workload change. Selecting such a configuration can cause queue buildup and temporary SLO violations. A practical controller should therefore evaluate candidate pool sizes before applying them, consider how the backend state may evolve over a short future horizon, and use a conservative latency bound rather than relying only on a point prediction. It should also avoid unnecessary thread and resource allocation when several configurations are expected to satisfy the latency requirement. These observations lead to the following research question:

Research Question:

How can local thread-pool sizes be adjusted in heterogeneous distributed cloud backends so that P99 latency SLO violations and queue buildup are reduced under changing workloads without excessive CPU, memory, and thread allocation?

Contribution:

To address this question, this paper presents CloudSafe-TP, a hybrid AI-based and SLO-aware predictive thread-pool controller for distributed cloud backends. Each backend node runs an independent CloudSafe-TP instance and uses only its local measurements. The

controller combines a sparse Gaussian-process state-space model, conformal calibration, and model predictive control. The sparse Gaussian-process model predicts the future P99 latency, queue length, CPU usage, and memory usage for bounded candidate pool sizes. Conformal calibration converts the predicted P99 distribution into a conservative upper bound. The model predictive controller then removes candidates that may violate the P99 SLO or resource limits and selects a safe configuration with lower queue, resource, thread-allocation, and adjustment cost.

Flow of Research:

The research follows a sequence from problem analysis to experimental evaluation. First, the prevalent analytical, supervised-learning, reinforcement-learning, and distributed thread-pool approaches are studied to identify their limitations. The local backend state and bounded candidate pool sizes are then defined, followed by the development of the sparse Gaussian-process predictor, conformal safety layer, and SLO-aware model predictive decision. Training traces are collected from the corresponding OCI [19] backend configurations, and separate models are prepared for heterogeneous backend types. The trained controller is deployed locally and evaluated under moderate, CPU-intensive, bursty, and non-stationary workloads. Finally, pool size, P99 latency, CPU usage, heap usage, queue length, and SLO violation rate are analyzed against the existing/baseline controllers.

Research Objectives:

The objectives of this research are to design a local predictive controller that models the short-term backend-state evolution for alternative pool sizes; develop a safety mechanism that uses conformal calibration to identify candidates whose upper P99 latency bound satisfies the configured SLO; formulate a resource-aware model predictive decision that considers queue length, CPU usage, memory usage, pool size, and adjustment stability; and evaluate the proposed controller on Oracle Cloud Infrastructure (OCI) under different workload conditions.

Novelty of the Study:

Existing thread-pool controllers provide different forms of runtime adaptation, but they still have limitations for predictive and SLO-aware control. ANN uses a feedforward supervised predictor to estimate the next-interval behaviour from the current runtime state and a candidate pool size. However, it relies on point prediction and does not model the multi-step changes in queue length, P99 latency, CPU usage, and memory usage. PPO [15] learns runtime pool-size adjustments through a scalar reward. However, its SLO protection is indirect, and candidate configurations are not explicitly checked using predicted latency uncertainty. SORA [18] directly considers a latency deadline and uses an analytical goodput–concurrency relationship. However, it does not provide probabilistic multi-step state prediction or a calibrated upper bound for future P99 latency. These differences are also discussed in the comparative analysis of the Related Work section.

The novelty of CloudSafe-TP is not based on the individual use of Gaussian processes, conformal calibration, or model predictive control. It lies in combining these techniques within a local thread-pool control framework. CloudSafe-TP uses a sparse Gaussian-process state-space model to predict the future behaviour of P99 latency, queue length, CPU usage, and memory usage for alternative pool sizes over a short horizon. It then applies one-sided conformal calibration to obtain a conservative upper bound for P99 latency and removes candidate configurations that may violate the P99 SLO or local resource limits. The model predictive controller then selects a resource-aware configuration from the remaining safe candidates. Thus, CloudSafe-TP evaluates candidate pool sizes using both predicted future system behaviour and calibrated latency uncertainty before applying a configuration.

Related Work: Machine-learning methods have been used for selecting suitable thread counts in parallel and server environments. [8] used program and platform features to predict thread-

pool size for Ballerina services. [9] applied Random Forest to select thread counts for OpenMP regions, while [10] investigated machine-learning-based autotuning for OpenMP task applications. These studies show that learned models can support thread selection, but they mainly target offline tuning, program execution time, or HPC workloads. They do not address continuous local thread-pool control for distributed cloud backends under changing request traffic.

Deep-learning methods have also been applied to parallel-configuration selection. [11] used Graph Neural Networks to identify OpenMP code patterns and select thread count, scheduling policy, and chunk size. A later study extended this approach to power-constrained autotuning [12]. These methods optimize execution time, power, or related batch-computing objectives. Their models operate on programs, kernels, or loops rather than on a long-running request queue whose pool size must be repeatedly adjusted according to P99 latency and resource conditions.

[13] proposed an ANN-based framework for selecting parallel-execution configurations in cloud servers. This work is relevant because it uses a supervised neural predictor to guide parallelism decisions. In the present study, its thread-only form is referred to as ANN. The predictor uses current runtime measurements and a candidate pool size to estimate the next-interval outcome. However, a feedforward point predictor does not represent the multi-step evolution of queue, latency, and resource states. It also does not indicate whether a predicted P99 value may be underestimated under a workload shift. Therefore, a candidate may appear acceptable according to its expected latency even when its prediction uncertainty is high. The absence of calibrated uncertainty is especially important under bursty conditions, where a small prediction error can produce queue buildup and SLO violations.

Runtime adaptation has also been studied through reinforcement learning and search. CAPES [14] uses deep Q-learning for online system tuning. [15] formulate thread-pool adaptation in an actor-based IoT system as a reinforcement-learning problem and use PPO to balance quality of service and resource usage. This controller is referred to as PPO in the present evaluation. PPO learns increase, hold, or decrease actions from runtime feedback through a scalar reward. However, it does not evaluate every candidate pool size against separate P99, CPU, and memory limits before selecting an action. Its decisions also depend on the training experience, reward weights, and available action space. Hence, latency protection is learned indirectly instead of being checked through an explicit candidate-level safety rule.

Search-based schemes such as Hoder [16] and Hebe [17] dynamically tune concurrency and other execution parameters. These schemes respond to runtime feedback and do not require a fixed offline configuration.

However, their main objectives include execution performance, concurrency throttling, or hardware ageing in HPC environments. They generally treat performance through a scalar objective and do not jointly predict request-queue behavior, tail latency, CPU usage, and memory usage for alternative server-side pool sizes.

SORA [18] is more closely related to the present problem because it provides latency-sensitive soft-resource adaptation for microservices. It uses runtime measurements to estimate a goodput–concurrency relationship and selects concurrency under a latency deadline. In this study, its concurrency-control component is referred to as SORA. SORA provides a strong analytical baseline, but it does not model the backend as a probabilistic multi-output state-space process. Its decision is not based on a conformal upper P99 latency bound, and it does not recursively evaluate the predicted queue, latency, CPU, and memory states over a future horizon. Therefore, its analytical concurrency estimate may respond to current measurements without representing the uncertainty associated with sudden bursts or workload transitions.

An intelligent distributed thread pool (DTP) is introduced in [20], where a central load balancer distributes requests among backend Thread Pool System instances. When a backend becomes overloaded, part of its admitted load can be shifted to another available node. This mechanism is useful for distributed request routing and overload recovery. However, load redistribution and local pool-size control address different decisions. Moving requests away from an overloaded backend does not determine which pool size should be maintained at that backend under its remaining workload. The DTP round-robin mechanism is therefore used in the experimental deployment, while its collaborative overload-shifting function is disabled so that the effect of each local pool-size controller can be evaluated separately.

Recent studies have also addressed SLO-aware cloud resource management and microservice autoscaling. GRAF [21] uses graph-neural-network-based prediction for proactive resource autoscaling under latency SLOs. MS-RA [22] applies a requirements-driven self-adaptive mechanism to adjust microservice resources and replicas, while LSRAM [23] uses gradient-descent-based optimization for lightweight SLO-oriented resource allocation. These studies show the importance of predictive and SLO-aware resource management in cloud environments. However, their control decisions mainly focus on CPU or resource allocation, service replicas, and microservice-level capacity. CloudSafe-TP follows a different control objective. It keeps the VM and routing configuration fixed and adjusts only the local backend thread-pool size at runtime. Therefore, these autoscaling approaches are useful for positioning CloudSafe-TP, but they are not directly suitable as experimental baselines for the thread-pool sizing problem considered in this study.

Table 1 summarizes the main strengths and limitations of the reviewed approaches and compares them with CloudSafe-TP.

Research Gap:

As summarized in Table 1, existing approaches cover offline thread selection, supervised prediction, reinforcement-learning-based adaptation, analytical concurrency control, search-based tuning, and distributed load management. However, these methods address only part of the runtime thread-pool control problem considered in this study.

For direct comparison, SORA, ANN, and PPO represent three different runtime control families. SORA uses an analytical goodput–concurrency model, ANN uses supervised point prediction, and PPO learns pool-size actions through a scalar reward. However, none of these methods jointly provides probabilistic multi-step prediction of P99 latency, queue length, CPU usage, and memory usage, together with calibrated P99 uncertainty, explicit rejection of unsafe candidate pool sizes, and resource-aware selection among the remaining safe configurations.

Therefore, a gap remains for a local thread-pool controller that can evaluate alternative pool sizes over a short future horizon, consider uncertainty in tail-latency prediction, and protect the P99 latency SLO without maintaining unnecessary concurrency. CloudSafe-TP addresses this gap by combining sparse Gaussian-process state-space prediction, conformal calibration, and SLO-aware model predictive control.

Material and Methods:

Methodological Overview:

The study follows five connected phases, as summarized in Figure 1. The first phase defines the thread-pool sizing problem, research gap, control objective, and local backend-state variables. The second phase establishes the PoolRunner-based [24] OCI [19] deployment, profiles the backend nodes, and collects separate workload traces for each backend configuration. The third phase develops the CloudSafe-TP controller, trains the sparse Gaussian-process predictor, performs conformal calibration, prepares the baseline controllers, and deploys the frozen models locally. The fourth phase performs runtime monitoring, candidate pool-size generation, multi-step state prediction, SLO and resource safety checking,

minimum-cost candidate selection, pool-size application, and decision logging. The final phase evaluates CloudSafe-TP against SORA, ANN, and PPO under moderate, CPU-intensive, bursty, and non-stationary workloads and analyzes pool size, P99 latency, CPU usage, heap usage, queue length, and SLO-violation rate.

Table 1. Comparative summary of the reviewed thread-pool and concurrency-control approaches.

Approach	Main approach / target	Strength	Key limitation / distinguishing feature
[8]	ML-based thread-pool tuning using program and platform features	Uses learned program characteristics to select suitable thread counts	Mainly supports offline or application-level tuning and does not provide continuous SLO-aware control under changing workloads
[9]	Random-Forest-based thread selection for OpenMP regions	Learns suitable thread counts from program and hardware features	Targets OpenMP/HPC execution rather than server request queues and does not consider P99 SLO or online queue-aware control
[10]	ML-based autotuning for OpenMP task applications	Supports runtime thread-selection decisions for parallel applications	Focuses on application execution performance rather than server-side queue, P99 latency, and resource-aware thread-pool control
[11]	GNN-based OpenMP configuration selection	Selects thread count, scheduling policy, and chunk size from code patterns	Designed for program and kernel optimization rather than changing server workloads and P99 SLO control
[12]	GNN-based autotuning under power constraints	Considers power constraints during configuration selection	Focuses on HPC and batch workloads rather than request queues, tail latency, and runtime server SLOs
[13]	Feedforward ANN prediction for thread configuration	Learns nonlinear relationships between runtime conditions and candidate configurations	Uses next-interval point prediction and does not provide multi-step queue/latency prediction or calibrated uncertainty
[14]	Deep-Q-learning-based online system tuning	Supports adaptive runtime tuning without a fixed analytical model	Depends on RL exploration and a scalar reward and does not explicitly filter candidates using P99 SLO and resource limits
[15]	RL-based runtime thread-pool adaptation	Learns increase, hold, and decrease actions while balancing QoS and resources	SLO protection is indirect through the reward and candidates are not explicitly checked against future P99, CPU, and memory limits
[16]	Online search-based tuning of concurrency and CPU frequency	Dynamically adjusts execution parameters without a fixed offline configuration	Mainly optimizes performance and energy in HPC environments and does not jointly predict server queue, P99 latency, CPU, and memory
[17]	Dynamic concurrency throttling	Learns suitable concurrency during runtime	Focuses on processor ageing and performance trade-offs rather than server-side P99 SLO and queue-aware control
[18]	Analytical goodput–concurrency adaptation under a latency deadline	Uses runtime measurements and directly considers a latency limit	Does not provide probabilistic multi-output prediction, calibrated P99 uncertainty, or multi-step candidate evaluation

[20]	Distributed load balancing and overload shifting among backend thread-pool systems	Supports request redistribution and overload recovery	Focuses on load redistribution rather than local predictive pool-size selection
[21]	GNN-based proactive resource autoscaling for microservices under latency SLOs	Predicts future resource demand and adjusts resources proactively	Works at the microservice resource-autoscaling level rather than local backend thread-pool sizing
[22]	Requirements-driven self-adaptive microservice autoscaling	Adjusts resources and replicas according to runtime requirements	Mainly controls microservice replicas and resources rather than local thread-pool size
[23]	Lightweight SLO-aware microservice resource allocation using gradient-descent optimization	Provides lightweight resource adjustment while considering SLO requirements	Focuses on microservice resource allocation and autoscaling rather than runtime backend thread-pool control
CloudSafe-TP	Sparse-GP state prediction + conformal calibration + MPC	Provides multi-step prediction of P99 latency, queue, CPU, and memory with explicit SLO and resource safety checking	Combines probabilistic prediction, calibrated P99 safety checking, and resource-aware candidate selection within a local thread-pool controller

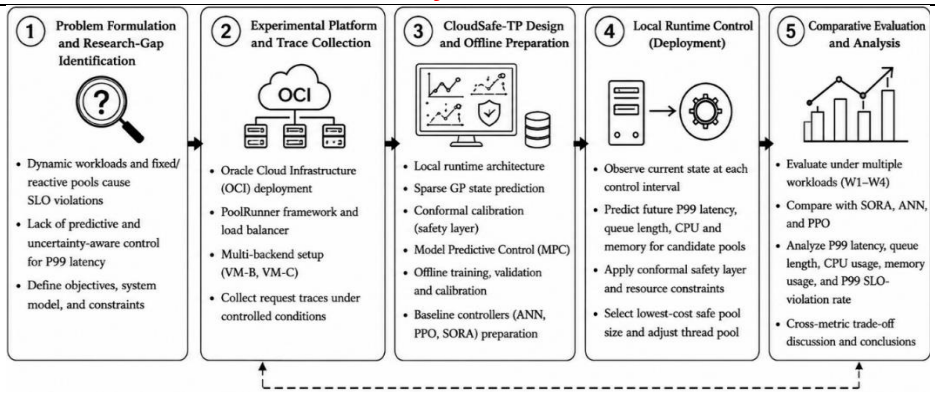


Figure 1. Methodological flow of CloudSafe-TP development, deployment, and evaluation.

Experimental Platform and Distributed Deployment:

We used PoolRunner [24] thread-pool system testing framework in a distributed environment that we deployed on Oracle Cloud Infrastructure (OCI) [19]. The client tier of the PoolRunner generates the workload, while its server tier is deployed on VM [25]. The server tier of PoolRunner is embedded with a load balancer that forwards requests to available VM instances equipped with the specific scheme under test**.** The OCI deployment contains one load-balancer VM and two backend VMs. Each backend runs an independent Thread Pool System and its own local instance of the scheme being evaluated. Therefore, CloudSafe-TP, SORA, ANN, and PPO are tested separately under the same deployment and workload conditions.

The PoolRunner client runs on a local Intel Core i7 machine with 16 GB RAM, Windows 11, and Java/OpenJDK 17. The OCI deployment uses three Ampere A1 VM.Standard.A1.Flex virtual machines with Ubuntu 22.04 LTS and Java/OpenJDK 17 [25][26]. VM-A hosts the load balancer and is configured with 16 OCPUs and 16 GB RAM. VM-B represents the resource-constrained backend and contains 2 OCPUs and 4 GB RAM. VM-C represents the higher-capacity backend and contains 16 OCPUs and 16 GB RAM. A separate Thread Pool System (scheme under test) runs on VM-B and VM-C.

The load balancer follows the round-robin scheduling mechanism used in DTP [20]. In the original DTP design, the scheduler can receive overload information from a weak backend and transfer part of its load to a stable backend. However, this collaborative overload-shifting function is disabled in the present evaluation. The load balancer therefore distributes requests through fixed round robin so that every tested controller receives the same workload share. This avoids mixing the effect of backend thread-pool tuning with the effect of load redistribution.

The local PoolRunner client sends the workload to the public IP address of VM-A. The load balancer then forwards the requests to VM-B and VM-C through their private IP addresses within the same OCI region and virtual cloud network [27]. No artificial delay, packet loss, bandwidth limitation, VPN, or WAN emulation is applied. The same request path, monitoring interval, workload pattern, and logging procedure are used in every experimental run.

VM-B is selected as the main backend for detailed evaluation because it has considerably lower processing and memory resources than VM-C. Therefore, it is more sensitive to unsuitable pool-size decisions, queue growth, resource pressure, and P99 latency SLO violations. The fixed round-robin scheduler distributes an equal share of requests to both backends. However, the same workload creates a more demanding condition for VM-B due to its limited resources. VM-C remains active throughout the experiments to preserve the heterogeneous distributed setup, while the detailed analysis is limited to VM-B to avoid combining results from machines having different capacities.

The trainable schemes use data collected from the same OCI VM types and backend configurations. However, their training workloads are kept separate from the final evaluation workloads. ANN [13], PPO [15], and CloudSafe-TP are prepared through their respective training procedures, whereas SORA develops its analytical model from runtime observations. During final evaluation, only the thread-pool tuning scheme is changed. The hardware, load balancer, workload generation, pool-size limits, and monitoring settings remain unchanged for all schemes. Each controller is evaluated through five independent 40-minute runs using fixed random seeds 0, 1, 2, 3, and 4.

For its integration with the evaluation framework, SORA [18] uses runtime measurements to estimate the relationship between goodput and concurrency. It then selects a pool size that operates within the configured latency limit.

ANN [13] is implemented as a thread-only ANN predictor that estimates next-interval P99 latency, CPU usage, and memory usage for candidate pool sizes. PPO [15] uses a PPO agent that observes the local backend state and selects a decrease, hold, or increase action. The trained ANN and PPO models are fixed before the final evaluation, while SORA updates its analytical model during runtime.

Design and Architecture of Cloudsafe-Tp:

Figure 2 shows the detailed design and architecture of the proposed CloudSafe-TP controller. In the distributed cloud environment, every backend node runs its own local Thread Pool System and an independent instance of CloudSafe-TP. There is no central CloudSafe-TP controller. Each local controller monitors only its associated backend, predicts its future behaviour, and adjusts its thread-pool size through the local thread-pool adapter. Therefore, failure or disconnection of one backend controller does not affect the operation of the remaining backend nodes. A backend Thread Pool System contains a request queue, worker thread pool, application service, response queue, runtime monitor, and thread-pool adapter. The runtime monitor collects local measurements after every monitoring interval Δ. These measurements are provided to CloudSafe-TP, which consists of four main components: the local state interface, sparse Gaussian-process state predictor, conformal safety layer, and model predictive decision module. The selected pool size is returned to the thread-pool adapter and applied during the next interval.

Local Runtime State and Candidate Pool Sizes:

Let $B_i$represent the ith backend node in the distributed cloud service. The local runtime monitor collects request rate, average service time, queue length, P99 response time, CPU usage, memory usage, and current pool size. The state of $B_i$at monitoring interval $t$is represented as

$$x_i(t) = \left[\lambda_i(t), s_i(t), q_i(t), L_i^{99}(t) U_i^{CPU}(t) U_i^{MEM}(t) p_i(t)\right]^T, \quad (1)$$

where $\lambda_i(t)$is the admitted request rate, $s_i(t)$is the average service time, $q_i(t)$is the number of waiting requests, $L_i^{99}(t)$is the observed P99 processing time, and $p_i(t)$is the current pool size.

The response-time measurement starts when a request enters the local thread-pool system and ends when its processing is completed. It includes queue waiting time, service execution time, and thread-management delay. Network and client-side delays are not included because the local controller cannot directly control them.

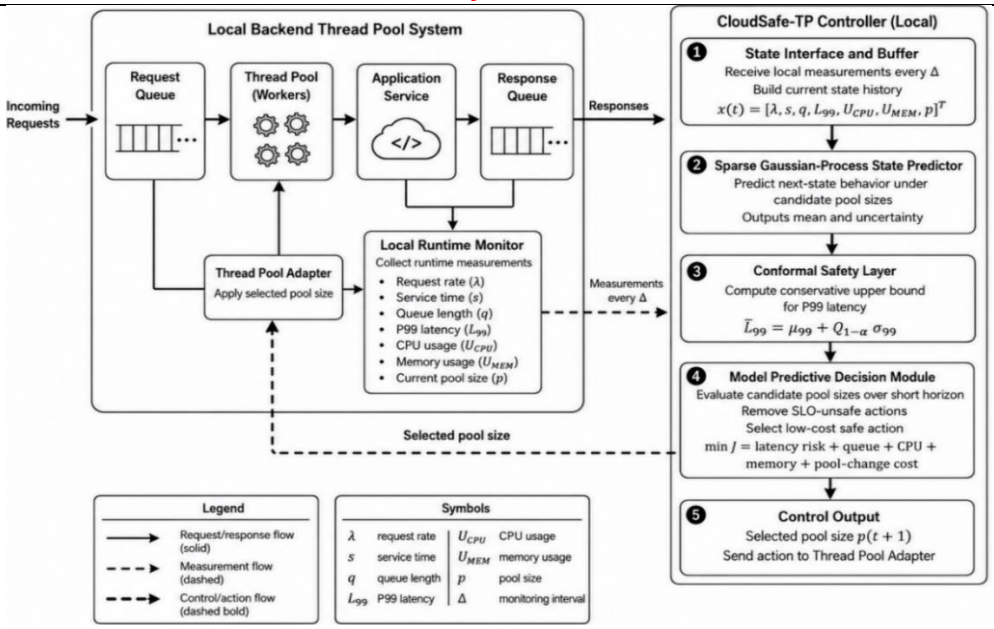


Figure 2. Architecture of the proposed CloudSafe-TP controller for a local backend Thread Pool System.

CloudSafe-TP does not evaluate every possible integer pool size. A bounded candidate set is generated around the current pool size:

$$\mathcal{P}_i(t) = \left\{ \text{clip}(p_i(t) + r\delta_i p_i^{min} p_i^{max}) \mid r \in \{-2, -1, 0, 1, 2\} \right\}, \quad (2)$$

where δ_i is the pool-adjustment step and p_i^{min} and p_i^{max} are the configured local limits. The candidate set therefore contains decrease, hold, and increase options. These limits are selected according to the CPU cores, memory capacity, and executor restrictions of the backend node.

The metrics are normalized using the limits observed during training or configured for the backend. This prevents large-valued metrics from dominating the remaining measurements and enables the same control process to work across cloud nodes having different capacities.

Sparse Gaussian-Process State Predictor:

CloudSafe-TP models the local thread-pool system as a dynamic process. The next queue length, P99 latency, CPU usage, and memory usage depend on the current backend state and the candidate pool size. For backend B_i, this relationship is written as:

$$y_i(t+1) = f_i(x_i(t), p_i^c) + \epsilon_i(t), \quad (3)$$

where $p_i^c \in \mathcal{P}_i(t)$, $\epsilon_i(t)$ represents workload and measurement variation, and

$$y_i(t+1) = \left[L_i^{99}(t+1), q_i(t+1) U_i^{CPU}(t+1) U_i^{MEM}(t+1) \right]^T.$$

A sparse Gaussian-process model learns the nonlinear transition function $f_i(\cdot)$. For every candidate pool size, it produces a predictive distribution:

$$y_i(t+1 \mid p_i^c) \sim \mathcal{N}\left(\mu_i(t+1 p_i^c) \Sigma_i(t+1 p_i^c)\right). \quad (4)$$

The predictive mean shows the expected next state, while the variance shows the uncertainty of prediction. The uncertainty remains low for familiar workload conditions. However, it increases when the current workload differs from the training data, and such candidates are treated more conservatively.

A standard Gaussian process becomes costly when the training data increases. Therefore, CloudSafe-TP uses a sparse model with a limited number of inducing points. These points represent the main operating conditions observed in the training traces. Hence, the model reduces prediction cost while maintaining nonlinear prediction and uncertainty

estimation. The predictor is applied recursively over a short horizon H. To keep runtime computation manageable, CloudSafe-TP uses a move-blocking strategy: each candidate pool size is assumed to remain fixed during the prediction horizon. The controller therefore evaluates a small number of candidate sizes rather than a large number of possible action sequences.

Conformal Safety Layer:

Gaussian-process variance provides useful uncertainty information, but the predicted interval may still underestimate future P99 latency. CloudSafe-TP therefore applies one-sided conformal calibration before a candidate is considered safe.

A separate calibration dataset is processed through the trained sparse Gaussian-process model. For calibration observation j, the normalized prediction error is

$$a_j = \frac{L_j^{99} - \mu_j^{99}}{\max(\sigma_j^{99}, \varepsilon)}, \qquad (5)$$

where L_j^{99} is the observed P99 response time, μ_j^{99} and σ_j^{99} are the predicted mean and standard deviation, and ε is a small value used to avoid division by zero.

Let $Q_{1-\alpha}$ be the empirical $(1-\alpha)$-quantile of the calibration scores. The conformalized upper bound for candidate pool size p_i^c is computed as

$$\bar{L}_i^{99}(t, +kp_i^c) = \mu_i^{99}(t, +kp_i^c) + Q_{1-\alpha}\,\sigma_i^{99}(t, +kp_i^c), \qquad (6)$$

where $k = 1, \ldots, H$. The value α represents the allowed prediction risk. For example, $\alpha = 0.01$ provides a conservative bound based on a one-percent risk level.

The decision module uses this upper bound instead of the P99 point prediction. A candidate may therefore be rejected even when its expected latency is below the SLO if its uncertainty is high. This prevents the local controller from selecting aggressive pool sizes under unfamiliar or rapidly changing workloads.

Each backend maintains its own calibration values. Therefore, differences in processor capacity, memory size, workload type, or operating-system behaviour are handled locally rather than through a common central safety margin.

SLO-Aware Model Predictive Decision:

Let L_{SLO} denote the configured P99 response-time SLO. For every candidate p_i^c, the model predictive decision module predicts system behaviour over horizon H. A candidate is placed in the safe set when its conformalized P99 bound remains within the SLO and its predicted CPU and memory usage remain within the configured local limits:

$$\mathcal{S}_i(t) = \left\{ p_i^c \,\middle|\, p_i^c \in \mathcal{P}_i(t) \;\; \begin{array}{c} \bar{L}_i^{99}(t, +kp_i^c) \le L_{SLO}, \\ \widehat{U}_i^{CPU}(t, +kp_i^c) \le U_i^{CPU,max}, \\ \widehat{U}_i^{MEM}(t, +kp_i^c) \le U_i^{MEM,max}, \\ k = 1, \ldots, H \end{array} \right\}. \qquad (7)$$

For each safe candidate, CloudSafe-TP calculates a resource-aware control cost:

$$J_i(p_i^c) = \sum_{k=1}^{H} \left[w_q \tilde{q}_i + w_c \tilde{U}_i^{CPU} + w_m \tilde{U}_i^{MEM} + w_p \tilde{p}_i^c \right] + w_\Delta |\tilde{p}_i^c - \tilde{p}_i(t)|. \qquad (8)$$

The first four terms represent predicted queue length, CPU usage, memory usage, and thread allocation. The final term represents the cost of changing the pool size. It reduces unnecessary oscillation when two nearby configurations provide similar performance. The tilde represents normalized values, while the weights determine the importance of each control objective.

Controller-parameter selection: The prediction horizon was set to H=5 control intervals. Since CloudSafe-TP updates its measurements and decisions every second, the controller

predicts system behavior for the next five seconds. This horizon was selected to identify developing queue and latency pressure without increasing recursive prediction error and runtime cost. CloudSafe-TP follows a receding-horizon process, where only the first pool-size decision is applied before new measurements are collected. Therefore, a much longer horizon was not required. The conformal risk level was set to $\alpha=0.01$, which gives a nominal one-sided confidence level of 99%. This value was selected to reduce the chance of underestimating future P99 latency and accepting an unsafe pool configuration. A much smaller value could make the controller unnecessarily conservative. The control weights were set to $w_q = .35$, w_c=.20, w_m=.15, w_p=.20, w_Δ=0.10. Queue length received the highest weight because queue buildup can quickly increase tail latency during bursty workloads. CPU usage and pool size received moderate weights to balance resource usage and required concurrency. Memory received a lower weight because heap usage changed less across feasible configurations. The pool-change penalty was also kept low so that unnecessary pool oscillations could be reduced without delaying required adjustments. P99 latency is not included as a weighted objective because it is directly controlled through the conformal SLO-safety constraint.

When the safe set is not empty, the candidate with the lowest cost is selected. When no candidate satisfies all constraints, the controller selects the candidate with the lowest conformalized P99 latency bound. Resource usage and pool-change cost are used as tie-breaking factors. This fallback does not guarantee SLO satisfaction during unavoidable overload, but it selects the locally safest available configuration.

The selected pool size is applied only for the next monitoring interval. New measurements are then collected and the complete prediction and decision process is repeated. This receding-horizon operation allows CloudSafe-TP to correct earlier predictions and respond to changes in workload or service time.

Runtime Control Algorithm:

Algorithm 1 presents one execution cycle of the local CloudSafe-TP controller. The backend scheduler invokes the algorithm once after every monitoring interval Δ. Each call receives the latest measurements, evaluates the candidate pool sizes, and returns the pool size to be used during interval $t + 1$.

Algorithm 1: CloudSafe-TP Runtime Control
Input: Current backend measurements at interval t; trained sparse Gaussian-process model f_i; conformal quantile Q_(1-α); prediction horizon H; latency SLO L^SLO; CPU and memory limits; pool-size limits p_i^minand p_i^max; adjustment step δ_i; and cost weights.
Output: Pool size p_i (t+1)for the next monitoring interval.
Receive the latest request rate, service time, queue length, P99 latency, CPU usage, memory usage, and current pool size from backend B_i.
If any required measurement is missing or outdated, retain p_i (t), record that no new decision was issued, and return.
Form and normalize the local runtime state x_i (t)using Equation (1).
Generate the bounded candidate set P_i (t)using Equation (2).
Initialize an empty safe set and set the current pool size as the initial fallback candidate.
For each candidate p_i^c∈P_i (t), recursively predict P99 latency, queue length, CPU usage, and memory usage over horizon Husing Equations (3) and (4).
At every prediction step, compute the conformal upper P99 latency bound using Equation (6).
Mark the candidate unsafe if its conformal upper P99 latency bound, predicted CPU usage, or predicted memory usage exceeds the limits defined in Equation (7).
Calculate the candidate control cost using Equation (8).
Add every safe candidate and its cost to S_i (t).

Update the fallback candidate when a candidate has a lower maximum conformal upper P99 latency bound; use the lower control cost to resolve a tie.
Select the minimum-cost candidate from S_i (t). If the safe set is empty, select the fallback candidate.
Apply the selected pool size through the local thread-pool adapter and set it as p_i (t+1).
Record the runtime state, candidate predictions, conformal bounds, costs, selected pool size, and decision status, and return p_i (t+1).

The algorithm is executed independently by every backend node. A candidate is considered safe only when its conformal upper P99 latency bound and predicted resource usage remain within the configured limits throughout the prediction horizon. If a controller temporarily fails, its associated backend continues operating with its last selected pool size, while other backend nodes remain unaffected.

Computational Complexity and Runtime Overhead:

Let K represent the number of candidate pool sizes, H the prediction horizon, and M the number of inducing points used by the sparse Gaussian-process model. CloudSafe-TP does not search the complete pool-size range. It evaluates only bounded decrease, hold, and increase candidates. Therefore, K remains small and does not depend on the maximum pool-size limit. For each candidate, the sparse GP predicts P99 latency, queue length, CPU usage, and memory usage over H future steps.

For a sparse GP with M inducing points, the main cost of calculating the predictive mean and variance for one state transition is bounded by $O(M^2)$, assuming the required training-dependent matrix terms are precomputed. Hence, the dominant complexity of one CloudSafe-TP decision cycle is:

$$O(KHM^2)$$

The conformal upper-bound calculation, safety checking, control-cost calculation, and minimum-cost candidate selection require approximately:

$$O(KH)$$

Therefore, sparse-GP prediction remains the dominant computational cost. Since K contains only local decrease, hold, and increase options and H=5, the online search remains limited. Sparse inducing-point prediction also avoids the higher cost of using a full Gaussian process, while move-blocking avoids evaluating all possible pool-size action sequences. These choices keep the runtime computation suitable for the one-second control interval used in this study. Only the first selected action is applied, after which new measurements are collected and the decision process is repeated.

Offline Training, Calibration, Baseline Preparation, and Local Deployment:

three representative schemes are selected in comparison with CloudSafe-TP. SORA [18] is included as a strong latency-aware analytical controller for cloud-service concurrency management. ANN [13] is selected as an ANN-based predictive method for cloud-server parallelism, while PPO [15] is included as a reinforcement-learning-based runtime thread-pool controller. Together, these schemes represent analytical control, supervised neural prediction, and reinforcement learning, providing a balanced comparison with the proposed probabilistic and SLO-aware predictive approach.

Backend Profiling and Trace Collection:

All trainable controllers were prepared using the OCI environment described earlier. Training traces were collected separately for each backend type so that measurements from heterogeneous nodes were not mixed. During trace collection, VM-A directed the workload to the backend being trained, while the the fixed round-robin configuration was used during final evaluation.

Before trace collection, short profiling runs were conducted to estimate the sustainable request rate R_i of backend B_i. The profiling data were used only to determine workload levels

and were not included in the training dataset. Each training rollout lasted 40 minutes and contained four 10-minute operating regimes: stationary moderate load at approximately $0.5R_i$; stationary CPU-bound load at approximately $0.9R_i$; bursty load with a baseline of approximately $0.7R_i$ and one-second spikes up to $1.2R_i$ every 10 seconds; and a ramp-and-drop load using approximately $0.4R_i$, $0.7R_i$, R_i, and $0.7R_i$ for 120, 240, 60, and 180 seconds, respectively. The ramp-and-drop regime contained 30% CPU-bound and 70% I/O-bound requests. After profiling, these workload levels remained fixed for all schemes. Separate runs and random seeds were used for training and final evaluation.

Each collected sample contained the current backend state, the pool size applied during the following interval, and the resulting P99 latency, queue length, CPU usage, and memory usage. Complete rollouts were assigned to training, validation, and conformal-calibration sets. Consecutive observations from the same rollout were not randomly distributed among these sets because such division could introduce data leakage.

CloudSafe-TP Training, Validation, and Conformal Calibration:

CloudSafe-TP used five independent 40-minute rollouts for each backend, providing 200 minutes of trace data and approximately 12,000 one-second observations. Three complete rollouts were used to train the sparse Gaussian-process predictor, one rollout was used for validation, and one was reserved for conformal calibration. The training set was used to learn the model parameters and inducing points, while the validation set supported parameter selection and early stopping. The calibration rollout was used only to calculate the conformal quantile in Equation (6). The trained model parameters and calibration value were frozen before final evaluation.

ANN and PPO Training:

The ANN baseline was trained once using the same training and validation traces used for the CloudSafe-TP predictor. It learned to estimate next-interval P99 latency, CPU usage, and memory usage from the current backend measurements and a candidate pool size. The trained ANN model was frozen before final testing.

PPO was trained separately because its policy learns through direct interaction with the backend. It used ten 40-minute rollouts, providing 400 minutes of interaction and approximately 24,000 one-second control decisions. Nine rollouts were used for policy learning, while one rollout was used to select the final checkpoint. The selected PPO policy was then frozen before evaluation.

SORA Online Preparation:

SORA [18] does not require offline training. It estimates the goodput–concurrency relationship from runtime measurements and updates its analytical model during each evaluation run. Therefore, SORA was initialized under the same deployment and monitoring conditions as the trainable controllers but constructed/updated its analytical control model online.

Model Freezing and Local Deployment:

After offline preparation, the CloudSafe-TP model and its conformal-calibration value were installed with the corresponding local backend controller. Each backend used its own measurements, pool-size limits, model parameters, and calibration value. Models prepared for different backend configurations were not exchanged between nodes. ANN and PPO similarly used their frozen models during final evaluation, whereas SORA continued updating its analytical model online. No runtime communication among CloudSafe-TP controllers was required.

Failure Handling and Measurement Logging:

Each local controller operated independently. If a controller temporarily failed, its associated backend continued operating with the last selected safe pool size, while the

remaining backend controllers continued without interruption. When a required runtime measurement was missing or stale, the controller retained the current pool size and did not issue a new adjustment decision. Runtime states, candidate predictions, conformal upper bounds, control costs, selected pool sizes, decision status, and observed outcomes were stored locally for evaluation and possible later model updating.

Evaluation Workload:

Each controller is evaluated through five independent 40-minute runs using fixed seeds 0, 1, 2, 3, and 4. All workload-generation experiments are implemented in Java using java.util.Random. Before each run, the corresponding seed is explicitly supplied to the random-number generator to generate the stochastic request-arrival sequence. Each 40-minute run contains four consecutive 10-minute workload periods.

Figure 3 shows one representative workload pattern generated using seed 0. The remaining seeds follow the same workload configuration but generate different stochastic request-arrival sequences. Request rate and average service time are plotted against time using separate vertical axes. Request arrivals are generated every second using Poisson distributions, and the stated rates represent the total client workload before fixed round-robin distribution.

W1 uses a stable moderate load of 500 requests/s with an average I/O-bound service time of approximately 200 ms. W2 applies a high CPU-bound load of 800 requests/s with an average service time of approximately 20 ms. W3 uses a baseline rate of 1,000 requests/s with one-second spikes between 1,500 and 1,700 requests/s every tenth second. Its service time is approximately 20 ms for CPU-bound requests and 200 ms for I/O-bound requests. W4 follows a ramp-and-drop pattern of 300, 700, 1,100, and 700 requests/s for 120, 240, 60, and 180 seconds, respectively. It contains 30% CPU-bound and 70% I/O-bound requests, with average service times of approximately 20 ms and 200 ms, respectively. The configured server-side latency requirement is P99 ≤ 2 s.

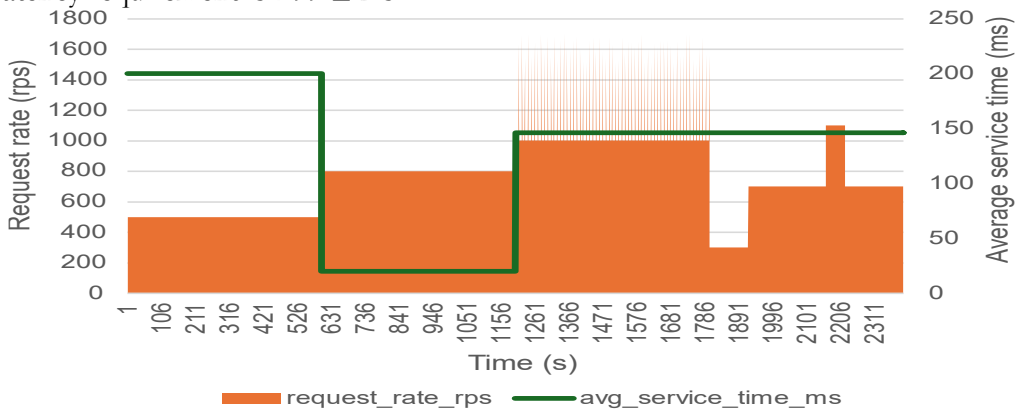


Figure 3. Request-rate and average-service-time patterns during the 40-minute evaluation workload.

Result and discussion:

Pool-Size Comparative Analysis:

This subsection compares the pool sizes selected by all evaluated schemes across W1–W4. Each controller was evaluated through five independent 40-minute runs using different random seeds. The phase-level averages were used to calculate the mean, standard deviation (SD), and 95% confidence interval (CI). Table 2 reports these values, while Figure 4 shows the corresponding mean pool sizes with 95% CIs. A smaller pool is not always better because higher concurrency may be required to avoid queue growth and high P99 latency.

Table 2. Mean, standard deviation, and 95% confidence intervals of average thread-pool size for PPO, SORA, ANN, and CloudSafe-TP across W1–W4 over five independent runs.

Scheme	Phase	Mean	SD	95% CI Lower	95% CI Upper

PPO	W1	56.938	4.107	51.839	62.037
PPO	W2	69.392	2.434	66.37	72.414
PPO	W3	96.545	2.769	93.106	99.983
PPO	W4	64.771	2.279	61.942	67.601
SORA	W1	58.355	3.831	53.597	63.112
SORA	W2	70.729	3.682	66.157	75.3
SORA	W3	98.288	3.585	93.837	102.74
SORA	W4	89.02	2.634	85.749	92.29
ANN	W1	61.514	2.786	58.055	64.973
ANN	W2	69.658	1.588	67.687	71.629
ANN	W3	97.385	3.093	93.544	101.225
ANN	W4	63.034	4.821	57.049	69.02
CloudSafe-TP	W1	45.15	1.184	43.68	46.621
CloudSafe-TP	W2	77.456	0.764	76.508	78.405
CloudSafe-TP	W3	109.735	0.957	108.546	110.924
CloudSafe-TP	W4	77.978	0.449	77.42	78.536

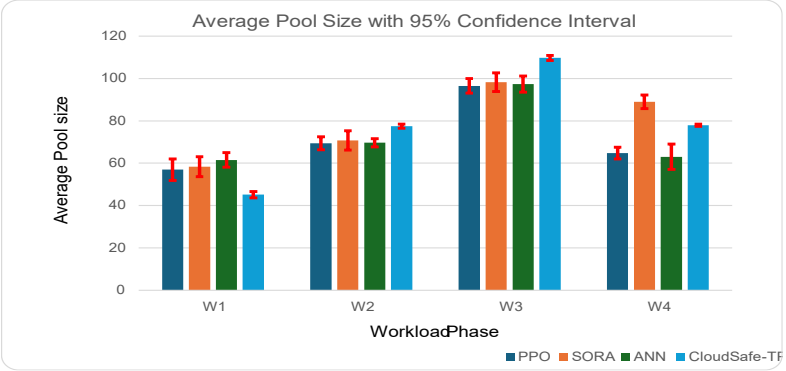


Figure 4. Average thread-pool size of PPO, SORA, ANN, and CloudSafe-TP across W1–W4 with 95% confidence intervals.

During W1, CloudSafe-TP recorded the lowest mean pool size of 45.15 threads, compared with 56.94 for PPO, 58.36 for SORA, and 61.51 for ANN. Thus, it used 20.70%, 22.63%, and 26.60% fewer threads, respectively. Its 95% CI of 43.68–46.62 remained completely below all baseline intervals, while its SD was only 1.18 threads. Hence, the lower W1 allocation remained consistent across the five runs. Under this moderate workload, CloudSafe-TP removed unnecessary concurrency after identifying safe candidate configurations.

During W2, CloudSafe-TP increased its mean pool to 77.46 threads, compared with 69.39 for PPO, 70.73 for SORA, and 69.66 for ANN. It therefore used 11.62%, 9.51%, and 11.19% more threads, respectively. Its 95% CI of 76.51–78.41 remained separated from all baseline intervals, and its SD was only 0.76 threads. This shows that the higher W2 allocation was also consistent across the five runs. The sustained CPU-intensive workload favored greater concurrency, and CloudSafe-TP accepted more threads when predicted queue and latency pressure increased while CPU and memory remained within their limits.

The largest increase relative to the baselines appeared in W3. CloudSafe-TP maintained a mean pool of 109.74 threads, compared with 96.55 for PPO, 98.29 for SORA, and 97.39 for ANN. This corresponds to 13.66%, 11.65%, and 12.68% more threads, respectively. Its 95% CI of 108.55–110.92 did not overlap with any baseline interval, and its SD remained low at 0.96 threads. W3 contains recurring one-second spikes. Therefore, CloudSafe-TP maintained more concurrency when smaller candidates were predicted to cause later queue growth or unsafe conformal P99 bounds.

The lower W3 pools of PPO, SORA, and ANN should not alone be considered more efficient. SORA does not provide probabilistic multi-step prediction, ANN uses next-interval point prediction, and PPO protects the SLO indirectly through its reward. Their pool decisions must therefore be considered together with P99 latency, queue length, CPU usage, and SLO violations.

During W4, CloudSafe-TP reduced its mean pool to 77.98 threads. PPO and ANN reduced their pools to 64.77 and 63.03 threads, while SORA maintained the highest value of 89.02 threads. Thus, CloudSafe-TP used 20.39% more threads than PPO and 23.71% more than ANN, but 12.40% fewer than SORA. Its 95% CI of 77.42–78.54 remained completely separated from the baselines, while its SD was only 0.45 threads. The controller therefore reduced part of the W3 concurrency while keeping enough workers for the changing W4 workload.

Overall, CloudSafe-TP did not always select the smallest or largest pool. It maintained the lowest pool in W1, increased above the baselines in W2 and W3, and reduced concurrency again in W4. Its SD values of 1.18, 0.76, 0.96, and 0.45 threads show that these decisions remained stable across the five random-seed runs. The separated 95% confidence intervals indicate clear differences in the observed mean pool sizes across the repeated experiments across repeated experiments. Hence, CloudSafe-TP reduced unnecessary concurrency under moderate load and increased it when additional processing capacity was required.

P99 Latency Comparative Analysis

This subsection compares the P99 latency of PPO, SORA, ANN, and CloudSafe-TP across workload phases W1–W4. Each scheme was evaluated through five independent 40-minute runs using different random seeds for the Poisson request-arrival process. Every run contained the same four consecutive 10-minute workload phases, while the exact request arrivals changed according to the seed. The five phase-level P99 averages were used to calculate the mean, standard deviation (SD), and 95% confidence interval (CI). Table 3 reports these values, while Figure 5 shows the mean P99 latency with 95% confidence intervals. P99 latency represents the 99th-percentile response time and therefore captures tail-latency behavior. The configured server-side P99 SLO was 2,000 ms. The measured response time includes queue waiting, service execution, and thread-management delay within the backend thread-pool system.

Table 3. Mean, standard deviation, and 95% confidence intervals of average P99 latency for PPO, SORA, ANN, and CloudSafe-TP across W1–W4 over five independent runs.

Scheme	Phase	Mean P99 (ms)	SD (ms)	95% CI Lower (ms)	95% CI Upper (ms)
PPO	W1	296.498	6.945	287.875	305.121
PPO	W2	264.631	5.203	258.171	271.092
PPO	W3	2409.447	26.121	2377.013	2441.881
PPO	W4	524.152	8.313	513.83	534.474
SORA	W1	238.489	4.636	232.733	244.245
SORA	W2	249.099	4.912	243	255.199
SORA	W3	1998.476	28.088	1963.6	2033.353
SORA	W4	1232.329	24.414	1202.016	1262.643
ANN	W1	353.384	6.451	345.374	361.394
ANN	W2	255.419	4.385	249.974	260.864
ANN	W3	2359.121	25.32	2327.682	2390.56
ANN	W4	460.798	6.503	452.723	468.873
CloudSafe-TP	W1	238.796	2.722	235.416	242.176
CloudSafe-TP	W2	241.592	2.334	238.694	244.489

CloudSafe-TP	W3	1855.132	17.512	1833.388	1876.876
CloudSafe-TP	W4	456.53	3.477	452.212	460.847

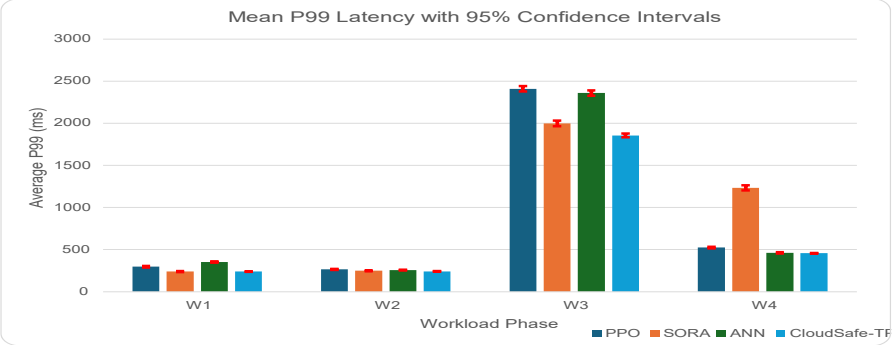


Figure 5. Mean P99 latency of PPO, SORA, ANN, and CloudSafe-TP across W1–W4 with 95% confidence intervals.

During W1, SORA and CloudSafe-TP produced almost the same P99 latency. SORA recorded the lowest mean of 238.49 ms, while CloudSafe-TP recorded 238.80 ms. The difference was only about 0.31 ms and is practically negligible. PPO recorded 296.50 ms, while ANN produced the highest W1 P99 latency of 353.38 ms. Thus, SORA and CloudSafe-TP provided considerably lower tail latency than PPO and ANN under the stable moderate workload.

The confidence intervals provide a clearer interpretation. CloudSafe-TP recorded a 95% CI of 235.42–242.18 ms, while SORA recorded 232.73–244.25 ms. These intervals overlap considerably. Therefore, the small difference between their mean values should not be considered a meaningful advantage for either scheme. In contrast, PPO and ANN recorded intervals of 287.88–305.12 ms and 345.37–361.39 ms, respectively, which remained clearly above CloudSafe-TP and SORA. Hence, the five-run results show that CloudSafe-TP and SORA operated in the same low-latency region during W1, while PPO and ANN produced higher P99 latency.

CloudSafe-TP also recorded the lowest SD of 2.72 ms, compared with 4.64 ms for SORA, 6.95 ms for PPO, and 6.45 ms for ANN. Although its mean P99 latency was almost identical to SORA, the smaller SD indicates that its W1 latency varied less across the five random-seed runs across the five random-seed runs. Therefore, CloudSafe-TP provided more repeatable latency behaviour under different Poisson arrival sequences.

The close W1 performance of CloudSafe-TP and SORA is consistent with their control mechanisms. W1 is a stable moderate workload where recent measurements provide a useful indication of the next operating condition. SORA estimates the goodput-concurrency relationship from runtime observations and directly considers a latency deadline. Therefore, under stable conditions, its analytical model can select suitable concurrency without requiring multi-step workload prediction. CloudSafe-TP also performs well, but the benefit of multi-step prediction and conformal uncertainty checking becomes less important when workload conditions are stable. This explains the almost identical P99 latency of both schemes during W1.

ANN and PPO follow different decision mechanisms. ANN predicts next-interval P99 latency, CPU usage, and memory usage for candidate pool sizes. However, it uses point prediction and does not provide calibrated uncertainty or multi-step prediction of queue and latency. PPO adjusts the pool through a learned reward rather than explicit candidate-level P99 safety checking. These limitations did not result in SLO violations in W1, but both schemes still produced higher P99 latency than CloudSafe-TP and SORA.

During W2, the P99 values of all schemes became closer. CloudSafe-TP recorded the lowest mean P99 latency of 241.59 ms, followed by SORA at 249.10 ms, ANN at 255.42 ms,

and PPO at 264.63 ms. Thus, CloudSafe-TP reduced mean P99 latency by approximately 3.01% compared with SORA, 5.41% compared with ANN, and 8.71% compared with PPO.

However, the confidence intervals should also be considered. CloudSafe-TP recorded a 95% CI of 238.69–244.49 ms, while SORA recorded 243.00–255.20 ms. These intervals slightly overlap between approximately 243 and 244.5 ms. Therefore, although CloudSafe-TP recorded the lower mean, the difference from SORA was not strongly separated by the 95% confidence intervals. ANN recorded a CI of 249.97–260.86 ms, while PPO recorded 258.17–271.09 ms. Their intervals remained progressively above CloudSafe-TP, with PPO showing the clearest separation.

CloudSafe-TP again recorded the lowest SD of 2.33 ms, compared with 4.91 ms for SORA, 4.39 ms for ANN, and 5.20 ms for PPO. Hence, its W2 P99 latency remained more stable across the five runs.

W2 applies a sustained CPU-intensive workload rather than rapidly changing traffic. Therefore, the four schemes remained relatively close. SORA can update its goodput-concurrency relationship from a stable operating condition, while ANN can make next-interval predictions with less difficulty than under recurring bursts. PPO also responds to the increased resource pressure through its learned policy. CloudSafe-TP achieved a small advantage because it evaluates candidate pools using predicted future P99 latency, queue length, CPU usage, and memory usage and removes candidates that exceed the predicted safety limits. Therefore, the W2 advantage was not very large, but CloudSafe-TP maintained a low P99 latency with smaller variation across the five runs.

The clearest difference appeared during W3. CloudSafe-TP recorded a mean P99 latency of 1,855.13 ms, compared with 1,998.48 ms for SORA, 2,359.12 ms for ANN, and 2,409.45 ms for PPO. Thus, CloudSafe-TP reduced mean P99 latency by approximately 7.17% compared with SORA, 21.36% compared with ANN, and 23.01% compared with PPO.

This result is important because W3 is the phase where the 2,000 ms SLO becomes critical. CloudSafe-TP remained approximately 144.87 ms below the SLO, while SORA remained only about 1.52 ms below it. ANN exceeded the SLO by approximately 359.12 ms, while PPO exceeded it by approximately 409.45 ms.

The confidence intervals further show the W3 difference. CloudSafe-TP recorded a 95% CI of 1,833.39–1,876.88 ms. SORA recorded 1,963.60–2,033.35 ms, ANN 2,327.68–2,390.56 ms, and PPO 2,377.01–2,441.88 ms. The CloudSafe-TP interval did not overlap with any baseline interval. Therefore, the W3 difference remained clear even after considering variation among the five random-seed runs.

CloudSafe-TP also recorded the lowest W3 SD of 17.51 ms. PPO, SORA, and ANN recorded SDs of 26.12, 28.09, and 25.32 ms, respectively. Higher variation is expected during W3 because of the recurring workload spikes. However, CloudSafe-TP still maintained the lowest SD. Thus, it achieved both lower mean P99 latency and more consistent behaviour across the five workload realizations.

The W3 result can be explained from the workload and controller mechanisms. W3 contains recurring one-second spikes. Requests accumulated during one spike may still remain in the queue when the next spike arrives. A controller that mainly responds to the current or next-interval state may therefore react after part of the backlog has already developed. CloudSafe-TP evaluates every candidate pool size over several future intervals. Its sparse Gaussian-process model predicts P99 latency, queue length, CPU usage, and memory usage, while the conformal layer increases the P99 safety bound when prediction uncertainty becomes higher. Therefore, a candidate can be rejected even when its immediate P99 prediction appears acceptable if its later predicted state or uncertainty-adjusted P99 becomes unsafe.

SORA performed considerably better than ANN and PPO during W3 because it directly considers latency while adjusting concurrency. However, it does not recursively predict the multi-output system state over a future horizon or calculate a calibrated upper P99 bound. Its mean P99 remained very close to the SLO, while its 95% confidence interval extended from below to above 2,000 ms. This indicates that the phase-average P99 has substantial variation relative to the 2,000 ms SLO across repeated runs. In contrast, the complete CloudSafe-TP confidence interval remained below the SLO.

ANN and PPO showed a larger difference from CloudSafe-TP. ANN uses a frozen next-interval point predictor and does not explicitly represent the risk that its prediction may become optimistic during recurring spikes. PPO adjusts the pool through learned decrease, hold, and increase actions and balances different objectives through its reward. However, it does not evaluate every candidate against a predicted future P99 safety condition. Therefore, both schemes remained more exposed to backlog accumulation during repeated spikes, which is consistent with their higher W3 P99 latency.

During W4, the ranking changed again. CloudSafe-TP recorded a mean P99 latency of 456.53 ms, while ANN recorded 460.80 ms, PPO 524.15 ms, and SORA 1,232.33 ms. Thus, CloudSafe-TP reduced P99 latency by approximately 0.93% compared with ANN, 12.90% compared with PPO, and 62.95% compared with SORA.

The difference between CloudSafe-TP and ANN should be interpreted carefully. CloudSafe-TP recorded a 95% CI of 452.21–460.85 ms, while ANN recorded 452.72–468.87 ms. These intervals overlap considerably. Therefore, the mean difference of approximately 4.27 ms should not be considered a clear statistical separation. Both schemes operated in almost the same P99 region during W4. However, CloudSafe-TP recorded a smaller SD of 3.48 ms compared with 6.50 ms for ANN, showing more repeatable latency across the five runs.

The differences from PPO and SORA were much clearer. PPO recorded a 95% CI of 513.83–534.47 ms, which remained completely above the CloudSafe-TP interval. SORA recorded 1,202.02–1,262.64 ms, which was far above the other schemes. SORA also produced the highest W4 SD of 24.41 ms, compared with 8.31 ms for PPO, 6.50 ms for ANN, and 3.48 ms for CloudSafe-TP. Thus, SORA produced both higher mean P99 latency and greater run-to-run variation during W4.

W4 follows a ramp-and-drop workload where each operating level lasts longer than the short recurring W3 spikes. ANN can therefore respond effectively to the current state because its next-interval predictor has more time to adjust as workload conditions change. This explains why its P99 latency became almost identical to CloudSafe-TP. CloudSafe-TP achieved the same low-latency region through its receding-horizon process. It applies one pool-size decision, collects new measurements, and then repeats the multi-step prediction and candidate evaluation. Hence, it can correct its previous decision when the workload increases or decreases.

SORA showed the poorest W4 performance. Its goodput–concurrency relationship is estimated and updated from runtime observations. Therefore, a model estimated during one workload level may become less suitable after the workload changes. SORA also maintained the largest W4 pool, showing that its higher P99 latency was not simply caused by insufficient concurrency. Rather, its analytical adaptation was slower under the changing W4 operating conditions. PPO produced an intermediate result. It reduced concurrency after W3, but its learned policy did not provide the same low P99 latency as CloudSafe-TP and ANN.

Across W1–W4, CloudSafe-TP recorded mean P99 latencies of 238.80, 241.59, 1,855.13, and 456.53 ms. Its corresponding SDs were 2.72, 2.33, 17.51, and 3.48 ms. These were the lowest SD values among the four controllers in every workload phase. Therefore,

CloudSafe-TP not only maintained competitive or lower mean P99 latency but also showed the smallest run-to-run variation across all four phases.

The 95% confidence intervals are important because they prevent small mean differences from being overinterpreted. In W1, the CloudSafe-TP and SORA intervals overlap considerably, so their P99 performance should be considered comparable. In W2, CloudSafe-TP recorded the lowest mean, but its interval slightly overlaps SORA, so the difference remains modest. In W4, CloudSafe-TP and ANN also have overlapping intervals and should be considered very close. The strongest result appeared in W3, where the CloudSafe-TP confidence interval was completely separated from all three baselines and remained entirely below the 2,000 ms SLO.

Overall, the P99 results show that the main advantage of CloudSafe-TP appears when the workload becomes difficult to represent using only the current or next-interval state. SORA remained highly competitive during the stable W1 workload and performed reasonably in W2. ANN became competitive again during the longer W4 transitions, while PPO provided acceptable latency during the less demanding phases. However, under the recurring W3 spikes, CloudSafe-TP achieved a clear advantage in both mean P99 latency and its confidence interval. This result is consistent with its multi-step prediction, uncertainty-aware conformal safety checking, and receding-horizon pool adjustment. The queue-length and SLO-violation results further examine whether this lower and more stable P99 latency was also supported by better backlog control and SLO compliance.

CPU-Usage Comparative Analysis:

This subsection compares the CPU usage of PPO, SORA, ANN, and CloudSafe-TP across workload phases W1–W4. Each scheme was evaluated through five independent 40-minute runs using different random seeds for the Poisson request-arrival process. The CPU averages from these five runs were used to calculate the mean, standard deviation (SD), and 95% confidence interval (CI). Table 4 reports these values, while Figure 6 shows the mean CPU usage with the corresponding 95% confidence intervals.

Table 4. Mean, standard deviation, and 95% confidence intervals of average CPU usage for PPO, SORA, ANN, and CloudSafe-TP across W1–W4 over five independent runs.

Scheme	Phase	Mean CPU (%)	SD (%)	95% CI Lower (%)	95% CI Upper (%)
PPO	W1	53.306	0.764	52.357	54.254
PPO	W2	68.324	0.741	67.404	69.243
PPO	W3	73.781	0.577	73.065	74.498
PPO	W4	50.831	0.698	49.965	51.698
SORA	W1	53.682	0.757	52.741	54.622
SORA	W2	68.087	0.865	67.014	69.161
SORA	W3	74.697	0.641	73.901	75.493
SORA	W4	66.197	0.639	65.404	66.99
ANN	W1	55.467	0.584	54.741	56.192
ANN	W2	67.82	0.526	67.167	68.473
ANN	W3	78.845	0.676	78.006	79.684
ANN	W4	50.155	0.662	49.333	50.978
CloudSafe-TP	W1	51.228	0.378	50.759	51.697
CloudSafe-TP	W2	69.831	0.366	69.377	70.286
CloudSafe-TP	W3	79.531	0.509	78.899	80.163
CloudSafe-TP	W4	51.251	0.352	50.814	51.688

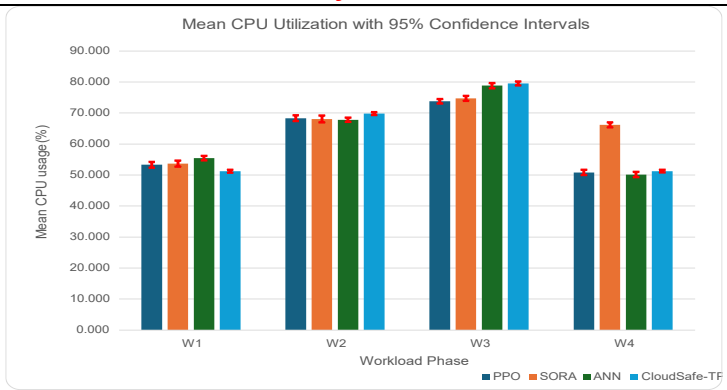


Figure 6. Mean CPU usage of PPO, SORA, ANN, and CloudSafe-TP across W1–W4 with 95% confidence intervals.

CPU usage should be considered together with pool size and workload conditions because a smaller pool does not always result in lower CPU usage. Active workers may remain continuously busy even when fewer threads are available, while other requests wait in the queue.

During W1, CloudSafe-TP recorded the lowest mean CPU usage of 51.23%, followed by PPO at 53.31%, SORA at 53.68%, and ANN at 55.47%. Its 95% CI of 50.76–51.70% did not overlap with PPO (52.36–54.25%), SORA (52.74–54.62%), or ANN (54.74–56.19%). CloudSafe-TP also recorded the lowest SD of 0.38%, showing that its CPU usage remained stable across the five runs.

This behaviour is consistent with its smaller W1 pool. The workload was moderate, and additional concurrency was not required. Therefore, CloudSafe-TP could reduce unnecessary thread activity while keeping predicted latency and resource usage within their limits. CPU usage is also considered during candidate filtering and in the control cost. ANN recorded the highest W1 CPU usage together with a larger pool. Thus, its additional concurrency did not provide a CPU advantage. PPO and SORA remained between ANN and CloudSafe-TP.

The CPU pattern changed during W2. ANN recorded the lowest mean CPU usage of 67.82%, followed by SORA at 68.09% and PPO at 68.32%. CloudSafe-TP recorded the highest value of 69.83%. Its 95% CI was 69.38–70.29%, which remained above ANN (67.17–68.47%), PPO (67.40–69.24%), and SORA (67.01–69.16%). However, the overall difference remained small because W2 was CPU intensive and all schemes were subjected to approximately the same computational workload..

CloudSafe-TP maintained a larger pool during W2, and its slightly higher CPU usage is consistent with the additional concurrency selected by the controller. It did not attempt to minimize CPU usage independently when more processing capacity was useful. Its SD was only 0.37%, which was again the lowest among the evaluated schemes. Hence, the higher W2 CPU usage remained consistent across the five seeds rather than reflecting irregular processor pressure.

During W3, PPO recorded the lowest mean CPU usage of 73.78%, followed by SORA at 74.70%. ANN recorded 78.85%, while CloudSafe-TP reached 79.53%. CloudSafe-TP had a 95% CI of 78.90–80.16%, whereas PPO and SORA remained at 73.07–74.50% and 73.90–75.49%, respectively. ANN recorded an interval of 78.01–79.68%, which overlaps with CloudSafe-TP. Therefore, the CPU difference between ANN and CloudSafe-TP was comparatively small, while the difference from PPO and SORA was clearer. CloudSafe-TP again maintained low variation with an SD of 0.51%.

The higher W3 CPU usage of CloudSafe-TP is consistent with its pool-size decision. W3 contains recurring workload spikes, and CloudSafe-TP maintained the largest pool to

provide sufficient concurrency and reduce future queue growth. Its sparse GP predicts CPU usage together with P99 latency, queue length, and memory usage over the prediction horizon. A candidate is rejected when its predicted CPU usage exceeds the configured limit. Therefore, the higher CPU value does not indicate uncontrolled processor usage. Instead, CloudSafe-TP accepted greater CPU utilization while keeping it within the allowed resource limit. PPO and SORA consumed less CPU while maintaining smaller W3 pools. ANN remained close to CloudSafe-TP in CPU usage despite using fewer threads, suggesting greater processor utilization by its active workers.

During W4, ANN recorded the lowest mean CPU usage of 50.16%, followed by PPO at 50.83% and CloudSafe-TP at 51.25%. SORA remained much higher at 66.20%. CloudSafe-TP had a 95% CI of 50.81–51.69%, which overlaps with ANN (49.33–50.98%) and PPO (49.97–51.70%). Therefore, the small mean differences among CloudSafe-TP, ANN, and PPO should not be considered strong CPU advantages because all three operated within a similar CPU region. CloudSafe-TP still recorded the smallest SD of 0.35%, compared with 0.66% for ANN and 0.70% for PPO.

SORA showed a different W4 behaviour. Its 95% CI of 65.40–66.99% remained completely separated from the other three schemes. SORA also maintained the largest pool during this phase. Since its analytical goodput–concurrency model is updated from current runtime measurements, its concurrency estimate may remain influenced by the preceding high-load condition while the ramp-and-drop workload is changing. ANN and PPO reduced both pool size and CPU demand more strongly, while CloudSafe-TP maintained an intermediate pool and kept its CPU usage close to them through repeated receding-horizon adjustment.

Across W1–W4, CloudSafe-TP recorded mean CPU usages of 51.23%, 69.83%, 79.53%, and 51.25%. Its SD remained the lowest among all four schemes in every phase, ranging from only 0.35% to 0.51%. This shows that its CPU behaviour remained highly consistent across the five random-seed runs. The confidence intervals also show that CloudSafe-TP used clearly less CPU in W1, more CPU than PPO and SORA during the demanding W3 phase, and remained close to PPO and ANN in W4 because their intervals overlap.

Overall, CloudSafe-TP does not minimize CPU usage independently. CPU is considered together with queue length, memory usage, thread allocation, and P99 safety. Therefore, the controller can accept higher CPU usage during W2 and W3 when additional processing capacity is required and reduce it again when workload pressure decreases. Hence, the CPU results should be considered together with P99 latency and queue length to determine whether the additional processor usage provides useful performance improvement.

Memory/Heap-Usage Comparative Analysis:

This subsection compares the heap usage of PPO, SORA, ANN, and CloudSafe-TP across workload phases W1–W4. Each scheme was evaluated through five independent 40-minute runs using different random seeds. The phase-level heap averages were used to calculate the mean, standard deviation (SD), and 95% confidence interval (CI). Table 5 reports these values, while Figure 7 shows the corresponding mean heap usage with 95% confidence intervals. Heap usage should be considered together with workload condition and pool size because a larger thread pool does not always produce proportionally higher Java heap usage.

Table 5. Mean, standard deviation, and 95% confidence intervals of average heap usage (%) for PPO, SORA, ANN, and CloudSafe-TP across W1–W4 over five independent runs.

Scheme	Phase	Mean Heap (%)	SD (%)	95% CI Lower	95% CI Upper
PPO	W1	32.117	0.341	31.693	32.54
PPO	W2	34.866	0.328	34.458	35.274
PPO	W3	38.311	0.306	37.932	38.691

PPO	W4	34.251	0.331	33.84	34.663
SORA	W1	32.086	0.498	31.467	32.705
SORA	W2	34.804	0.451	34.243	35.364
SORA	W3	38.679	0.281	38.329	39.028
SORA	W4	40.971	0.378	40.502	41.441
ANN	W1	40.558	0.597	39.816	41.299
ANN	W2	34.828	0.364	34.376	35.281
ANN	W3	38.637	0.416	38.12	39.154
ANN	W4	33.203	0.368	32.747	33.66
CloudSafe-TP	W1	31.225	0.285	30.871	31.579
CloudSafe-TP	W2	36.355	0.354	35.915	36.795
CloudSafe-TP	W3	39.371	0.329	38.962	39.779
CloudSafe-TP	W4	34.862	0.266	34.531	35.193

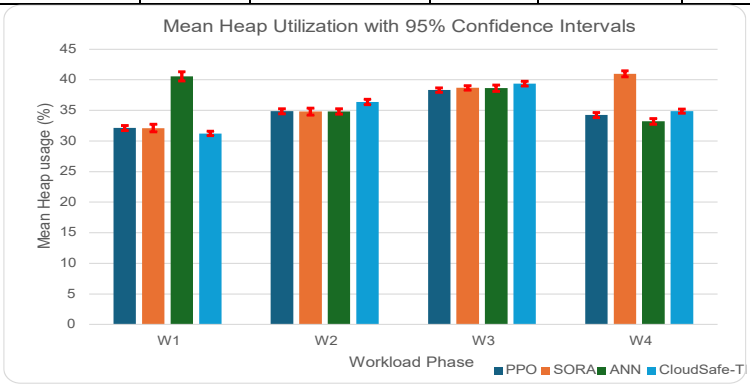


Figure 7. Mean heap usage of PPO, SORA, ANN, and CloudSafe-TP across W1–W4 with 95% confidence intervals.

During W1, CloudSafe-TP recorded the lowest mean heap usage of 31.23%, followed by SORA at 32.09% and PPO at 32.12%, while ANN recorded a much higher value of 40.56%. CloudSafe-TP had a 95% CI of 30.87–31.58%, which remained below PPO (31.69–32.54%) and clearly below ANN (39.82–41.30%). Its interval was also slightly separated from SORA's interval CloudSafe-TP recorded the lowest SD of 0.29%. This result is consistent with its smaller W1 pool and its consideration of predicted memory usage when evaluating candidate configurations.

The pattern changed during W2. PPO, SORA, and ANN recorded very similar mean heap usages of 34.87%, 34.80%, and 34.83%, respectively, and their confidence intervals largely overlapped. CloudSafe-TP recorded the highest mean of 36.36%, with a 95% CI of 35.92–36.80%, which remained above the three baselines. This increase is consistent with its larger W2 pool. CloudSafe-TP accepted additional concurrency under the more demanding workload, which increased active request and runtime objects. However, its SD remained low at 0.35%, showing that the higher heap usage was consistent across the five runs.

During W3, heap usage increased for all schemes. PPO recorded the lowest mean of 38.31%, followed by ANN at 38.64% and SORA at 38.68%, while CloudSafe-TP recorded 39.37%. Its 95% CI was 38.96–39.78%. This interval slightly overlaps with the intervals for SORA and ANN but remains above the PPO interval. Therefore, the heap differences during W3 were relatively small. CloudSafe-TP used slightly more heap because it maintained the largest pool to provide sufficient concurrency during repeated workload spikes. Even with this higher concurrency, its SD remained low at 0.33%, showing stable memory usage across the five seeds.

PPO recorded the lowest W3 heap usage, but this result should not be considered independently from its smaller pool and lower concurrency. SORA and ANN also remained

in almost the same heap region. CloudSafe-TP accepted slightly higher memory usage when additional concurrency was required to reduce future queue and latency risk. Thus, its controller does not minimize memory independently but considers it together with queue length, CPU usage, pool size, and P99 safety.

During W4, ANN recorded the lowest mean heap usage of 33.20%, followed by PPO at 34.25%, CloudSafe-TP at 34.86%, and SORA at 40.97%. Their 95% CIs were 32.75–33.66% for ANN, 33.84–34.66% for PPO, 34.53–35.19% for CloudSafe-TP, and 40.50–41.44% for SORA. The intervals show clear separation among ANN, CloudSafe-TP, and SORA, while the PPO and CloudSafe-TP intervals slightly overlap.

ANN's lower W4 heap usage is consistent with its stronger reduction in pool size. PPO also reduced concurrency and remained below CloudSafe-TP. CloudSafe-TP maintained a somewhat larger pool so that it could respond to the increasing portions of W4 without allowing excessive queue growth. This explains its slightly higher heap usage. However, its SD was only 0.27%, which was the lowest in W4 and shows very stable memory behaviour across the five runs.

SORA showed a different W4 behaviour. It recorded the highest heap usage by a large margin, and its confidence interval was completely separated from the other schemes. SORA also maintained the largest W4 pool. Since its analytical goodput–concurrency model is updated from runtime measurements, its concurrency decision may remain influenced by an earlier high-load condition while the workload is decreasing or changing. Therefore, it retained more resources during W4 than the other controllers.

Across W1–W4, CloudSafe-TP recorded mean heap usages of 31.23%, 36.36%, 39.37%, and 34.86%. Its SD remained between 0.27% and 0.35%, showing consistent memory behaviour across all five random-seed runs. The confidence intervals show that CloudSafe-TP used less heap in W1, allowed higher heap usage in W2 and W3 when it maintained greater concurrency, and reduced its heap usage again in W4.

Overall, CloudSafe-TP does not always minimize heap usage. Its memory consumption changes according to the concurrency required by the workload. It reduced heap usage during moderate W1 conditions, accepted additional memory usage during W2 and W3 when higher concurrency was useful, and reduced heap usage again in W4. The small SD values further show that this behaviour remained consistent across the repeated runs.

Queue-Length Comparative Analysis:

This subsection compares the average queue length of PPO, SORA, ANN, and CloudSafe-TP across workload phases W1–W4. Each scheme was evaluated through five independent 40-minute runs using different random seeds. The phase-level queue averages from these runs were used to calculate the mean, standard deviation (SD), and 95% confidence interval (CI). Table 6 reports these values, while Figure 8 shows the mean queue lengths with 95% confidence intervals. Queue length is important because it indicates whether the available worker threads can process incoming requests without creating a persistent backlog.

Table 6. Mean, standard deviation, and 95% confidence intervals of average queue length for PPO, SORA, ANN, and CloudSafe-TP across W1–W4 over five independent runs.

Scheme	Phase	Mean Queue Length	SD	95% CI Lower	95% CI Upper
PPO	W1	28.093	0.917	26.955	29.231
PPO	W2	47.48	1.42	45.717	49.242
PPO	W3	74.992	1.914	72.616	77.369
PPO	W4	85.941	1.748	83.771	88.111
SORA	W1	24.028	0.946	22.853	25.203
SORA	W2	35.826	1.003	34.582	37.071
SORA	W3	55.988	1.275	54.404	57.571

SORA	W4	68.82	1.06	67.504	70.136
ANN	W1	21.982	0.865	20.908	23.056
ANN	W2	43.774	1.132	42.369	45.179
ANN	W3	70.287	1.616	68.281	72.294
ANN	W4	88.385	1.655	86.33	90.439
CloudSafe-TP	W1	27.139	0.487	26.535	27.743
CloudSafe-TP	W2	28.646	0.585	27.919	29.372
CloudSafe-TP	W3	33.738	0.989	32.51	34.967
CloudSafe-TP	W4	87.061	0.962	85.867	88.255

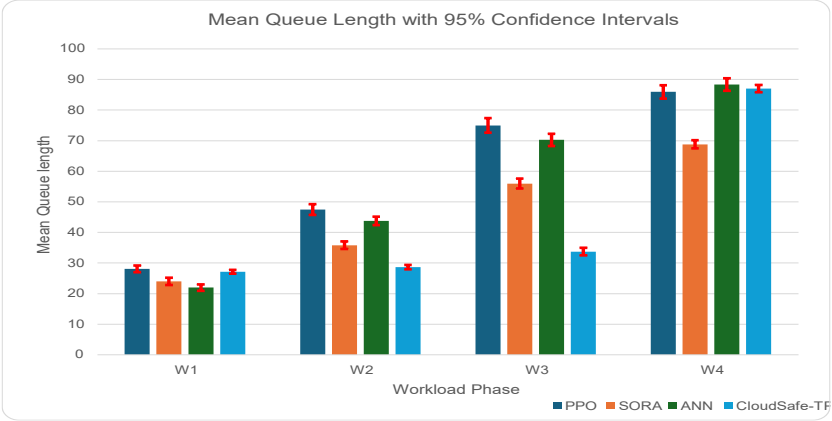


Figure 8. Mean queue length of PPO, SORA, ANN, and CloudSafe-TP across W1–W4 with 95% confidence intervals.

During W1, ANN recorded the lowest mean queue length of 21.98 requests, followed by SORA at 24.03, CloudSafe-TP at 27.14, and PPO at 28.09. ANN had a 95% CI of 20.91–23.06, SORA 22.85–25.20, CloudSafe-TP 26.54–27.74, and PPO 26.96–29.23. Thus, ANN and SORA recorded shorter mean queues, while the intervals of CloudSafe-TP and PPO overlapped. CloudSafe-TP still recorded the lowest SD of 0.49 requests, showing stable queue behaviour across the five runs. Its slightly higher W1 queue is consistent with the smaller pool selected under moderate load. The controller therefore accepted a small amount of waiting instead of maintaining unnecessary concurrency only to reduce the queue.

During W2, CloudSafe-TP achieved the lowest mean queue length of 28.65 requests, followed by SORA at 35.83, ANN at 43.77, and PPO at 47.48. Their confidence intervals were clearly separated. CloudSafe-TP remained within 27.92–29.37, SORA within 34.58–37.07, ANN within 42.37–45.18, and PPO within 45.72–49.24. CloudSafe-TP also recorded the lowest SD of 0.59 requests. Hence, its W2 queue advantage remained consistent across all five runs.

This lower queue is consistent with the larger W2 pool selected by CloudSafe-TP. The controller predicts future queue behaviour for every candidate and also includes queue length in its control cost. Therefore, a candidate expected to create continuing backlog becomes less suitable even when its immediate latency remains acceptable. SORA produced the second-lowest queue because its analytical goodput–concurrency model recognized the need for additional capacity. ANN and PPO maintained smaller pools and allowed more requests to remain waiting.

The difference became much larger during W3. CloudSafe-TP recorded a mean queue of only 33.74 requests, compared with 55.99 for SORA, 70.29 for ANN, and 74.99 for PPO. Its 95% CI was 32.51–34.97, which remained completely separated from SORA (54.40–57.57), ANN (68.28–72.29), and PPO (72.62–77.37). CloudSafe-TP also recorded a low SD of 0.99 requests. Therefore, its lower W3 queue remained consistent across the different workload realizations generated by the different random seeds.

This result is directly related to the recurring workload spikes in W3. CloudSafe-TP maintained the largest pool and predicted how the queue could develop over the following control intervals. Therefore, a smaller candidate could be avoided when it was expected to leave part of the backlog unresolved before the next spike. SORA performed better than ANN and PPO because its latency-aware analytical model increased concurrency under higher demand. However, it does not explicitly predict the future queue over several intervals. ANN uses next-interval prediction, while PPO selects increase, hold, and decrease actions without evaluating the future queue trajectory of every candidate. Hence, both schemes allowed greater queue buildup during W3.

During W4, the pattern changed considerably. SORA recorded the lowest mean queue length of 68.82 requests, followed by PPO at 85.94, CloudSafe-TP at 87.06, and ANN at 88.39. SORA had a 95% CI of 67.50–70.14, which remained clearly separated from the other schemes. CloudSafe-TP recorded 85.87–88.26, which overlapped with PPO (83.77–88.11) and ANN (86.33–90.44). Therefore, the queue differences among CloudSafe-TP, PPO, and ANN were small, while SORA clearly maintained the shortest W4 queue. CloudSafe-TP again showed low variation with an SD of 0.96 requests.

The W4 result is different because the workload increases and decreases within the same phase. SORA maintained the largest pool, which provided more concurrency and reduced waiting requests. CloudSafe-TP reduced its pool after W3 and maintained an intermediate allocation, while PPO and ANN reduced concurrency more strongly. Therefore, CloudSafe-TP did not achieve the lowest W4 queue because its controller balances queue length with thread allocation, CPU usage, memory usage, and adjustment cost rather than minimizing queue length independently.

Across W1–W4, CloudSafe-TP recorded mean queue lengths of 27.14, 28.65, 33.74, and 87.06 requests. Its SD remained between 0.49 and 0.99 requests, showing repeatable queue behaviour across the five runs. The confidence intervals show that CloudSafe-TP was comparable with PPO in W1, clearly better than all baselines in W2 and particularly W3, while SORA achieved the lowest queue in W4 and CloudSafe-TP remained close to PPO and ANN.

Overall, CloudSafe-TP did not minimize queue length in every workload phase. Its strongest queue-control advantage appeared in W2 and W3, where insufficient concurrency could quickly create persistent backlog. During W1, it accepted a slightly larger queue to avoid unnecessary thread allocation. During W4, it maintained an intermediate queue because the controller balanced queue reduction with its other resource and stability objectives.

P99 SLO-Violation Analysis:

No SLO violations were observed for any scheme during W1, W2, and W4. Therefore, Table 7 and Figure 9 report the SLO-violation statistics only for W3.

Table 7. Mean, standard deviation, and 95% confidence intervals of SLO violation rate during W3 across five independent runs.

Scheme	Mean SLO Violation (%)	SD (%)	95% CI Lower (%)	95% CI Upper (%)
PPO	23.568	1.196	22.083	25.052
SORA	10.831	1.648	8.785	12.877
ANN	23.759	0.863	22.687	24.831
CloudSafe-TP	2.479	0.932	1.322	3.635

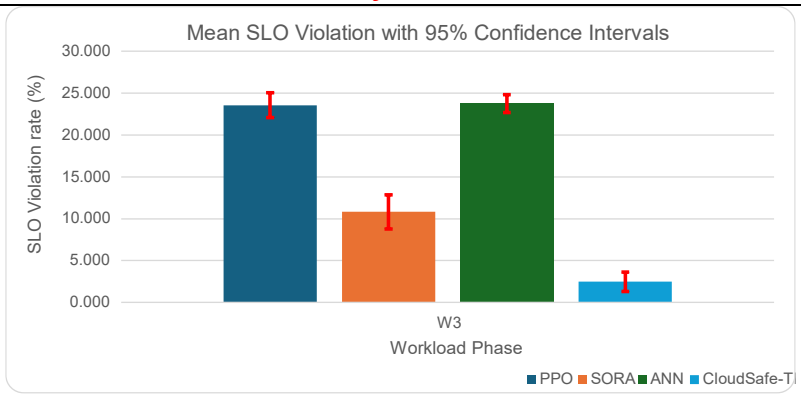


Figure 9. Mean SLO violation rate (%) with 95% confidence interval of all schemes in W3 phase.

During W1, the workload was moderate and stable, and all controllers maintained P99 latency below the 2,000 ms SLO. W2 increased the request rate, but the CPU-bound requests had relatively short service times. Hence, all schemes again remained within the SLO. W4 contained changing workload levels, but these transitions were more gradual than the recurring short spikes of W3. This provided more time for the controllers to adjust their pool sizes, and no SLO violations occurred during this phase.

The main difference appeared during W3, where the workload contained repeated short-duration spikes. PPO recorded a mean SLO-violation rate of 23.57%, with an SD of 1.20% and a 95% CI of 22.08–25.05%. PPO adjusts the pool through learned increase, hold, and decrease actions, while SLO protection is handled indirectly through its reward. Therefore, its response may not be sufficient when high-intensity spikes repeatedly occur within short intervals.

ANN produced a similar result and recorded the highest mean violation rate of 23.76%, with an SD of 0.86% and a 95% CI of 22.69–24.83%. Its confidence interval strongly overlaps with PPO, showing similar W3 violation behaviour across the five runs. ANN predicts the next-interval outcome for candidate pool sizes, but it relies on point prediction without calibrated uncertainty or multi-step latency evaluation. Therefore, a candidate that appears suitable for the next interval may still lead to an SLO violation during later spikes.

SORA performed considerably better than PPO and ANN. It recorded a mean violation rate of 10.83%, with an SD of 1.65% and a 95% CI of 8.79–12.88%. Its confidence interval remained clearly below PPO and ANN. This result is consistent with its latency-aware analytical control, which can increase concurrency when workload pressure becomes high. However, SORA does not predict the multi-step P99 trajectory with calibrated uncertainty. Therefore, violations still occurred when W3 repeatedly changed from its normal request rate to short high-intensity spikes.

CloudSafe-TP achieved the lowest W3 SLO-violation rate. Its mean violation rate was only 2.48%, with an SD of 0.93% and a 95% CI of 1.32–3.64%. This interval was completely separated from PPO, SORA, and ANN. CloudSafe-TP predicts future P99 latency and other system states over multiple control intervals, applies the conformal upper P99 bound, and removes candidates that are expected to cross the SLO. Therefore, during recurring W3 spikes, the controller maintains higher concurrency when required and reduces the chance that queued requests remain long enough to push P99 latency beyond the 2,000 ms limit.

Overall, SLO violations were mainly associated with the bursty W3 workload rather than all workload phases. PPO and ANN recorded mean violation rates of about 24%, while SORA reduced the rate to about 11%. CloudSafe-TP further reduced it to approximately 2.5%. Its relatively small SD and clearly separated 95% confidence interval show that this lower violation behaviour remained consistent across the five independent runs.

Statistical Significance Analysis:

The mean, SD, and 95% confidence intervals reported for W1–W4 provide the repeated-run descriptive analysis for all workload phases. In addition, W3 was selected for targeted inferential analysis because it represents the most demanding workload condition, contains repeated short-duration spikes, and is the only phase where SLO violations occurred. Therefore, W3 provides the most relevant condition for examining whether the QoS differences between CloudSafe-TP and the baseline schemes are statistically significant. This analysis complements the five-run descriptive results already reported for W1–W4.

Since the same five random seeds were used for CloudSafe-TP and each baseline, a two-sided paired t-test was used. For each paired run, the difference was calculated as:

$$d_i = X_{\text{CloudSafe-TP},i} - X_{\text{Baseline},i}$$

The mean paired difference was calculated as:

$$\bar{d} = \frac{1}{n}\sum_{i=1}^{n} d_i$$

The corresponding paired t-statistic was then calculated as:

$$t = \frac{\bar{d}}{s_d / \sqrt{n}}$$

where s_d is the sample standard deviation of the paired differences. Since five runs were used:

$$n = 5, \quad df = n - 1 = 4$$

The null and alternative hypotheses were defined as:

$$H_0: \mu_d = 0$$
$$H_1: \mu_d \neq 0$$

A comparison was considered statistically significant when:

$$p < 0.05$$

The 95% confidence interval of the paired difference was also calculated as:

$$\bar{d} \pm t_{0.975,4} \frac{s_d}{\sqrt{n}}$$

For the W3 inferential analysis, CloudSafe-TP was compared separately with PPO, SORA, and ANN for P99 latency, average queue length, and SLO violation rate. Table 8 reports the mean paired differences, t-statistics, p-values, and 95% confidence intervals. A negative mean difference shows that CloudSafe-TP produced a lower value than the corresponding baseline.

Table 8. Paired t-test results for CloudSafe-TP against PPO, SORA, and ANN for the main W3 QoS metrics across five runs.

Metric	Comparison	Mean Difference	t-statistic	p-value	95% CI of Difference	Result
P99 latency	CloudSafe-TP vs PPO	-554.32 ms	-29.986	<0.001	[-605.64, -502.99]	Significant
P99 latency	CloudSafe-TP vs SORA	-143.34 ms	-7.450	0.0017	[-196.76, -89.93]	Significant
P99 latency	CloudSafe-TP vs ANN	-503.99 ms	-27.439	<0.001	[-554.99, -452.99]	Significant
Queue length	CloudSafe-TP vs PPO	-41.25	-33.536	<0.001	[-44.67, -37.84]	Significant
Queue length	CloudSafe-TP vs SORA	-22.25	-62.839	<0.001	[-23.23, -21.27]	Significant
Queue length	CloudSafe-TP vs ANN	-36.55	-33.664	<0.001	[-39.56, -33.53]	Significant
SLO violation rate	CloudSafe-TP vs PPO	-21.09%	-24.560	<0.001	[-23.47, -18.70]	Significant

SLO violation rate	CloudSafe-TP vs SORA	-8.35%	-10.226	<0.001	[-10.62, -6.08]	Significant
SLO violation rate	CloudSafe-TP vs ANN	-21.28%	-35.676	<0.001	[-22.94, -19.62]	Significant

As shown in Table 8, all W3 comparisons have $p < 0.05$, and none of the 95% confidence intervals of the paired differences include zero. Therefore, CloudSafe-TP achieved statistically significant reductions in P99 latency, queue length, and SLO violation rate compared with PPO, SORA, and ANN under the bursty W3 workload. This supports the main observation that the strongest advantage of CloudSafe-TP appears when the system is exposed to repeated and rapidly changing workload spikes.

Runtime Overhead:

The runtime overhead of CloudSafe-TP was measured directly on the OCI VM deployment by recording the execution time of the controller decision loop. During each one-second control interval, the time required to process the current state, evaluate candidate pool sizes, perform prediction and safety checking, and select the next pool configuration was measured.

CloudSafe-TP recorded an average decision time of 20.8 ms, a median of 19.6 ms, a 95th-percentile time of 32.7 ms, and a maximum of 46.3 ms. These values represent 2.08%, 1.96%, 3.27%, and 4.63% of the one-second control interval, respectively. Even the maximum decision time remained below 5% of the control interval. Therefore, the controller introduced only a small runtime overhead and remained suitable for the one-second real-time control used in the OCI deployment.

Limitations:

Scalability to Larger Distributed Deployments:

CloudSafe-TP was evaluated using one load-balancer VM and two heterogeneous backend VMs, while detailed results were reported for the resource-constrained backend. Each backend runs an independent CloudSafe-TP controller without relying on a central controller. However, the present evaluation does not confirm its behaviour in much larger deployments. At larger scale, workload imbalance, dynamic routing, service placement, and infrastructure-level autoscaling may affect local pool-size decisions. Therefore, the current study used fixed routing so that thread-pool control could be evaluated separately.

Communication and Heterogeneous Cloud Conditions:

CloudSafe-TP uses only local backend measurements and does not require communication among controllers for normal pool-size decisions. Thus, failure of one controller does not stop the remaining backend controllers. However, real deployments may include network delays, routing changes, backend failures, and temporary workload imbalance. The current P99 measurement is internal to the thread-pool system and does not include network or client-side latency. Different VM types or cloud regions may also require separate training and conformal calibration. Cross-node coordination would be required if CloudSafe-TP is later combined with adaptive load balancing or global resource management.

Highly Dynamic and Previously Unseen Workloads:

CloudSafe-TP uses models trained from workload traces, and these models remain fixed during final evaluation. Therefore, workloads that differ substantially from the training and calibration conditions may reduce prediction accuracy, particularly during rapid workload changes. Conformal calibration provides additional protection against prediction uncertainty, but it cannot guarantee the same behaviour under every unseen condition. Future work should therefore evaluate larger deployments, longer workloads, more extreme workload changes, online model updating, and adaptive load balancing.

Component-Level Ablation: A conventional removal-based ablation was not included because CloudSafe-TP operates as an integrated hybrid control pipeline. The sparse GP

provides multi-step state predictions, conformal calibration converts the predicted P99 latency into a safety bound, and the MPC layer uses these predictions and safety constraints to select the final pool size. Therefore, removing one component would also require changes in the inputs, decision rules, or tuning of the remaining components. Such a reduced version would become a different controller rather than a directly comparable form of CloudSafe-TP. Hence, the present study evaluates the complete hybrid controller against representative analytical, supervised-learning, and reinforcement-learning baselines. A separate component-level study with redesigned and retuned reduced variants is left for future work.

Conclusion:

This paper presented CloudSafe-TP, an SLO-aware predictive controller for dynamic thread-pool management in heterogeneous distributed cloud backends. Each backend runs an independent local controller that combines sparse Gaussian-process prediction, conformal calibration, and model predictive control. For every candidate pool size, CloudSafe-TP predicts future P99 latency, queue length, CPU usage, and memory usage. It then applies the conformal upper P99 latency bound and resource constraints to remove unsafe candidates and selects a safe configuration with the lowest control cost.

CloudSafe-TP was evaluated on Oracle Cloud Infrastructure under moderate, CPU-intensive, bursty, and ramp-and-drop workloads. Each controller was tested through five independent 40-minute runs using random seeds 0, 1, 2, 3, and 4. The results were analyzed using mean values, standard deviations, and 95% confidence intervals. The controller did not follow a fixed preference for either small or large pools. It reduced unnecessary concurrency during moderate load, increased the pool when sustained or bursty demand required more processing capacity, and reduced it again when workload pressure decreased.

The strongest improvement appeared during the bursty W3 workload. CloudSafe-TP achieved a mean P99 latency of 1,855.13 ms, a mean queue length of 33.74 requests, and an SLO-violation rate of 2.48%. Compared with the baselines, it reduced mean P99 latency by 7.17–23.01%, queue length by 39.74–55.01%, and SLO violation rate by 77.11–89.57%. Its 95% confidence intervals for these W3 QoS metrics showed clear separation from the baselines, while paired t-tests confirmed statistically significant improvements in P99 latency, queue length, and SLO violation rate.

CloudSafe-TP used higher concurrency during W3 and therefore also accepted higher CPU and heap usage when required. However, these resources remained within the configured limits, while the controller achieved much lower P99 latency, queue length, and SLO violations. Under the moderate W1 workload, it used 20.70–26.60% fewer threads and also recorded the lowest CPU and heap usage. This shows that CloudSafe-TP increases resources when they provide useful processing capacity and reduces them when additional concurrency is not required.

Overall, the results have shown that effective thread-pool control requires a balance among latency, queue length, concurrency, and resource usage. CloudSafe-TP achieved this balance by considering future backend conditions and prediction uncertainty before changing the pool size. However, the evaluation was performed on one OCI deployment with two heterogeneous backend VMs, and detailed results were reported for the resource-constrained backend. Larger deployments, more diverse workload conditions, online model updating, and evaluation of throughput and cross-backend fairness under adaptive workload distribution remain areas for future work.

Acknowledgement: The authors gratefully acknowledge Hazara University, Mansehra, for its academic support and for facilitating access to the cloud-computing resources used in this study.

Author's Contribution: Faisal Bahadur: Conceptualization, Methodology, Software, Writeup and Supervision. Arooba Akhtar: Experimental Deployment and Configurations, Model

Training. Nadir Hussain: Data Curation, Investigation and Validation. Abdul Qadir: Testing, Evaluation and Analysis.

Conflict of interest: The authors declare no conflict of interest.

References:

[1] Z. Abdiramanov, "Design and implementation of a scalable microservice-based cloud computing service for stochastic fluid flow modeling," *Array*, vol. 29, p. 100679, Mar. 2026, doi: 10.1016/J.ARRAY.2026.100679.

[2] L. Zanussi, D. Tessera, L. Massari, B. Bermejo, C. Juiz, and M. Calzarossa, "Load-aware predictive auto-scaling framework for cloud environments," *Clust. Comput. 2026 293*, vol. 29, no. 3, pp. 197-, Mar. 2026, doi: 10.1007/S10586-026-05944-X.

[3] D. Charlak, J. Brzeziński, and G. Kozieł, "Comparative analysis of reactive programming and Java virtual threads," *J. Comput. Sci. Inst.*, vol. 39, pp. 154–160, Jun. 2026, doi: 10.35784/JCSI.9409.

[4] C. Costa and J. H. Brito, "Throughput impact of software multithreading for deep-learning inference on the AMD Kria KV260," *J. Real-Time Image Process. 2026 232*, vol. 23, no. 2, pp. 89-, Apr. 2026, doi: 10.1007/S11554-026-01889-X.

[5] S. Harutyunyan Gevorgyan, E. César, A. Sikora, J. Filipovič, and J. Alcaraz, "Automatic tuning based on hardware performance counters and machine learning," *Futur. Gener. Comput. Syst.*, vol. 179, p. 108358, Jun. 2026, doi: 10.1016/J.FUTURE.2025.108358.

[6] W. Du, H. Yuan, Y. Ren, J. Yu, and X. Ren, "Tiered Scheduling and Cost-Benefit-Aware Speculation for Optimizing Spark in Heterogeneous Clusters," *Concurr. Comput. Pract. Exp.*, vol. 38, no. 13, p. e70844, Jul. 2026, doi: 10.1002/CPE.70844.

[7] "Chapter 9. Distributed Executor Service." Accessed: Aug. 14, 2026. [Online]. Available: https://docs.hazelcast.org/docs/2.3/manual/html/ch09.html

[8] L. Akash, D. Fernando, M. Jayasinghe, C. Keppitiyagama, and K. Thangarajah, "Machine Learning Based Thread Pool Tuning via Program Analysis," *2021 IEEE 23rd Int. Conf. High Perform. Comput. Commun. 7th Int. Conf. Data Sci. Syst. 19th Int. Conf. Smart City 7th Int. Conf. Dependability Sensor, C…*, pp. 648–653, 2022, doi: 10.1109/HPCC-DSS-SMARTCITY-DEPENDSYS53884.2021.00108.

[9] J. Alcaraz *et al.*, "Predicting number of threads using balanced datasets for openMP regions," *Comput. 2022 1055*, vol. 105, no. 5, pp. 999–1017, Apr. 2022, doi: 10.1007/S00607-022-01081-6.

[10] F. H. S. da Silva, J. B. Fernandes, I. M. Sardina, T. Barros, S. Xavier-de-Souza, and I. A. S. Assis, "Auto-Tuning for OpenMP Dynamic Scheduling applied to Full Waveform Inversion," May 2025, doi: 10.1016/j.cageo.2025.105932.

[11] A. Dutta, J. Alcaraz, A. Tehranijamsaz, A. Sikora, E. Cesar, and A. Jannesari, "Pattern-based Autotuning of OpenMP Loops using Graph Neural Networks," *Proc. AI4S 2022 Artif. Intell. Mach. Learn. Sci. Appl. Held conjunction with SC 2022 Int. Conf. High Perform. Comput. Networking, Storage Anal.*, pp. 26–31, 2022, doi: 10.1109/AI4S56813.2022.00010.

[12] A. Dutta, J. Choi, and A. Jannesari, "Power Constrained Autotuning using Graph Neural Networks," *Proc. - 2023 IEEE Int. Parallel Distrib. Process. Symp. IPDPS 2023*, pp. 535–545, 2023, doi: 10.1109/IPDPS54959.2023.00060.

[13] E. C. de Lima, F. D. Rossi, M. C. Luizelli, R. N. Calheiros, and A. F. Lorenzon, "A neural network framework for optimizing parallel computing in cloud servers," *J. Syst. Archit.*, vol. 150, p. 103131, May 2024, doi: 10.1016/J.SYSARC.2024.103131.

[14] "CAPES: Unsupervised Storage Performance Tuning Using Neural Network-Based Deep Reinforcement Learning | IEEE Conference Publication | IEEE Xplore." Accessed: Aug. 14, 2026. [Online]. Available:

https://ieeexplore.ieee.org/document/9926120
[15] H. T. Nguyen, T. V. Do, and C. Rotter, "Optimizing the resource usage of actor-based systems," *J. Netw. Comput. Appl.*, vol. 190, p. 103143, Sep. 2021, doi: 10.1016/J.JNCA.2021.103143.
[16] J. Schwarzrock, C. C. De Oliveira, M. Ritt, A. F. Lorenzon, and A. C. S. Beck, "A Runtime and Non-Intrusive Approach to Optimize EDP by Tuning Threads and CPU Frequency for OpenMP Applications," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 7, pp. 1713–1724, Jul. 2021, doi: 10.1109/TPDS.2020.3046537.
[17] T. S. Medeiros, L. Pereira, F. D. Rossi, M. C. Luizelli, A. C. S. Beck, and A. F. Lorenzon, "Mitigating the processor aging through dynamic concurrency throttling," *J. Parallel Distrib. Comput.*, vol. 156, pp. 86–100, Oct. 2021, doi: 10.1016/J.JPDC.2021.05.006.
[18] J. Liu, Q. Wang, S. Zhang, L. Hu, and D. Da Silva, "Sora: A Latency Sensitive Approach for Microservice Soft Resource Adaptation," *Middlew. 2023 - Proc. 24th ACM/IFIP Int. Middlew. Conf.*, pp. 43–56, Nov. 2023, doi: 10.1145/3590140.3592851.
[19] "Cloud Infrastructure | Oracle." Accessed: Aug. 14, 2026. [Online]. Available: https://www.oracle.com/cloud/
[20] F. Bahadur, A. I. Umar, I. Ullah, F. Algarni, and M. A. Khan, "The Double Edge Sword Based Distributed Executor Service," *Comput. Syst. Sci. Eng.*, vol. 42, no. 2, pp. 589–604, Jan. 2022, doi: 10.32604/CSSE.2022.022319.
[21] J. Park, B. Choi, C. Lee, and D. Han, "Graph Neural Network-Based SLO-Aware Proactive Resource Autoscaling Framework for Microservices," *IEEE/ACM Trans. Netw.*, vol. 32, no. 4, pp. 3331–3346, 2024, doi: 10.1109/TNET.2024.3393427.
[22] J. P. K. S. Nunes, S. Nejati, M. Sabetzadeh, and E. Y. Nakagawa, "Self-adaptive, Requirements-driven Autoscaling of Microservices," *Proc. - 2024 IEEE/ACM 19th Symp. Softw. Eng. Adapt. Self-Managing Syst. SEAMS 2024*, pp. 168–174, Apr. 2024, doi: 10.1145/3643915.3644094
[23] K. Hu, M. Xu, K. Ye, and C. Xu, "LSRAM: A Lightweight Autoscaling and SLO Resource Allocation Framework for Microservices Based on Gradient Descent," *Softw. - Pract. Exp.*, vol. 55, no. 4, pp. 714–730, Nov. 2024, doi: 10.1002/spe.3395.
[24] F. Bahadur, Z. Ahmad, and A. Algarni, "PoolRunner: An Extensible Performance Testing Simulation Tool for Thread-Pool Middleware," *IEEE Access*, vol. 13, pp. 96660–96680, 2025, doi: 10.1109/ACCESS.2025.3575437.
[25] "Compute Shapes." Accessed: Aug. 14, 2026. [Online]. Available: https://docs.oracle.com/enus/iaas/Content/Compute/References/computeshapes.htm
[26] "JDK 17 Readme." Accessed: Aug. 14, 2026. [Online]. Available: https://www.oracle.com/java/technologies/javase/jdk17-readme-downloads.html
[27] "Networking Overview." Accessed: Aug. 14, 2026. [Online]. Available: https://docs.oracle.com/en-us/iaas/Content/Network/Concepts/overview.htm



Copyright © by authors and 50Sea. This work is licensed under Creative Commons Attribution 4.0 International License.