

A Comparative Analysis of Static Scaling, Kubernetes Horizontal Pod Autoscaling (HPA), and Kubernetes Event-Driven Autoscaling (KEDA) in Moodle LMS

Robert Achang'a, Elisha Abade, Andrew Kahonge

Department of Computing and Informatics, University of Nairobi, Nairobi, Kenya

Email: achangarobert@gmail.com, eabade@uonbi.ac.ke, andrew.mwaura@uonbi.ac.ke

How to cite this paper: Achang'a, R., Abade, E. and Kahonge, A. (2026) A Comparative Analysis of Static Scaling, Kubernetes Horizontal Pod Autoscaling (HPA), and Kubernetes Event-Driven Autoscaling (KEDA) in Moodle LMS. *Journal of Computer and Communications*, 14, 1-19.
<https://doi.org/10.4236/jcc.2026.1410001>

Received: September 1, 2026

Accepted: October 18, 2026

Published: October 21, 2026

Copyright © 2026 by author(s) and Scientific Research Publishing Inc.
This work is licensed under the Creative Commons Attribution International License (CC BY 4.0).
<http://creativecommons.org/licenses/by/4.0/>



Open Access

Abstract

Moodle is widely used to administer online examinations, but fixed-capacity deployments can become unstable when many learners log in, open quiz pages, and submit responses within the same examination window. This study evaluates three deployment strategies for a Moodle-based examination platform running on an on-premises Kubernetes cluster: static scaling, Kubernetes Horizontal Pod Autoscaling (HPA), and Kubernetes Event-Driven Autoscaling (KEDA). Apache JMeter generated workloads from 500 to 70,000 concurrent connections. Performance was assessed using average response time, 95th percentile latency, 99th percentile latency, throughput, and error rate. Static scaling degraded earliest: average response time increased from 328 ms at 500 concurrent connections to 10,347 ms at 2500 concurrent connections, while the error rate rose from 0% to 81% over the same interval. HPA produced the strongest reliability profile, maintaining 0% errors from 500 to 5000 concurrent connections and recording the lowest error rate among the three strategies at every workload from 10,000 to 70,000 concurrent connections. KEDA produced the strongest responsiveness among successful requests at several higher-load points, including 412 ms average response time at 10,000 concurrent connections and 1604 ms at 70,000 concurrent connections, and it achieved the highest throughput at most large workloads. However, KEDA's error rate increased sharply after 5,000 concurrent connections, reaching 93.725% at 10,000 concurrent connections. The findings show that autoscaling improves Moodle performance compared with static provisioning, but HPA and KEDA optimize different parts of the performance envelope. For high-stakes online examinations, HPA is the safer reliability-oriented option in this dataset, while KEDA is promising when responsiveness and throughput are prioritized and its triggers are carefully tuned.

Keywords

Kubernetes, Horizontal Pod Autoscaler, Kubernetes Event-Driven Autoscaler, Moodle, Autoscaling, Online Examination, Performance Evaluation, Peak Workload

1. Introduction

In scheduled online examinations, many candidates access the platform within the same examination window, authenticate almost at the same time, open quiz pages, save responses, and submit answers close to fixed deadlines. This concentration of activity makes examination traffic more demanding than ordinary LMS browsing or asynchronous course access. Digital learning and online assessment have therefore become central to higher education because institutions increasingly need platforms that can support content delivery, learner engagement, grading workflows, and high-stakes examinations at scale [1]-[3].

Learning Management Systems (LMSs) provide the application layer through which these teaching, learning, and assessment processes are delivered. Moodle is a widely adopted open-source LMS that supports course management, quizzes, grading, and examination workflows [4] [5]. This makes Moodle a suitable workload for studying the infrastructure behavior of online examination systems under peak demand. The shift from general LMS use to Moodle is therefore not only a platform choice; it is a practical way to evaluate how a real educational application behaves when many learners perform time-sensitive actions simultaneously.

Many institutional Moodle deployments are still operated on fixed-capacity infrastructure. Static provisioning is simple to manage, but it offers no automatic response when demand exceeds the configured number of application instances. When demand rises quickly, the platform may respond slowly, queue requests, or fail requests altogether. Prior work on e-learning implementation has associated reliable online learning delivery with infrastructure readiness, technical capacity, and sustainable operational support [6]-[9]. In this study, that broader infrastructure concern is examined through the narrower and measurable problem of Moodle peak-load performance.

Kubernetes provides a practical way to run and scale containerized applications. It manages replicated application instances across a cluster and can adjust capacity when workloads change [10] [11]. Kubernetes autoscaling patterns commonly include Horizontal Pod Autoscaling (HPA), Vertical Pod Autoscaling (VPA), and event-driven scaling. Kubernetes Event-Driven Autoscaler (KEDA) extends Kubernetes scaling by allowing workloads to scale based on event sources and external metrics while integrating with Kubernetes autoscaling mechanisms [12]-[15].

Online examinations provide a useful test case for comparing these approaches because examination traffic can change faster than ordinary e-learning usage.

CPU-based scaling can provide a stable response once resource pressure is visible, but it may react only after demand has already affected the running pods. Event-driven scaling can react closer to a demand signal, but it may still be limited by polling intervals, pod startup time, database capacity, network capacity, and trigger thresholds. Recent research on Service Level Objective (SLO)-aware autoscaling also shows that adaptive scaling remains an active area of cloud-native performance research [16] [17]. This study therefore compares static scaling, HPA, and KEDA using the same Moodle workload and the same performance indicators.

2. Research Gap and Contributions

Prior studies on e-learning and online assessment have examined adoption challenges, assessment quality, and educational technology use, while cloud-native literature has described Kubernetes, autoscaling algorithms, and elastic resource management [6]-[11] [16]-[23]. However, empirical evidence remains limited for Moodle-based online examination workloads that combine synchronized user actions, high reliability expectations, and containerized infrastructure. This gap matters because a platform that appears responsive under ordinary LMS activity may degrade sharply when examination users arrive in a concentrated window.

This study addresses that gap by comparing static scaling, Kubernetes HPA, and KEDA under the same Moodle examination workload across 500 to 70,000 concurrent connections. It evaluates average latency, tail latency, throughput, and error rate together because examination readiness depends on both responsiveness and reliability. The results show a practical trade-off: HPA protected a larger share of requests from failure, while KEDA produced faster successful responses and strong throughput under several high-load conditions. Presenting the metrics separately keeps each finding tied to its unit, workload level, and scaling strategy.

3. Related Work

3.1. E-Learning Infrastructure, Moodle, and Online Assessment

E-learning research identifies infrastructure, policy, technical capacity, learner access, and implementation support as recurring factors that influence online learning delivery [6]-[9]. These issues are relevant to online examination systems because reliability depends on both the application and the infrastructure that supports high-concurrency access. Online formative assessment literature also emphasizes that assessment quality depends on dependable online environments, timely feedback, interaction, and continuity of access [1].

Moodle was introduced as an open-source course management system designed around learning communities [4]. It has since become a widely used LMS for course delivery and assessment workflows. Official Moodle documentation and administrator guidance emphasize that production deployments require careful attention to web server configuration, database performance, caching, file storage, cron processing, and monitoring [5] [24]. These concerns align with the pre-

sent study because autoscaling Moodle application pods may not solve every bottleneck if the database, storage, ingress, or cache layer becomes saturated.

From a performance-engineering perspective, examination workloads should be evaluated as workload-specific scenarios rather than as generic web traffic. Computer systems performance analysis emphasizes the need to define workload, metrics, measurement procedure, and interpretation criteria before drawing conclusions [25] [26]. For this reason, the present study evaluates average response time, p95 latency, p99 latency, throughput, and error rate together. This helps avoid the misleading conclusion that a system is performing well merely because the requests that did succeed were fast.

3.2. Kubernetes, Container Orchestration, and Autoscaling

Kubernetes emerged from experience with large-scale container and cluster management systems and is now widely used for deploying and managing containerized workloads [10] [11] [27] [28]. A Kubernetes deployment can run multiple replicas of an application pod and expose them through services and ingress components. This architecture makes horizontal scaling practical for stateless or mostly stateless application tiers, including the web/PHP tier of a Moodle deployment when shared storage, database, and session handling are configured appropriately.

Autoscaling decisions are only as effective as the metrics and thresholds that guide them. Kubernetes HPA adjusts the number of pod replicas based on observed resource metrics or supported custom metrics [12] [13]. This approach can be stable because it responds to measured utilization, but it can lag sudden demand spikes if CPU or memory pressure appears after queues have already formed. Research surveys on cloud elasticity and autoscaling similarly show that reactive scaling is useful but sensitive to metric choice, thresholds, cooldown behavior, and provisioning delay [18]-[23].

3.3. KEDA and Event-Driven Autoscaling

KEDA is a Kubernetes-based event-driven autoscaler that scales workloads using event sources and external metrics [14] [15]. In contrast to basic resource-driven autoscaling, KEDA can connect scaling decisions to signals such as queue length, message backlog, Prometheus metrics, and other supported scaler sources [14]. For an examination workload, this is attractive because demand signals may appear before CPU utilization fully reflects the incoming pressure.

However, event-driven scaling still depends on trigger configuration, polling behavior, cooldown periods, pod readiness, image pull time, and downstream service capacity. In a Moodle examination environment, bottlenecks may include PHP workers, database connections, shared files, cache services, ingress configuration, network throughput, and the load generator itself. These factors make empirical evaluation necessary before adopting KEDA for high-stakes assessment workloads [14] [15] [20] [24].

3.4. Peak-Load and Reliability Considerations in Examination Systems

Scheduled assessment platforms experience concentrated demand because access is controlled by examination timetables and deadlines. The relevant engineering question is therefore not whether the application works under normal use, but how it behaves when many users perform similar actions within a short window. Public-facing examination services and institutional assessment platforms can both experience this pattern, even though their application workflows differ [29].

For high-stakes examinations, error rate is a direct reliability indicator because failed requests may prevent login, page access, answer saving, or final submission. Tail latency is also important because the slowest successful requests often represent the user experience of candidates at the edge of acceptable service. The present study therefore treats latency, throughput, and error rate as complementary indicators rather than competing measures.

4. Methodology

4.1. Research Design

The study used a controlled experimental design. The independent variable was the scaling strategy: static scaling, Kubernetes HPA, and KEDA. The dependent variables were average response time, p95 latency, p99 latency, throughput, and error rate. The purpose was not to introduce new Moodle functionality, but to evaluate how the same Moodle examination workload behaved under each scaling configuration.

The comparison used identical workload levels and the same measurement definitions across all strategies. This design supports direct comparison because changes in latency, throughput, and error rate can be interpreted against the scaling strategy rather than against different workloads or application features. Each workload level was executed four times for each scaling strategy. Reported values represent arithmetic means across repeated trials. The purpose of repetition was to reduce the impact of transient system variability and improve reliability of measurements.

4.2. Experimental Environment

The experiments were conducted on an on premises k3s Kubernetes cluster deployed on HPE Synergy 660 Gen10 infrastructure. The cluster consisted of one master node (16 CPU cores, 32 GB RAM), two worker nodes (each 4 CPU cores and 8 GB RAM), and a dedicated NFS storage server (4 CPU cores, 8 GB RAM). Moodle 5.2 LMS was deployed as a containerized application running on k3s version 1.34.5+k3s1, with MySQL 8.3 used as the backend database. Moodle application data were stored on a shared NFS-backed persistent volume. Prometheus and Grafana were used for monitoring, while Apache JMeter generated examination workloads which is commonly used for load, performance, and functional testing of web applications [30]. Moodle pods were distributed across worker nodes using Kubernetes topology-spread constraints.

4.3. Scaling Configurations

The static condition used two Moodle application pods and did not automatically add capacity when demand increased. This represents a common fixed-capacity deployment model in which administrators provision resources before an examination and rely on that capacity throughout the test window.

The HPA configuration used Kubernetes autoscaling/v2. Moodle pods were configured with resource requests of 250m CPU and 512 MiB memory and resource limits of 1000m CPU and 1024 MiB memory. HPA was configured with a minimum replica count of 2 and a maximum replica count of 25. Scaling decisions were based on both CPU and memory utilization with target utilization thresholds of 50% for each metric. Stabilization and scaling policies were configured to prevent rapid oscillation of replica counts during workload fluctuations. KEDA was implemented using an HTTPScaledObject. Scaling decisions were based on HTTP request rate, with a target value of five requests per second, a granularity interval of 1 s, and a measurement window of 1 minute. The configuration used a scale-down period of 180 s, a minimum replica count of 0, and a maximum replica count of 25 replicas. Readiness and startup probes were inherited from the Moodle deployment and were used to ensure traffic was directed only to healthy pods.

4.4. Workload Generation and Test Procedure

Apache JMeter was used to emulate concurrent Moodle users. In this study, “concurrent connections” refers to the number of simultaneous JMeter virtual users (threads) executing Moodle examination transactions. The workload model represented a typical student examination journey consisting of four phases: (i) login and dashboard access, (ii) quiz start, (iii) question interaction and answer saving, and (iv) final submission. Workload levels were configured at 500, 1000, 2500, 5000, 10,000, 20,000, 30,000, 40,000, 50,000, 60,000, and 70,000 concurrent users. Ramp-up periods varied from 5 s for 500 users to 360 s for 70,000 users. Each workload level was executed four times under identical conditions, and the reported values represent the mean across repeated trials. The same HTTP connection reuse settings, request mix, and workload profile were applied to all scaling strategies to preserve comparability.

4.5. Metrics and Data Analysis

Average response time, p95 latency, and p99 latency were calculated using successful requests reported by Apache JMeter. Error rate was computed as the percentage of failed or timed-out requests relative to total requests (**Table 1**). Failed requests included HTTP errors and timeout events recorded by the load-testing framework. Consequently, latency metrics should be interpreted as the performance of successful requests, while reliability is captured separately through the error-rate metric.

To support reproducibility of the results, all the deployment manifests, autoscaler configurations, workload definitions, resource requests and limits, and test parameters are provided below.

Table 1. Experimental variables, units, and measurement focus.

Element	Description	Value/setting and units
Independent variable	Scaling strategy	Static scaling; Kubernetes HPA; KEDA
Workload levels	Concurrent connections	500 to 70,000 concurrent connections
Latency metrics	Response-time indicators	Average, p95, and p99 response times in milliseconds (ms)
Throughput	Completed request rate	Requests per second (req/s)
Error rate	Failed or timed-out requests	Percentage of total requests (%)
Environment	On-premises Kubernetes Moodle deployment	HPE Synergy 660 Gen10 servers; k3s v1.34.5+k3s1; Moodle 5.2 LMS; MySQL 8.3

Kubernetes HPA deployment

```
root@master1:~/moodle# more moodle-hpa.yaml
```

```
apiVersion: autoscaling/v2
```

```
kind: HorizontalPodAutoscaler
```

```
metadata:
```

```
  name: moodle-hpa
```

```
  namespace: moodle
```

```
spec:
```

```
  scaleTargetRef:
```

```
    apiVersion: apps/v1
```

```
    kind: Deployment
```

```
    name: moodle
```

```
  minReplicas: 2
```

```
  maxReplicas: 25
```

```
  metrics:
```

```
  - type: Resource
```

```
    resource:
```

```
      name: cpu
```

```
      target:
```

```
        type: Utilization
```

```
        averageUtilization: 50
```

```
  - type: Resource
```

```
    resource:
```

```
      name: memory
```

```
      target:
```

```
        type: Utilization
```

```
        averageUtilization: 50
```

```
root@master1:~/moodle#
```

```
root@master1:~/moodle#
```

KEDA deployment

```
root@master1:~/moodle# more moodle-keda.yaml
apiVersion: http.keda.sh/v1alpha1
kind: HTTPScaledObject
metadata:
  name: moodle-http-scaledobject
  namespace: moodle
spec:
  hosts:
  - mydomain.domain.com
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: moodle
    service: moodle
    port: 80
  replicas:
    min: 0
    max: 25
  scalingMetric:
    requestRate:
      targetValue: 5
      granularity: 1s
      window: 1m
    scaledownPeriod: 180
root@master1:~/moodle#
```

JMeter Workload timing**Table 2.** JMeter workload timings.

Concurrent connections	Time (s)
500	5
1000	10
2500	15
5000	30
10,000	60
20,000	120
30,000	150
40,000	200
50,000	250
60,000	300
70,000	360

5. Results

The results are presented as a continuous full-range analysis from 500 to 70,000 concurrent connections (**Table 2**).

5.1. Average Response Time

At 500 concurrent connections, all strategies were relatively close: static scaling recorded 328 ms, HPA 283 ms, and KEDA 389 ms (**Table 3**). The first major separation occurred at 1000 concurrent connections, where static scaling rose to 5419 ms while HPA and KEDA remained at 253 ms and 347 ms, respectively. Static scaling then reached 10,347 ms at 2500 concurrent connections, showing that fixed capacity began to queue requests early in the workload curve.

Table 3. Average response time by workload (ms).

Concurrent connections	Static scaling (ms)	HPA (ms)	KEDA (ms)
500	328	283	389
1000	5419	253	347
2500	10,347	225	338
5000	7085	275	252
10,000	6496	2476	412
20,000	4277	1964	791
30,000	3112	2127	941
40,000	3159	2041	1419
50,000	2788	2071	1576
60,000	3133	2119	1776
70,000	2655	2122	1604

HPA maintained low average response time through 5000 concurrent connections, reaching 275 ms at that point, but rose to 2476 ms at 10,000 concurrent connections. KEDA recorded 252 ms at 5000 concurrent connections, 412 ms at 10,000 concurrent connections, and 1604 ms at 70,000 concurrent connections (**Figure 1**). These values show that KEDA delivered the fastest average response time among successful requests under several high-load conditions. However, this must be read together with the error-rate results because a fast average response time is not sufficient if many requests fail.

5.2. Tail Latency

The p95 and p99 results show the slower end of successful request performance (**Figure 2** & **Figure 3**). Static scaling deteriorated immediately after the first work-

load increase: p95 latency rose from 741 ms at 500 concurrent connections to 10,689 ms at 1000 concurrent connections and 16,803 ms at 2500 concurrent connections. It remained above 10,000 ms across most of the upper range, demonstrating persistent tail-latency instability.

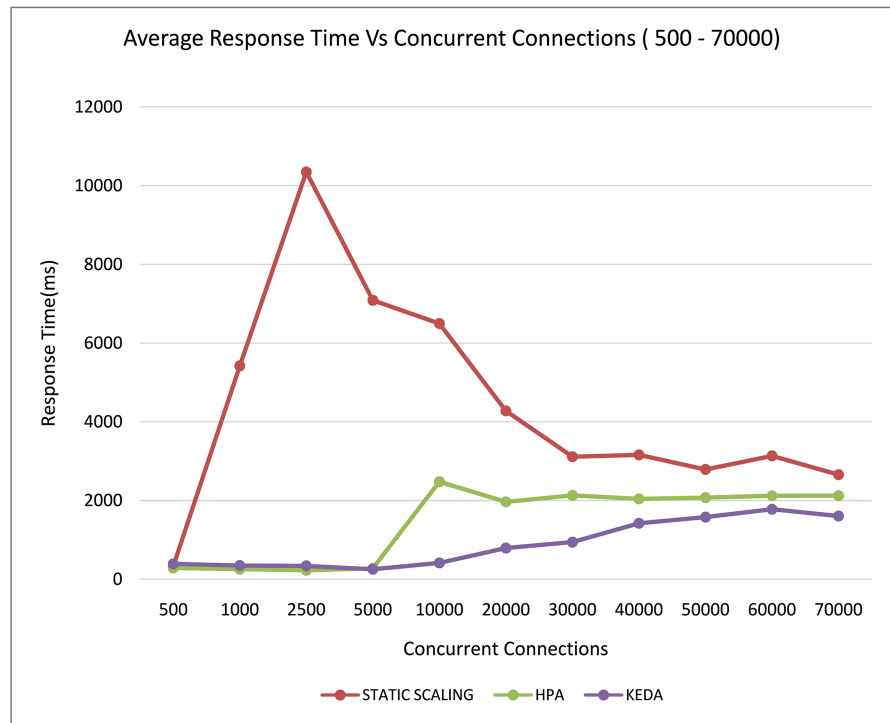


Figure 1. Average response time for static scaling, HPA, and KEDA across workloads.

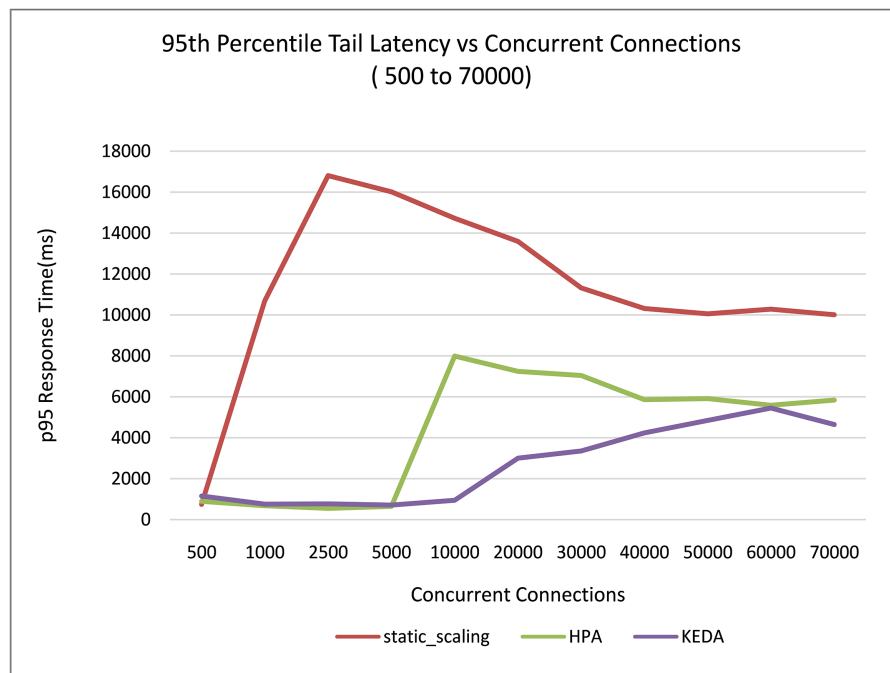


Figure 2. 95th percentile latency across workloads.

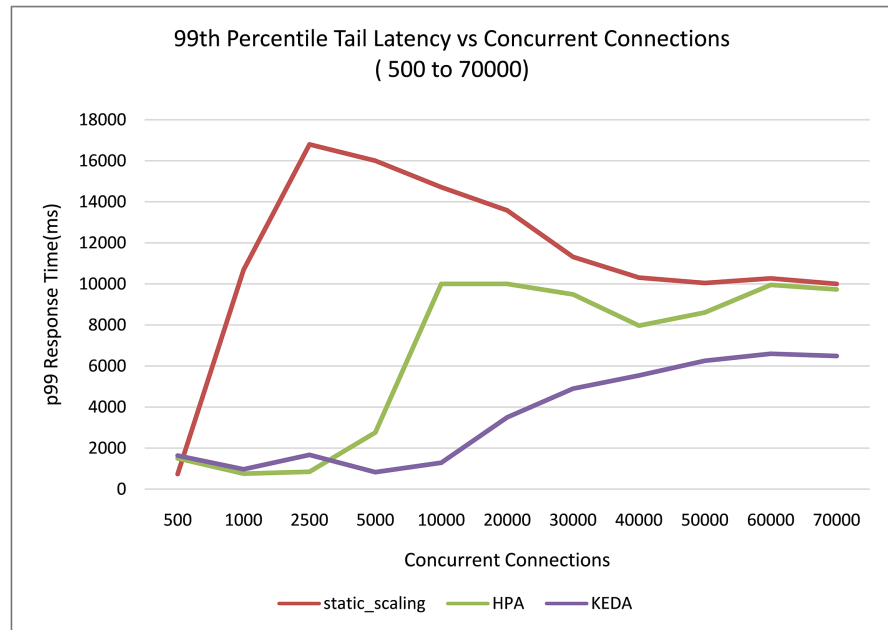


Figure 3. 99th percentile latency across workloads.

HPA performed strongly at lower workloads but increased sharply at higher loads. At 10,000 concurrent connections, HPA reached 7987 ms on p95 and 10,005 ms on p99, whereas KEDA recorded 939 ms and 1287 ms, respectively (**Table 4** & **Table 5**). From 20,000 to 70,000 concurrent connections, KEDA kept successful tail requests faster than HPA. Nevertheless, the lower KEDA tail latency should not be interpreted alone because its error rate was substantially higher than HPA's at the same high-load points.

Table 4. 95th percentile latency by workload (ms).

Concurrent connections	Static scaling p95 (ms)	HPA p95 (ms)	KEDA p95 (ms)
500	741	888	1152
1000	10,689	678	755
2500	16,803	545	769
5000	16,011	640	708
10,000	14,716	7987	939
20,000	13,587	7243	3001
30,000	11,320	7040	3347
40,000	10,313	5856	4235
50,000	10,051	5912	4843
60,000	10,276	5587	5447
70,000	10,005	5837	4639

Table 5. 99th percentile latency by workload (ms).

Concurrent connections	Static scaling p99 (ms)	HPA p99 (ms)	KEDA p99 (ms)
500	741	1501	1634
1000	10,689	758	966
2500	16,803	850	1669
5000	16,011	2761	828
10,000	14,716	10,005	1287
20,000	13,587	10,004	3498
30,000	11,320	9491	4897
40,000	10,313	7970	5540
50,000	10,051	8609	6258
60,000	10,276	9957	6598
70,000	10,005	9739	6489

5.3. Throughput

Throughput increased for all three strategies as the workload increased, but autoscaling strategies separated from static scaling early. At 1000 concurrent connections, static scaling reached 41.99 req/s, compared with 65.21 req/s for HPA and 64.27 req/s for KEDA (Table 6). At larger workloads, KEDA recorded the highest throughput at most points, including 323.17 req/s at 10,000 concurrent connections, 756.46 req/s at 40,000 concurrent connections, and 777.74 req/s at 60,000 concurrent connections (Figure 4).

Table 6. Throughput by workload (req/s).

Concurrent connections	Static scaling (req/s)	HPA (req/s)	KEDA (req/s)
500	32.63	32.53	32.08
1000	41.99	65.21	64.27
2500	89.76	157.54	161.12
5000	120.39	160.03	163.93
10,000	287.51	299.02	323.17
20,000	423.30	478.72	484.02
30,000	396.07	489.26	484.34
40,000	643.14	714.55	756.46
50,000	736.54	746.44	773.78
60,000	692.38	757.89	777.74
70,000	733.19	779.23	771.16

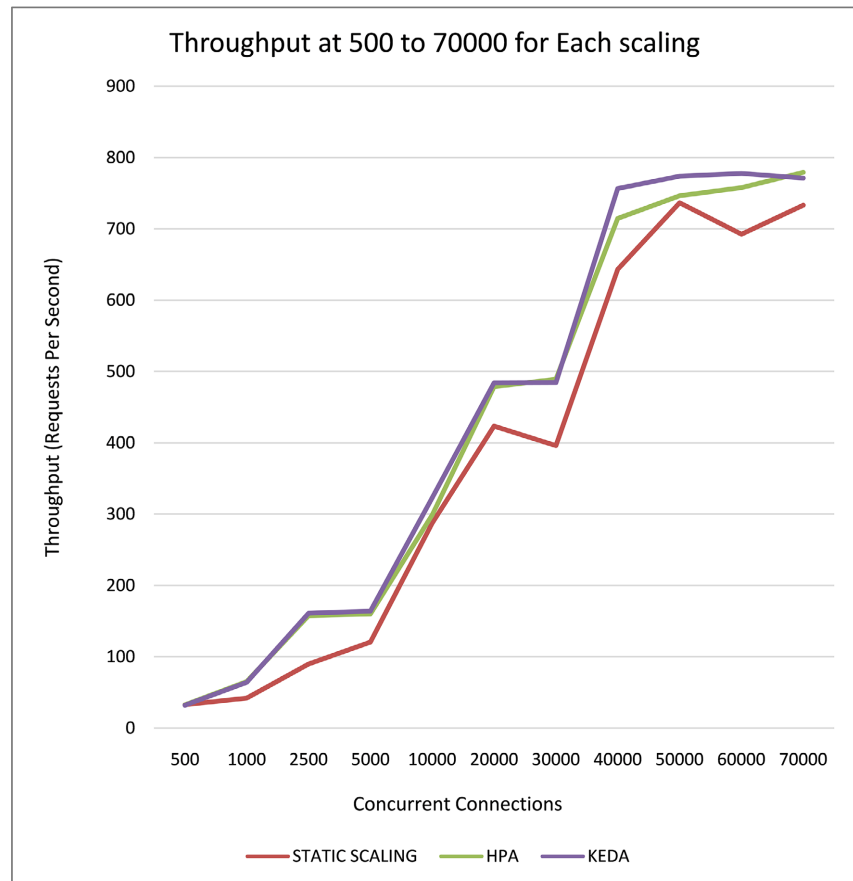


Figure 4. Throughput for static scaling, HPA, and KEDA across workloads.

Throughput must be interpreted with error rate because a system can appear to process many requests per second while still failing a large proportion of total attempts. For example, KEDA produced the highest throughput at 10,000 concurrent connections, but its error rate at the same point was 93.725%. This means that its throughput advantage reflected the rate of completed requests among a workload in which many requests failed.

5.4. Error Rate

Error rate is the most important metric for examination continuity because failed requests can prevent candidates from logging in, loading pages, saving answers, or submitting work (**Figure 5**). Static scaling recorded 0% error at 500 concurrent connections, 13.775% at 1000 concurrent connections, 81% at 2500 concurrent connections, and 92% at 5000 concurrent connections. From 10,000 to 70,000 concurrent connections, its error rate remained between 97.95% and 99.7%, making it unsuitable for the upper workload range (**Table 7**).

Across the four experimental trials, HPA consistently produced lower observed error rates than KEDA and static scaling under the tested workload conditions. It maintained 0% error from 500 through 5000 concurrent connections. At 10,000 concurrent connections, HPA's error rate increased to 61.675%, which is severe,

but it was still lower than KEDA's 93.725% and static scaling's 98.4%. HPA remained the lowest-error strategy at every larger workload. KEDA performed well at the beginning of the curve, but its error rate rose to 14.5% at 2500 concurrent connections, 29.575% at 5000 concurrent connections, and 93.725% at 10,000 concurrent connections.

Table 7. Error rate by workload (%).

Concurrent connections	Static scaling (%)	HPA (%)	KEDA (%)
500	0	0	0.50
1000	13.78	0	0.15
2500	81	0	14.50
5000	92	0	29.57
10,000	98.40	61.67	93.72
20,000	98.58	83.62	98.15
30,000	98.75	87.03	97.38
40,000	99.60	91.28	99.40
50,000	97.95	94.55	99.53
60,000	99.70	93.88	99.28
70,000	98.78	96.45	99.72

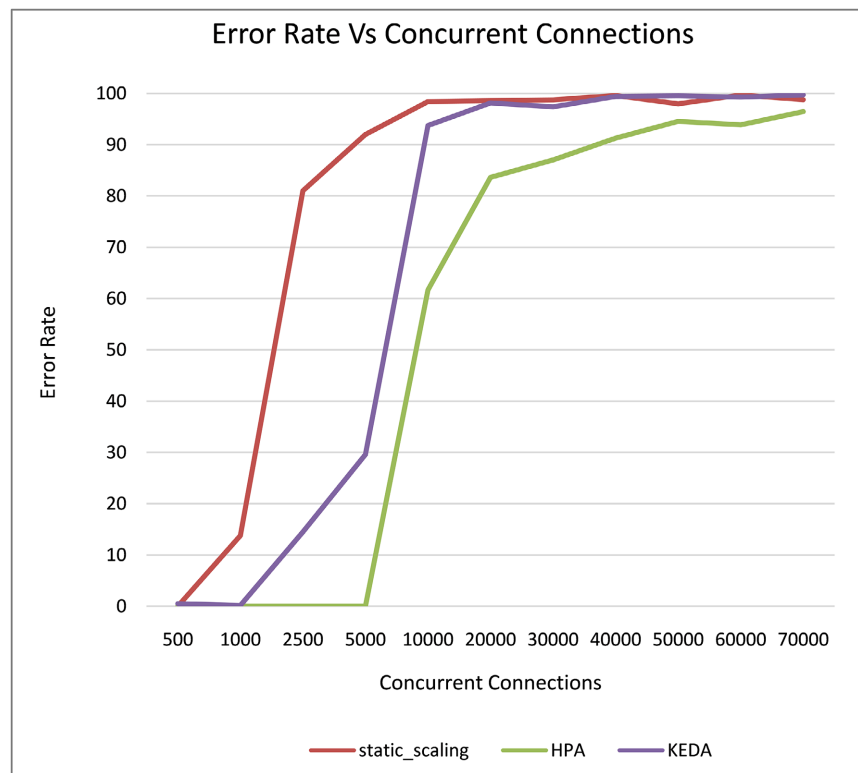


Figure 5. Error rate for static scaling, HPA, and KEDA across workloads.

A sample interpretation illustrates the trade-off. At 10,000 concurrent connections, KEDA had a much lower average response time than HPA, 412 ms compared with 2476 ms. If response time alone were considered, KEDA would appear superior. However, KEDA failed 93.725% of requests at that workload, while HPA failed 61.675%. Therefore, HPA served a larger share of the workload successfully, even though the successful HPA requests were slower. In an examination setting, that distinction is critical because a slower successful submission is preferable to a fast failure.

6. Discussion

The results clarify the practical difference between fixed provisioning and autoscaling. Static scaling failed not only at extreme load but also at relatively early points in the workload curve. In an examination setting, this would correspond to delayed logins, stalled page loads, failed quiz access, or failed submissions. This supports the performance-engineering principle that systems must be evaluated under representative workload and metric conditions rather than under ordinary functional use alone [25] [26] [30].

HPA is the most defensible option when the objective is continuity of service. Its zero-error performance up to 5,000 concurrent connections and lowest error rate at every larger workload show that resource-based autoscaling protected a larger share of requests than KEDA in this experiment. The trade-off is that HPA's successful requests became slower under high load, particularly in p95 and p99 latency after 10,000 concurrent connections. This aligns with the known behavior of reactive autoscaling: it can improve resilience, but it may still react after workload pressure has already accumulated [12] [13] [18]–[23].

KEDA should not be interpreted as unsuccessful overall. It demonstrated strong responsiveness among requests that were completed, especially after the workload entered the high-load region. Its lower high-load latency and slightly stronger throughput suggest that event-driven scaling can keep successful requests efficient. The problem is that KEDA's reliability deteriorated earlier in this configuration. Possible causes include trigger thresholds, polling interval, cooldown settings, pod startup time, readiness behavior, database saturation, ingress limits, or another downstream bottleneck. These factors require further measurement rather than assumption because KEDA scaling behavior depends directly on event sources, external metrics, and scaler settings [14] [15].

For institutional deployment, the choice should be guided by service objectives. If the goal is to maximize the probability that candidates can complete examination actions, HPA is the safer option in this dataset. If the goal is to maximize responsiveness among successful requests and throughput, KEDA becomes attractive, but only with careful tuning and a non-zero warm baseline. A hybrid design that combines HPA stability with KEDA demand signals is a strong candidate for future evaluation, alongside predictive and SLO-aware autoscaling approaches [16] [17] [20]–[23].

7. Threats to Validity

Internal validity is limited by the specific Moodle deployment, Kubernetes environment, database configuration, network, and autoscaler settings used in the experiment. Any unobserved bottleneck, including database connection limits, disk input/output (I/O), PHP worker limits, ingress configuration, shared storage behavior, or JMeter generator limits, could affect the measured results.

Construct validity is limited by the workload configuration from 500 to 70,000 concurrent connections and by the selected metrics. Although average response time, p95, p99, throughput, and error rate are appropriate for this study, additional metrics such as pod startup time, CPU utilization, memory utilization, database wait time, queue length, and autoscaler event timelines would strengthen causal interpretation.

External validity is limited because the experiment used an on-premises Kubernetes cluster. Results may differ in public cloud environments, with different pod startup times, managed load balancers, storage systems, database architectures, network latency, or cluster autoscaling behavior. Conclusion validity is limited because the study compares the observed configurations of static scaling, HPA, and KEDA; it does not prove that HPA is always superior to KEDA under all trigger settings or infrastructure designs.

Although four repeated trials were conducted for each workload level, the current study reports mean values and do not include formal confidence intervals or hypothesis-testing statistics. Consequently, the findings should be interpreted as comparative experimental observations within the evaluated environment rather than evidence of universal superiority of one autoscaling strategy over another.

8. Recommendations

Institutions should avoid relying on static Moodle deployments for high-stakes online examinations with synchronized access patterns. Static capacity may be sufficient for ordinary LMS use, but this experiment shows that it can fail early when many users access the examination workflow at the same time.

HPA should be considered the baseline autoscaling strategy where examination continuity is the primary objective. KEDA should be evaluated through institution-specific load testing before production use because it may require lower trigger thresholds, shorter polling intervals, adjusted cooldown behavior, pre-warmed minimum replicas, readiness probes, database-side capacity tuning, and careful ingress configuration before it can combine its responsiveness advantage with acceptable reliability [12]–[15] [24].

Load testing should be treated as a pre-examination operational requirement. Raw JMeter logs, Kubernetes manifests, autoscaler events, resource metrics, database metrics, and monitoring dashboards should be archived for reproducibility. Future work should evaluate hybrid HPA-KEDA autoscaling, predictive pre-scaling before scheduled examination starts, database and cache scaling, and repeated trials across public cloud and on-premises environments. More advanced adap-

tive approaches, including SLO-aware autoscaling, should also be compared against practical Kubernetes autoscaling patterns in education workloads [16] [17] [20]–[23].

9. Conclusion

This study evaluated static scaling, Kubernetes HPA, and KEDA for a Moodle-based online examination deployment. Static scaling broke down early in both latency and reliability, making it unsuitable for demanding synchronized examination workloads. HPA provided the strongest reliability profile and demonstrated the most favorable reliability characteristics within the tested environment and configuration. KEDA provided stronger responsiveness and throughput among successful requests at several high-load points, but its error rate increased earlier and more sharply than HPA's in this configuration. The central finding is therefore not that one autoscaler is universally best, but that autoscaling strategy must match the service objective and must be validated under realistic examination load before production use.

Author Contributions

Robert Achang'a conceived the research topic, developed the study content, designed the experimental setup, and carried out the experiments. Andrew Kahonge contributed to the design and execution of the workload tests, analysis of the experimental results, and overall research supervision. Elisha Abade provided guidance on the research methodology and contributed to the overall supervision of the study.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

- [1] Gikandi, J.W., Morrow, D. and Davis, N.E. (2011) Online Formative Assessment in Higher Education: A Review of the Literature. *Computers & Education*, **57**, 2333–2351. <https://doi.org/10.1016/j.compedu.2011.06.004>
- [2] Siemens, G. (2013) Learning Analytics: The Emergence of a Discipline. *American Behavioral Scientist*, **57**, 1380–1400. <https://doi.org/10.1177/0002764213498851>
- [3] Ifenthaler, D. and Yau, J.Y. (2020) Utilising Learning Analytics to Support Study Success in Higher Education: A Systematic Review. *Educational Technology Research and Development*, **68**, 1961–1990. <https://doi.org/10.1007/s11423-020-09788-z>
- [4] Dougiamas, M. and Taylor, P. (2003) Moodle: Using Learning Communities to Create an Open Source Course Management System. ED-MEDIA 2003—World Conference Educational Multimedia, Hypermedia & Telecommunications, Honolulu, 23–28 July 2003, 171–178.
- [5] Moodle (2026) Moodle Documentation. <https://docs.moodle.org/>
- [6] Tarus, J.K., Gichoya, D. and Muumbo, A. (2015) Challenges of Implementing E-Learning in Kenya: A Case of Kenyan Public Universities. *The International Review*

- of Research in Open and Distributed Learning*, **16**, 120-141.
<https://doi.org/10.19173/irrodl.v16i1.1816>
- [7] Kibuku, R.N., Ochieng, P.D.O. and Wausi, P.A.N. (2020) E-Learning Challenges Faced by Universities in Kenya: A Literature Review. *Electronic Journal of e-Learning*, **18**, 150-161. <https://doi.org/10.34190/ejel.20.18.2.004>
- [8] Nyerere, J. (2016) Delivery of Open, Distance, and E-Learning in Kenya. *International Journal of Advanced Research in Education & Technology*, **3**, 167-171.
- [9] Deya, F. (2025) Accessing Online and Remote Learning Platforms in Kenyan Remote Areas during COVID-19 Pandemic. David Publisher.
- [10] Burns, B., Beda, J., Hightower, K. and Evenson, L. (2019) Kubernetes: Up and Running. 2nd Edition, O'Reilly Media.
- [11] Cloud Native Computing Foundation (2026) Kubernetes Documentation. <https://kubernetes.io/docs/>
- [12] Kubernetes (2026) Horizontal Pod Autoscaling. <https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/>
- [13] Kubernetes (2026) Resource Metrics Pipeline. <https://kubernetes.io/docs/tasks/debug/debug-cluster/resource-metrics-pipeline/>
- [14] KEDA (2026) Kubernetes Event-Driven Autoscaling Documentation. <https://keda.sh/docs/>
- [15] Red Hat (2019) Celebrating KEDA 1.0: Providing an Event-Driven Scale Capability for Any Container Workload. <https://www.redhat.com/>
- [16] Mayerhofer, R., Morichetta, A., Furutanpey, A. and Dustdar, S. (2025) HPAQT: Adaptive and Interpretable High-Level SLO-Aware Autoscaling with Reinforcement Learning. *Proceedings of the 18th IEEE/ACM International Conference on Utility and Cloud Computing*, France, 1-4 December 2025, 1-9.
<https://doi.org/10.1145/3773274.3774274>
- [17] Rodriguez, M.A. and Buyya, R. (2018) Container-Based Cluster Orchestration Systems: A Taxonomy and Future Directions. *Software: Practice and Experience*, **49**, 698-719. <https://doi.org/10.1002/spe.2660>
- [18] Lorido-Botran, T., Miguel-Alonso, J. and Lozano, J.A. (2014) A Review of Auto-Scaling Techniques for Elastic Applications in Cloud Environments. *Journal of Grid Computing*, **12**, 559-592. <https://doi.org/10.1007/s10723-014-9314-7>
- [19] Ali-Eldin, A., Kihl, M., Tordsson, J. and Elmroth, E. (2012) Efficient Provisioning of Bursty Scientific Workloads on the Cloud Using Adaptive Elasticity Control. *Proceedings of the 3rd Workshop on Scientific Cloud Computing*, Delft, 18 June 2012, 31-40. <https://doi.org/10.1145/2287036.2287044>
- [20] Mao, M. and Humphrey, M. (2011) Auto-Scaling to Minimize Cost and Meet Application Deadlines in Cloud Workflows. *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, Seattle, 12-18 November 2011, 1-12. <https://doi.org/10.1145/2063384.2063449>
- [21] Taherizadeh, S. and Stankovski, V. (2018) Dynamic Multi-Level Auto-Scaling Rules for Containerized Applications. *The Computer Journal*, **62**, 174-197.
<https://doi.org/10.1093/comjnl/bxy043>
- [22] Herbst, N.R., Kounev, S. and Reussner, R. (2013) Elasticity in Cloud Computing: What It Is, and What It Is Not. *Proceeding of 10th International Conference Automatic Computing (ICAC)*, San Jose, 26-28 June 2013, 23-27.
- [23] Qu, C., Calheiros, R.N. and Buyya, R. (2018) Auto-Scaling Web Applications in

- Clouds: A Taxonomy and Survey. *ACM Computing Surveys*, **51**, 1-33.
<https://doi.org/10.1145/3148149>
- [24] Moodle (2026) Performance Recommendations. <https://docs.moodle.org/>
 - [25] Jain, R. (1991) The Art of Computer Systems Performance Analysis. Wiley.
 - [26] Harchol-Balter, M. (2013) Performance Modeling and Design of Computer Systems. Cambridge University Press. <https://doi.org/10.1017/cbo9781139226424>
 - [27] Verma, A., Pedrosa, L., Korupolu, M., Oppenheimer, D., Tune, E. and Wilkes, J. (2015) Large-Scale Cluster Management at Google with Borg. *Proceedings of the Tenth European Conference on Computer Systems*, Bordeaux, 21-24 April 2015, 1-17. <https://doi.org/10.1145/2741948.2741964>
 - [28] Burns, B., Grant, B., Oppenheimer, D., Brewer, E. and Wilkes, J. (2016) Borg, Omega, and Kubernetes. *Communications of the ACM*, **59**, 50-57.
<https://doi.org/10.1145/2890784>
 - [29] Kemibaro, M. (2023) Why Does the KNEC Examinations Results Portal Crash Every Year? Medium. <https://medium.com/>
 - [30] Apache Software Foundation (2026) Apache JMeter User Manual.
<https://jmeter.apache.org/usermanual/>