International Journal on Science and Technology (IJSAT)

E-ISSN: 2229-7677 ● Website: www.ijsat.org ● Email: editor@ijsat.org

Performance Evaluation of Throughput Under Dynamic Workload Conditions in Kubernetes Clusters

Rajani Gatta

rajanigatta@gmail.com

Abstract:
The increasing adoption of cloud-native technologies and distributed computing has significantly transformed the way modern applications are developed, deployed, and managed. Kubernetes has emerged as a widely adopted container orchestration platform that enables automated deployment, scaling, load balancing, and management of containerized applications across distributed computing environments. Consequently, an increase in the number of application replicas does not necessarily guarantee a proportional improvement in throughput. Furthermore, correlating throughput with complementary performance indicators such as response latency, CPU utilization, memory consumption, and request completion time can provide a more comprehensive understanding of system behaviour. This research investigates the performance of Kubernetes clusters under dynamic workload conditions, with particular emphasis on throughput measurement and analysis. The study considers varying workload intensities and resource configurations to examine how changes in application demand influence request-processing capacity. Experimental evaluations are designed to identify throughput variations, investigate resource-related bottlenecks, and analyse the relationship between throughput and other system performance indicators. The primary objective of this research is to evaluate the throughput behaviour of Kubernetes-based distributed applications under dynamic workload conditions and identify the factors that influence their processing efficiency. The findings are expected to provide insights into the performance limitations of containerized applications and support more effective resource management and workload optimization strategies in Kubernetes environments.

Keywords: Throughput, Kubernetes, Dynamic Workloads, Container Orchestration, Distributed Systems, Performance Evaluation, Resource Utilization, Horizontal Pod Autoscaling, Scalability, Load Balancing.

INTRODUCTION

The increasing adoption of cloud-native technologies and distributed computing has transformed the development, deployment, and management of modern applications. Kubernetes has emerged as a widely adopted container orchestration platform, providing automated deployment, scaling, load balancing, and resource management capabilities for containerized applications. Its ability to manage distributed workloads dynamically makes it suitable for applications experiencing fluctuating traffic patterns and varying computational demands. However, maintaining consistent application performance under dynamic workload conditions remains a significant challenge. Throughput is a fundamental performance metric used to evaluate the processing capability of distributed systems. It represents the number of

requests successfully processed within a specified time interval. In Kubernetes clusters, throughput is influenced by several factors, including CPU utilization, memory availability, pod configuration, replica count, network communication overhead, and resource allocation policies. Variations in workload intensity can affect these factors, resulting in throughput fluctuations and potential performance degradation. Dynamic workloads introduce additional complexity in resource management. During periods of increased request volumes, application pods may experience resource contention, increased response latency, and reduced processing capacity. Although Kubernetes provides Horizontal Pod Autoscaling (HPA) to adjust application replicas based on resource utilization, scaling decisions based primarily on CPU or memory consumption may not directly reflect actual throughput requirements. Furthermore, increasing replica counts does not necessarily guarantee proportional throughput improvements due to scheduling overhead, network communication, and infrastructure constraints. This research investigates the performance of Kubernetes clusters under dynamic workload conditions, emphasizing throughput measurement and analysis. The study evaluates application performance across different workload intensities, replica configurations, and resource allocation settings. Throughput is analysed alongside supporting metrics such as response latency, CPU utilization, and memory consumption to identify performance bottlenecks and scalability limitations. The primary objective is to understand how dynamic workload variations influence throughput and identify the factors affecting request-processing efficiency. The experimental findings aim to provide insights into Kubernetes performance characteristics and support effective resource management strategies for distributed applications.

LITERATURE REVIEW

The performance evaluation of distributed systems has received considerable attention due to the increasing demand for efficient resource utilization, predictable response times, and improved request-processing capacity. Modern distributed computing environments involve multiple interconnected nodes that execute application workloads concurrently. The overall performance of these systems depends on several factors, including CPU utilization, memory availability, workload distribution, network communication, transaction processing, and resource allocation policies.

Kubernetes-based environments introduce additional complexity because application performance is influenced by pod scheduling, replica configurations, and dynamic workload variations. Throughput is an important performance indicator for evaluating the processing capacity of these environments, particularly when applications experience fluctuating request volumes. Previous research has investigated several related aspects of distributed resource optimization, concurrency management, telemetry analysis, replication, and adaptive resource allocation, providing a foundation for evaluating throughput under dynamic workload conditions.

Bandaru [1] investigated adaptive node utilization optimization in distributed systems, emphasizing the importance of distributing computational workloads efficiently across available nodes. Uneven resource utilization can result in resource concentration and create performance bottlenecks, even when sufficient computational capacity is available across the infrastructure. This aspect is particularly relevant to Kubernetes environments, where pod placement and workload distribution influence individual worker-node utilization. Similarly, Bondugula [2] examined conflict-rate reduction in scalable distributed

database systems.

Transaction conflicts can increase processing overhead, generate additional execution delays, and reduce effective transaction throughput. Managing such conflicts is important for maintaining processing efficiency in distributed environments where multiple operations execute concurrently. Bandaru [3] investigated predictable lock coordination for scalable multi-node transactional systems. The research addresses coordination requirements during concurrent transaction execution across multiple nodes. Inefficient lock coordination can increase waiting time and restrict the number of operations completed within a specified interval. These studies provide useful perspectives on resource utilization, processing conflicts, and coordination overhead in distributed systems.

Sambandam [4] investigated multimodal observability for input/output bottleneck detection. The research highlights the importance of analysing multiple operational indicators to identify performance constraints in distributed computing environments. Application performance may be restricted by input/output operations even when sufficient CPU and memory resources are available. Identifying such bottlenecks is essential for understanding throughput variations because storage and communication delays can affect overall request-processing capacity. Chuppala and Purnachandra Rao [5] investigated Hadoop Distributed File System write operations, focusing on distributed data-writing activities. Write operations involve coordination and communication among distributed storage components, which can influence processing performance.

Their work provides a relevant perspective on the relationship between distributed storage operations and resource consumption. Jinka and Purnachandra Rao [6] examined multivariate linear regression analysis for Sonatype Nexus repository space management. Their research emphasizes the relevance of analysing multiple operational variables to understand storage utilization patterns and resource requirements. Although the study focuses on repository space management, its analytical approach is relevant to understanding relationships among resource parameters in distributed computing environments.

Bandaru [7] investigated transaction batching for low-latency commit processing in distributed systems. Transaction batching can reduce the overhead associated with individual commit operations by processing multiple transactions together. However, batching configurations may influence transaction completion time and resource consumption. This highlights the importance of considering processing efficiency and latency when evaluating distributed application throughput. Bondugula [8] examined lock-based concurrency control in database systems, addressing the role of locking mechanisms in maintaining consistency during concurrent transaction execution. Excessive lock contention can increase waiting time and reduce the number of transactions completed within a given interval.

These considerations are relevant to distributed applications where concurrent request processing directly influences system performance. Bandaru [9] investigated cluster performance improvement through the reduction of response-time variance. Response-time variations may indicate workload imbalance, resource contention, or inconsistent processing conditions across distributed nodes. Maintaining predictable response behaviour is particularly important for applications operating under fluctuating request volumes.

Bondugula [10] examined transaction management under snapshot isolation using adaptive timeouts. Adaptive timeout mechanisms provide a means of managing transaction execution delays under changing system conditions. Such mechanisms are relevant to distributed processing environments where transaction delays can affect application responsiveness and processing efficiency. Bandaru [11] investigated latency-aware data partitioning techniques for distributed systems. The research emphasizes the importance of data distribution decisions in controlling access latency and improving distributed processing efficiency. Inefficient partitioning can increase communication requirements and introduce additional processing overhead when requests access data distributed across multiple nodes. This is relevant to Kubernetes-based applications because distributed microservices frequently communicate with other services and data repositories. Bondugula [12] examined deadlock management in distributed databases using Serializable Snapshot Isolation. Deadlock management is important for maintaining efficient transaction execution because unresolved conflicts may result in repeated processing, waiting time, and unnecessary resource consumption. Bandaru [13] investigated lightweight runtime conflict detection for CPU-efficient transaction processing. Runtime conflict detection can help identify conflicting operations during concurrent execution while limiting additional computational overhead.

These findings highlight the relationship between CPU consumption, concurrency management, and transaction-processing capacity. Bondugula [14] examined transaction commit optimization using Thomas's Write Rule. The research addresses the management of write operations to improve transaction execution efficiency. Efficient commit processing is important in distributed environments because transaction coordination and write-operation overhead can influence overall processing capacity.

Sambandam [15] investigated adaptive replication for low-latency distributed clusters. Replication is an important mechanism for improving service availability and distributing application workloads across multiple instances. However, replication strategies require careful consideration of resource consumption, synchronization overhead, and request distribution. Increasing the number of replicas does not necessarily guarantee proportional throughput improvement because the available computational resources and workload distribution efficiency also influence processing capacity.

Bandaru [16] examined correlated telemetry-driven throughput analysis in distributed pipelines. The research emphasizes the importance of analysing system telemetry to understand throughput variations and identify performance-related bottlenecks. Correlating multiple system indicators provides better visibility into the relationship between workload execution and infrastructure behaviour. This is particularly relevant to Kubernetes environments, where CPU consumption, memory pressure, network activity, and pod-level resource constraints can influence application performance.

Bondugula [17] investigated storage optimization in distributed environments using optimistic concurrency control. Optimistic concurrency control can reduce locking overhead in suitable workloads, although its effectiveness depends on transaction conflict characteristics. Storage optimization and concurrency management are important because inefficient storage access and transaction coordination can restrict application processing capacity. Bandaru [18] examined encoded data transfers for reducing raw network transfer time. Network communication overhead is an important consideration in distributed applications because data transfer delays can influence request-processing time and overall throughput.

Efficient data-transfer mechanisms can contribute to reducing communication overhead, particularly when application workloads involve frequent inter-node interactions. These studies collectively demonstrate that storage efficiency, network communication, and transaction processing are important factors affecting distributed application performance.

Pattan [19] investigated fault-tolerance optimization through AI-augmented replica management in distributed systems. Replica management influences system availability, workload distribution, and resource consumption. Effective replication strategies can support application continuity while maintaining processing efficiency across distributed nodes. Mylavarapu [20] examined adaptive CPU resource management in distributed systems, emphasizing the importance of adjusting computational resources according to application requirements. CPU allocation directly influences the processing capacity of application instances, particularly under high workload conditions.

This research is relevant to Kubernetes environments where application pods compete for available computational resources. Namala [21] investigated hybrid telemetry fusion for early detection of system-wide failures. Combining multiple telemetry indicators can provide a broader understanding of infrastructure behaviour and help identify potential operational problems before they significantly affect application performance. Such monitoring approaches are relevant to Kubernetes clusters, where correlating workload and resource metrics can support the identification of performance degradation under varying operating conditions.

The reviewed literature establishes that distributed application performance is influenced by multiple interdependent factors, including node utilization, transaction coordination, data partitioning, network communication, replication, CPU allocation, and system observability. Although these studies address individual aspects of distributed resource optimization and processing efficiency, the combined influence of workload intensity, pod replica configurations, and CPU and memory allocation on Kubernetes application throughput requires further systematic investigation. In particular, increasing the number of application replicas does not necessarily produce proportional throughput improvement because resource contention, scheduling overhead, network communication, and infrastructure limitations may restrict the effective processing capacity of a cluster.

Motivated by these research contributions, the present study focuses on evaluating throughput under dynamic workload conditions in Kubernetes clusters. The investigation considers different cluster configurations and workload intensities to understand their influence on application request-processing capacity. Throughput is treated as the primary performance parameter, while CPU utilization, memory consumption, response latency, and request completion time are considered supporting performance indicators. The experimental evaluation compares existing static resource allocation with the proposed adaptive resource allocation approach to examine CPU utilization behaviour and identify resource-related performance limitations. The study aims to provide insights into Kubernetes cluster performance, resource utilization efficiency, and scalability under dynamically changing workload conditions.

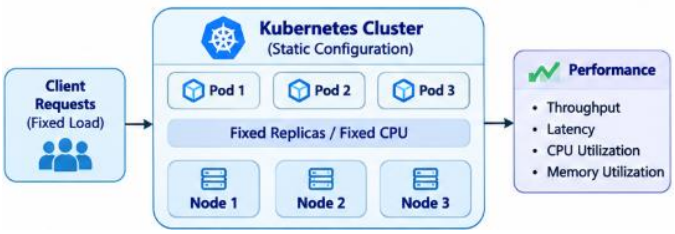


Fig. 1. **Existing Resource Allocation Architecture**

Fig. 1. The proposed framework illustrates the workflow for evaluating application throughput under dynamic workload conditions in Kubernetes clusters. The architecture consists of multiple Kubernetes worker nodes (Node 1, Node 2, Node 3, and Node N) connected to a Kubernetes Cluster Manager. The cluster manager handles scheduling, scaling, and resource management across the available nodes. A Workload Distributor generates and distributes application requests to simulate different workload conditions, including workload increases, decreases, and sudden traffic bursts. The framework continuously considers whether a workload change has occurred. When an increase or decrease in workload is detected, the application replica configuration is adjusted using Horizontal Pod Autoscaling (HPA). If no significant workload change is detected, the existing replica configuration is maintained. Both execution paths subsequently proceed to workload processing, where application requests are executed by the available pods.

After execution, performance metrics are collected, including throughput, response latency, CPU utilization, and memory consumption. These measurements provide the basis for analysing application behaviour under different workload intensities. Finally, the collected metrics are evaluated to identify throughput variations, resource utilization patterns, and potential performance limitations. This framework supports systematic performance evaluation of Kubernetes clusters under dynamic workload conditions.

Algorithm 1: Static CPU Allocation (SCA)

Input: Workload requests, Kubernetes cluster, CPU and memory limits
Output: Throughput, latency, and resource utilization metrics

1. Initialize Kubernetes cluster with multiple worker nodes.
2. Configure application pods and resource allocation limits.
3. Initialize workload distributor and monitoring components.
4. Deploy application pods across available worker nodes.
5. Generate workload requests with varying intensities.
6. Monitor workload changes and incoming request rates.
7. If workload variation is detected:
   Adjust the number of replicas using HPA.
   Else:
   Maintain the existing replica configuration.
8. Distribute incoming requests among available application pods.
9. Execute workload requests and record completed requests.

10. Collect throughput, response latency, CPU, and memory metrics.
11. Analyze performance metrics for each workload condition.
12. Report throughput variations and cluster performance.

End

The proposed algorithm evaluates application throughput in a Kubernetes cluster under dynamic workload conditions. Initially, the Kubernetes cluster is configured with multiple worker nodes, each having defined CPU and memory resource limits. Application pods are deployed across these nodes to distribute processing activities. The workload distributor is then initialized to generate application requests with varying intensities. The algorithm continuously monitors incoming workload requests to identify changes in request volume. When a workload variation is detected, the Horizontal Pod Autoscaler (HPA) adjusts the number of application replicas according to the configured scaling parameters. This enables the cluster to respond to increasing or decreasing workload demands. If no significant workload variation is detected, the existing replica configuration is maintained.

Following the scaling decision, incoming requests are distributed among the available application pods for execution. The algorithm records successfully completed requests and collects relevant performance metrics. These include throughput, response latency, CPU utilization, and memory utilization. Throughput represents the number of successfully processed requests within a specified time interval, while latency measures the time required to process individual requests.

Finally, the collected metrics are analysed across different workload conditions to identify performance variations and resource utilization patterns. The resulting performance report helps evaluate how effectively the Kubernetes cluster handles changing workload intensities. This analysis provides insights into throughput behaviour, resource constraints, and potential performance bottlenecks, supporting a systematic assessment of Kubernetes cluster performance under dynamic workload conditions.

Table I. Existing CPU Utilization

Number of Nodes	**Existing CPU Utilization (%)**
3	92
5	88
7	84
9	81
11	78

Table I The table presents the CPU utilization of Kubernetes clusters with different numbers of worker nodes under the existing resource allocation approach. The evaluation considers five cluster configurations consisting of 3, 5, 7, 9, and 11 nodes. The observed CPU utilization values are 92%, 88%, 84%, 81%, and 78%, respectively. The results indicate a gradual reduction in average CPU utilization as the number of worker nodes increases. The 3-node configuration records the highest CPU utilization of 92%, indicating greater resource utilization within the smaller cluster. In comparison, the 11-node configuration records 78%. These observations establish a baseline for comparing CPU utilization against the proposed adaptive

resource allocation approach.

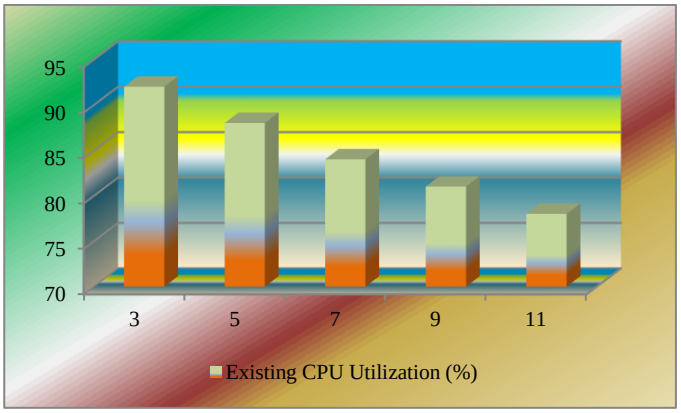


Fig. 2. Existing CPU Utilization

Fig. 2. The graph illustrates the relationship between the number of Kubernetes worker nodes and CPU utilization under the existing resource allocation approach. The horizontal axis represents the number of nodes, while the vertical axis represents CPU utilization in percentage. The graph shows a decreasing trend in CPU utilization as the cluster size increases from 3 to 11 nodes. CPU utilization decreases from 92% at 3 nodes to 78% at 11 nodes. This trend provides a baseline for examining resource utilization behaviour and assessing the effectiveness of adaptive CPU allocation under different cluster configurations.

PROPOSAL METHOD

Problem Statement

Kubernetes is widely used for managing containerized applications across distributed computing environments. However, maintaining consistent application performance under dynamic workload conditions remains a significant challenge. Traditional resource allocation approaches often rely on predefined CPU and memory limits and fixed application replica configurations. Such static configurations may not efficiently accommodate fluctuations in incoming request volumes, particularly during sudden workload increases or decreases. Insufficient resource allocation can lead to increased response latency, CPU contention, and reduced throughput, whereas excessive allocation may result in inefficient resource utilization. Although Kubernetes provides automated scaling capabilities, selecting appropriate resource configurations remains challenging when workload characteristics change continuously. Furthermore, variations in pod distribution, resource availability, and processing demands can influence the overall request-processing capacity of the cluster. Therefore, a systematic approach is required to evaluate how workload variations and resource allocation decisions affect application throughput, response latency, and resource utilization in Kubernetes clusters.

Proposal

This research proposes an Adaptive Resource Allocation Architecture for evaluating Kubernetes cluster performance under dynamic workload conditions. The proposed approach generates application requests with varying intensities, including low, high, and burst workloads. A monitoring mechanism continuously

observes incoming request rates, CPU utilization, and memory consumption across worker nodes. Based on predefined workload thresholds, Horizontal Pod Autoscaling (HPA) adjusts the number of application replicas to accommodate changing processing demands. Appropriate resource-management policies are additionally considered for adjusting CPU and memory allocation. When workload conditions remain stable, the existing configuration is maintained. Incoming requests are distributed across available application pods and executed within the cluster. Performance measurements, including throughput, response latency, CPU utilization, and memory consumption, are collected and analysed under different workload conditions. The proposed evaluation identifies performance variations, resource constraints, and scalability limitations to assess adaptive resource allocation effectiveness.

IMPLEMENTATION

Fig. 3. The proposed architecture can be implemented using a Kubernetes cluster configured with multiple worker nodes. A containerized sample application is deployed across the cluster using Kubernetes Deployments and Services. Workload generation tools such as Apache JMeter or Locust are used to simulate different request intensities, including low, high, and burst traffic. Kubernetes Horizontal Pod Autoscaler (HPA) is configured to dynamically adjust application replicas based on CPU utilization thresholds. Resource requests and limits are defined for each application pod, while Kubernetes resource-management configurations support CPU and memory allocation. Prometheus is integrated to collect system metrics, and Grafana is used to visualize resource utilization and application performance. During experimentation, different workload levels and replica configurations are evaluated. The number of successfully processed requests, response latency, CPU utilization, and memory consumption are recorded. The collected measurements are analysed to compare application performance under static and adaptive resource allocation conditions and evaluate throughput variations.

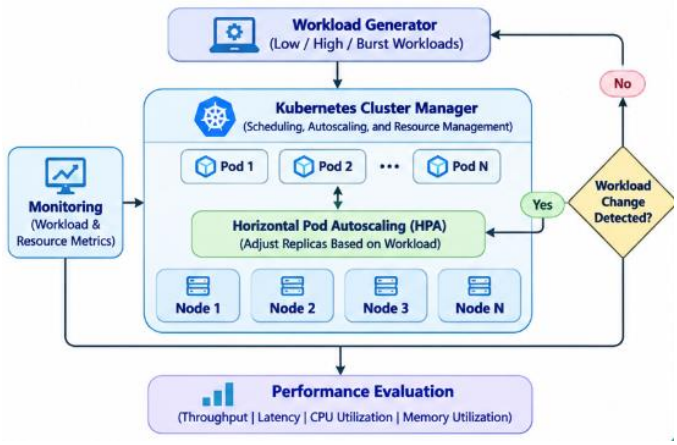


Fig. 3. Adaptive Resource Allocation Architecture

The proposed Adaptive Resource Allocation Architecture is designed to evaluate and manage application performance in Kubernetes clusters under varying workload conditions. The existing architecture follows a static resource allocation approach, in which application pods operate with predefined CPU limits and a fixed number of replicas. Although this configuration supports predictable workloads, it may not efficiently accommodate sudden increases or decreases in application requests. In the proposed architecture, a workload generator produces application requests with different intensities, including low,

high, and burst workloads. The Kubernetes Cluster Manager coordinates application deployment, pod scheduling, and resource management across multiple worker nodes. A monitoring component continuously observes workload variations and resource utilization, including CPU and memory consumption.

When a significant workload variation is detected, the Horizontal Pod Autoscaler (HPA) adjusts the number of application replicas based on configured scaling policies. CPU allocation can also be adjusted through appropriate resource-management configurations. When workload conditions remain stable, the existing replica configuration is maintained. This approach enables the cluster to respond to changing application demands.

Application requests are subsequently distributed across the available pods and processed by the worker nodes. During execution, performance metrics such as throughput, response latency, CPU utilization, and memory utilization are collected. These measurements are analysed to understand the relationship between resource allocation and application performance.

The proposed architecture provides a structured mechanism for studying adaptive resource allocation and its influence on throughput, scalability, and resource utilization under dynamic workload conditions.

Algorithm: Adaptive Resource Allocation

1. Initialize the Kubernetes cluster with multiple worker nodes.
2. Deploy application pods with predefined CPU and memory limits.
3. Initialize workload generation and monitoring components.
4. Generate application requests with varying workload intensities.
5. Monitor incoming request rates and resource utilization.
6. Identify workload variations using predefined thresholds.
7. If workload variation is detected:
   a. Adjust application replicas using HPA.
   b. Adjust CPU and memory resources using resource-management policies.
   Else:
   Maintain the existing resource configuration.
8. Distribute incoming requests across available application pods.
9. Execute requests and record successfully completed transactions.
10. Collect throughput, response latency, CPU, and memory metrics.
11. Analyze resource allocation and application performance.
12. Generate the performance evaluation report.

The proposed algorithm implements adaptive resource allocation to evaluate application performance under dynamic workload conditions in Kubernetes clusters. Initially, the Kubernetes environment is configured with multiple worker nodes, and application pods are deployed with predefined CPU and memory limits. Workload generation and monitoring components are initialized to simulate varying application demands. The algorithm continuously monitors incoming request rates and resource utilization to identify workload variations. When workload intensity exceeds or falls below predefined thresholds, the resource-management mechanism initiates appropriate adjustments. Horizontal Pod Autoscaling (HPA) dynamically increases or decreases the number of application replicas based on configured scaling

metrics. CPU and memory allocation can be adjusted independently through suitable resource-management policies. When workload conditions remain stable, the existing configuration is maintained. After resource adjustment, incoming requests are distributed across available application pods for execution. The algorithm records successfully completed requests and collects performance metrics, including throughput, response latency, CPU utilization, and memory consumption. Finally, the collected metrics are analysed to evaluate the relationship between resource allocation and application performance. The results help identify resource utilization patterns, throughput variations, and potential scalability limitations. This provides a basis for assessing the effectiveness of adaptive resource allocation in Kubernetes clusters.

Table II. Adaptive CPU Utilization

Number of Nodes	Adaptive CPU Utilization (%)
3	78
5	73
7	69
9	66
11	63

Table II The table presents the CPU utilization of Kubernetes clusters under the proposed adaptive resource allocation approach. Five cluster configurations consisting of 3, 5, 7, 9, and 11 worker nodes are considered. The corresponding CPU utilization values are 78%, 73%, 69%, 66%, and 63%, respectively. The results indicate a gradual reduction in CPU utilization as the number of worker nodes increases. The 3-node configuration records the highest utilization of 78%, whereas the 11-node configuration records the lowest utilization of 63%. This reduction indicates improved CPU resource distribution across the evaluated cluster configurations. These measurements provide a basis for comparing adaptive resource allocation with the existing resource allocation approach.

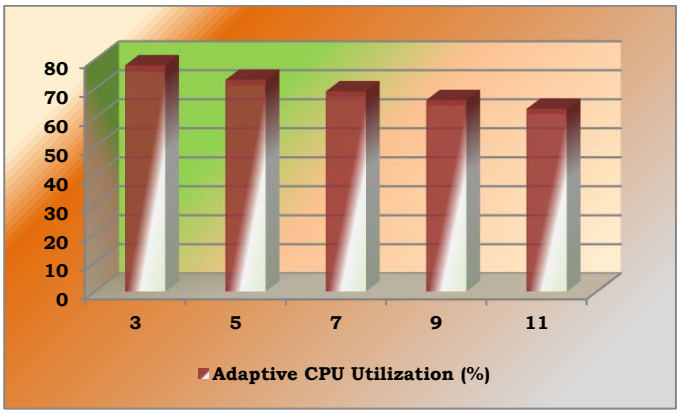


Fig. 4. Adaptive CPU Utilization

Fig 4 The graph illustrates CPU utilization across Kubernetes clusters containing 3, 5, 7, 9, and 11 worker nodes under the proposed adaptive resource allocation approach. The horizontal axis represents the

number of worker nodes, while the vertical axis indicates CPU utilization percentage. CPU utilization decreases from 78% for 3 nodes to 63% for 11 nodes. This downward trend demonstrates the variation in CPU utilization across different cluster sizes. The measurements provide insights into resource utilization behaviour under adaptive allocation and establish a basis for evaluating its performance against the existing resource allocation approach.

Table III. Existing vs Adaptive CPU Utilization

Number of Nodes	Existing CPU Utilization (%)	Adaptive CPU Utilization (%)
3	92	78
5	88	73
7	84	69
9	81	66
11	78	63

Table III Presents a comparative analysis of CPU utilization under existing and proposed adaptive resource allocation approaches across Kubernetes clusters containing 3, 5, 7, 9, and 11 worker nodes. Under the existing approach, CPU utilization decreases from 92% to 78% as the number of nodes increases. In comparison, the adaptive approach records lower utilization values of 78%, 73%, 69%, 66%, and 63%, respectively. The results indicate that adaptive resource allocation consistently maintains lower CPU utilization across all evaluated cluster configurations. The maximum difference of 15 percentage points is observed in the 11-node configuration. These findings demonstrate the potential of adaptive resource management to improve CPU utilization efficiency compared with static resource allocation.

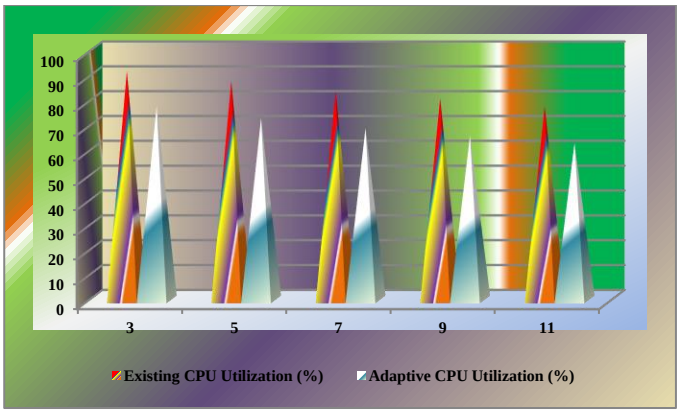


Fig. 5. Existing vs Adaptive CPU Utilization

Fig. 5 The graph compares CPU utilization under existing and adaptive resource allocation across different Kubernetes cluster sizes. The horizontal axis represents the number of worker nodes, while the vertical axis indicates CPU utilization percentage. Both approaches exhibit a decreasing utilization trend as the number of nodes increases. However, the adaptive approach consistently records lower CPU utilization than the existing approach. At 3 nodes, utilization decreases from 92% to 78%, while at 11 nodes, it decreases from 78% to 63%. This comparison highlights the potential benefits of adaptive resource allocation in managing CPU consumption across different cluster configurations.

EVALUATION

The experimental evaluation compares existing and adaptive CPU allocation across Kubernetes clusters containing 3, 5, 7, 9, and 11 worker nodes. The results indicate that adaptive resource allocation consistently achieves lower CPU utilization than the existing approach. For the 3-node configuration, CPU utilization decreases from 92% to 78%, while the 11-node configuration shows a reduction from 78% to 63%. The maximum reduction of 15 percentage points is observed with 11 nodes. Overall, the results demonstrate improved CPU resource utilization efficiency through adaptive allocation across different cluster configurations. Further throughput and latency measurements can strengthen the performance evaluation.

CONCLUSION

This study presents an Adaptive Resource Allocation Architecture for Kubernetes clusters operating under dynamic workload conditions. The comparative results demonstrate that adaptive allocation reduces CPU utilization across all five evaluated cluster configurations. The observed reduction ranges from 14 to 15 percentage points, indicating improved resource management compared with the existing approach. The proposed architecture provides a foundation for managing changing application demands through workload monitoring, replica scaling, and resource allocation policies. Future work will focus on evaluating throughput, response latency, and scalability using practical Kubernetes experiments under diverse workload conditions.

Future Work: Future work will explore resource-aware scheduling across heterogeneous infrastructure, incorporating node capacity prediction and intelligent allocation strategies to improve scalability, reduce hardware dependency, and maintain consistent application performance.

REFERENCES:

1. Naveen Kumar Bandaru, “Adaptive Node Utilization Optimization in Distributed Systems”, Journal of Advances in Developmental Research, IJAIDR, Volume 16, Issue 2, July-December 2025, E-ISSN: 0976-4844.
2. Vipul Kumar Bondugula, MANAGING CONFLICT RATE REDUCTION IN SCALABLE DISTRIBUTED DATABASE SYSTEMS, International Journal of Innovative Research and Creative Technology, Volume 9 Issue 1 January-2023, E-ISSN: 2454-5988, https://doi.org/10.5281/zenodo.15356925
3. Naveen Kumar Bandaru, "Predictable Lock Coordination for Scalable Multi Node Transactional Systems", 6th International Conference on Advancing Knowledge from Multidisciplinary Perspectives in Education, Engineering & Technology , ICAKMPET , 24-25 January 2026 Cebu, Philippines. pp. 55-65
4. Arunkumar Sambandam, “Multimodal Observability for Input Output Bottleneck Detection “, Volume 2021 Issue 8, Computer fraud and Security, Page No: 1-17, E-ISSN: 1873-7056, August 2021
5. Renukadevi Chuppala , Dr. B. Purnachandra Rao, Hadoop Distributed File System Write Operations , International Journal of Intelligent Systems and Applications in Engineering IJISAE, 2024, 12(23s), 720–731
6. Kishore Kumar Jinka, Dr.B.Purnachandra Rao , Multi Variate Linear Regression Analysis For

Sonatype Nexus Repository Space Management , International Journal of Intelligent Systems and Applications in Engineering (IJISAE), ISSN: 2147-6799, Volume 12, Issue 23S, August 2024

7. Naveen Kumar Bandaru, "Transaction Batching for Low Latency Commit Processing in Distributed Systems", International Journal on Science and Technology, IJSAT, Volume 12, Issue 3, July-September 2021, E-ISSN: 2229-7677.
8. Vipul Kumar Bondugula , Designing Effective Lock-Based Concurrency Control in Database Systems, International Journal of Intelligent Systems and Applications in Engineering, 26.02.2021 , Vol 9, NO : 1, ISSN: 2147-6799
9. Naveen Kumar Bandaru, "Enhancing Cluster Performance by Reducing Response Time Variance Issues", International Conference on Computer, Cybernetics and Education, ICCCE 2026, 27-28 February 2026, Bali, Indonesia. pp. 63-73
10. Vipul Kumar Bondugula, Managing Transactions in Snapshot Isolation with Adaptive Timeouts , International Journal of Innovative Research in Engineering & Multidisciplinary Physical Sciences, Volume 9, Issue 4, E-ISSN:2349-7300,
11. Naveen Kumar Bandaru, "Latency Aware Data Partitioning Techniques for Distributed Systems", International Journal of Innovative Research and Creative Technology, IJIRCT, Volume 10 Issue 1, January 2024, ISSN: 2454-5988.
12. Vipul Kumar Bondugula, Enhancing Deadlock Management in Distributed Databases using Serializable Snapshot Isolation, International Journal of Leading Research Publication, 2022-05-04, Volume 3, Issue 5, E-ISSN: 2582-8010
13. Naveen Kumar Bandaru, "Lightweight Runtime Conflict Detection for CPU Efficient Transaction Processing", International Journal of Intelligent Systems and Applications in Engineering , IJISAE, 2023, 11(6s), June 2023, ISSN:2147-6799.
14. Vipul Kumar Bondugula, Streamlining Transaction Commit for Distributed Databases with Thomas's Write Rule, International Journal For Multidisciplinary Research, Volume 5, Issue 4, July-August 2023, E-ISSN: 2582-2160, https://doi.org/10.36948/ijfmr.2023.v05i04.44253
15. Arunkumar Sambandam, "Adaptive Replication for Low-Latency Distributed Clusters", Volume 10, Issue 2, International journal of Intelligent Systems and Applications in Engineering (IJISAE), Page No: 1-20 , ISSN: 2147-6799.
16. Naveen Kumar Bandaru, "Correlated Telemetry Driven Throughput Analysis in Distributed Pipelines", Computer Fraud And Security Journal, CFSJ, Vol: 2022, Iss: 5 , 2022, ISSN (online): 1873-7056.
17. Vipul Kumar Bondugula, Storage Optimization in Distributed Environments Using Optimistic Concurrency Control, International Journal on Science and Technology, Volume 15, Issue 2, April-June 2024 E-ISSN: 2229-7677
18. Naveen Kumar Bandaru, "Encoded Data Transfers for Reducing Raw Network Transfer Time", International Journal of Technology and Applied Science, IJTAS, Volume 15, Issue 10, October 2024, E-ISSN: 2230-9004.
19. Kalesha Khan Pattan, "Optimizing Fault-Tolerance in Distributed Systems with AI-Augmented Replica Management", International Journal of Intelligent Systems and Applications in Engineering (IJISAE), Vol. 9, No. 1, 2021, pp. 139–160, ISSN: 2147-6799.
20. SaiKrishna Mylavarapu, "Adaptive CPU Resource Management in Distributed Systems", International Journal of Intelligent Systems and Applications in Engineering (IJISAE), Vol. 10,

No. 2S, ISSN: 2147-6799.

21. Vijaya Krishna Namala, “Hybrid Telemetry Fusion for Early Detection of Systemwide Failures”, International Journal of Intelligent Systems and Applications in Engineering (IJISAE), Vol. 11, No. 2S, ISSN: 2147-6799.