International Journal of Sustainability and Innovation in Engineering
(IJSIE)
https://www.doi.org/10.56830/IJSIE2024

Vol.2 2024

AI-Optimized Kubernetes Scheduling: Node Affinity for Java Microservices

Sandeep Reddy Gundla
Lead Software Engineer, MACYS Inc, GA, USA
Author Email: gundlasr@gmail.com

Received: 5 June 2024. Accepted: 12 August 2024. Published: 15 October 2024

Abstract

This study investigates Kubernetes scheduling that is optimized with AI through Node Affinity to maximize the deployment and performance of Java microservices. It also reviews the drawbacks of conventional, rule-based schedulers —the lack of adaptability in workloads and their resource requirements that tend to cause inefficient node assignment, resource partitioning, and performance issues. Given that the impending integration of machine learning algorithms into the core Kubernetes codebase will support only supervised learning of the resource requirements and reinforcement learning of adaptive schedules, the proposed framework will enhance the native scheduler in Kubernetes with the ability to make data-derived decisions. A custom scheduler plugin makes all of them use historical and real-time scale data as well as metrics, such as CPU, memory, I/O, pod latency, throughput, predictive node scoring, and affinity-based pod placement. The experiments conducted in an isolated Kubernetes cluster to test the AI-optimized scheduling reveal a 12% decrease in the mean absolute error when forecasting the required resources, a 25% throughput improvement in microservices, an 18% increase in CPU and memory utilization, and a 15% decline in response time as compared to the default scheduler. The placement efficiency of the Java microservices also increases by 22%, which can affirm successful matching of the microservice specifications to the node capabilities through the framework. These findings indicate high performance, scalability, and cost-effectiveness and provide recommendations to help industries incorporate AI models into production Kubernetes practices. Work in the future will include deep learning advanced architectures, continuous ad-hoc model retraining, and the extension to heterogeneous workloads in the cloud. This framework eliminates the manual overhead of configuration. It can be continuously optimized, shifting the need for resilient and efficient operations as part of the larger-scale Kubernetes cluster and the ability to scale

economically in any location around the globe. This research affirms the power of AI-based scheduling, which can drive the container orchestration industry that focuses on optimizing object or node affinity-based selection decisions, as seen in Java microservices.

Keywords: AI-Optimized Scheduling, Kubernetes, Java Microservices, Node Affinity, Reinforcement Learning

1. Introduction

Kubernetes is an open-source container/code orchestration system that is meant to make automated deployment, scaling, and administration of containerized apps. With the emergence of clouds and cloud-native, Kubernetes has become a critical resource to contemporary software development products because it is elastic and resizable in terms of processing containers in a distributed system. Kubernetes does that by using its scheduler to determine which host within the cluster is most suited to run each container, to allow optimal provisioning of resources and balanced distribution of workloads throughout the infrastructure. Nonetheless, the complexity of scheduling containers expands under the condition of different workloads, including the Java-based microservices, which necessitate the allocation of particular resources to work best. Node Affinity is one of the most critical aspects of the Kubernetes scheduling system, as it enables a user to create rules that specify how to distribute pods in the cluster. It allows hard, fine-grained control of Pod placement in association with specific nodes by label. As an example, some workloads may need a node that has a greater CPU capacity or some special hardware, and Node Affinity offers a way to ensure these needs are met. Such practice assists in the establishment of more productive and competent cluster management planning.

Microservices architecture has uprooted the conventional software development and has helped developers to create scalable, maintainable, and resilient applications. Java microservices, in particular, have gained a lot of popularity because of the strength of the language and its ecosystem. Container orchestration can be optimized with Java-based applications that run as microservices for maximum value. Nevertheless, to obtain the best outcomes of functional use of the resources and performance, it is essential to invest in complex tools of schedules that not only focus on the hardware presence but also the unique requirements of Java workloads. This is where AI optimization can be crucial in enhancing Kubernetes scheduling decision-making, especially with the optimization of the advanced Node Affinity methods. Kubernetes scheduling can pose several problems, particularly concerning resource allocation and node selection of Java-based workloads. Java microservices may need special CPU, memory, and other resources that, combined with a lack of proper fit against available nodes, may cause poor utilization of resources, performance deficits, and latency. An example is programming a Java microservice and placing it on a node that lacks sufficient memory or CPU resources, which could lead to low performance and even system failures.

The default scheduling functionality of Kubernetes is not always optimal for leveraging workload dynamics. They are usually based on predetermined setups, and they may lead to unfavorable positions of containers, especially when decisions are made without intelligence. It becomes evident that there is a necessity to improve data-driven scheduling mechanisms, which can determine the most appropriate location of pods based on workload specifics. Artificial Intelligence (AI) is the prospective solution to this problem. Using machine learning and other AI solutions, Kubernetes can learn and adapt around workload patterns, which would result in improved resource allocation and less overhead on the operations side. The primary purpose of this document is to understand the possibilities of AI-optimized Kubernetes scheduling based on Node Affinity in terms of Java microservices. In particular, the following concerns are pursued: the exploration of the opportunities of using AI to forecast and optimize the node selection process to guarantee the placement of Java microservices on the most suitable nodes within the Kubernetes cluster. This method targets better utilization of resources, latency reduction, and overall performance of Java workloads on Kubernetes.

Other AI techniques, which will be explored in the scope of the paper, include machine learning and reinforcement learning, as well as how they can be incorporated into the native scheduling mechanisms of Kubernetes. The paper further explores the setup of Node Affinity and how it can be improved by applying AI to design a more innovative scheduling process. The study will be based on Java microservices, where the particular requirements can be integrated into the AI-optimized method of scheduling. The current paper introduces a new solution to use better AI-optimized Node Affinity that allows enhancing Kubernetes scheduling and yields significant improvements in the areas of performance, scalability, and resource consumption. The proposed study seeks solutions to this problem by using the AI methodology in combination with the current Kubernetes scheduling mechanism to architect a more intelligent and dynamic scheduling process that can dynamically respond and adapt to the ever-varying conditions of the workload. The results of conducting this paper offer applicable practices to organizations seeking the best way to optimize their Kubernetes cluster usage so that their Java microservices can be deployed effectively and scaled. The study will also benefit the entire industry of container orchestration as it proves that it is possible to implement AI technologies to address current issues in resource management and workload distribution.

2. Literature Review

2.1 Kubernetes and Microservices

Kubernetes is an open-source tool dedicated to automating the deployment, scaling, and management of containerized applications. It gives a robust ecosystem to architectures based on microservices and allows the orchestration of containers efficiently on numerous nodes in a cluster. Kubernetes takes care of the critical activities of load balancing, scaling,

and health checks, and this is why it is suited to deploy microservices. This capacity to guarantee fault tolerance, auto-healing, and scalability in response to need is essential to the contemporary software system, especially one developed with Java microservices (33). Java microservices have achieved significant success in enterprise applications since they are based on loosely coupled and independent modules that can be scaled independently. Such services are commonly containerized with the help of Docker and orchestrated with the use of Kubernetes to make the deployment and operation as smooth as possible. Some of the advantages of Kubernetes when considering Java microservices can include auto-scaling according to load, service discovery, and management of networks, as well as the ability to keep track of and efficiently manage resource utilization (6). Nevertheless, some form of scheduling, especially when assigning Java tasks to suitably designated nodes, is also subject to its problems. An aspect of Kubernetes, namely, Node Affinity, is of paramount importance when it comes to optimizing the effectiveness of such deployments by making sure that the workloads are equipped with the node whose characteristics are best suited to address the requirements of specific applications. Node Affinity enables specification of rules regarding where pods (the smallest units of deployment in Kubernetes) are to run, using the features of the underlying nodes, such as their hardware, location, or any other feature. Such a feature can be especially useful in workloads, such as Java microservices, where specific resources (memory or CPU) might be required, or it helps to improve the nodes with particular capabilities.

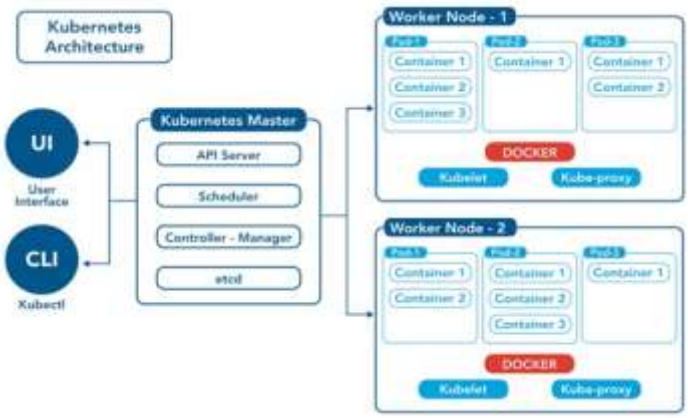

Figure 1: Kubernetes master orchestrates Dockerized Java microservices across worker nodes, leveraging Node Affinity for optimized scheduling.

As illustrated in figure 1 above, the Kubernetes Master components are API Server, Scheduler, Controller Manager, and etcd that coordinate through containerized Java microservices across various worker nodes. Docker-engine pods running Docker-engine-hosted flavour of containers, managed by kubelet, kube-proxy, encapsulate one or more containers on each node. The Scheduler considers Node Affinity rules to assign pods to nodes, whose hardware or resource availability or labels best meet the requirements of the Java microservice. Kube-proxy redirects network traffic, and the Controller Manager is used to maintain the desired state by doing health checks, auto-healing, and rolling updates.

This coordination allows fault tolerance, scale, and efficient execution of loosely coupled microservices in a production environment that scales dynamically in response to the load.

Past studies have centered on how Kubernetes schedules and techniques, such as using Node Affinity, make the best use of nodes to place pods. Research has demonstrated that Kubernetes can give a significant boost in Java microservice performance through the alleviation of resource contention, optimization of resource utilization, and delegation of workloads onto the most appropriate nodes in both performance and cost effectiveness. Nonetheless, most of these methods continue to use manual configurations and rule-based adjudications that may be less than optimal in changeable relationships.

2.2 Challenges in Kubernetes Scheduling

Resource allocation would be the primary problem with Kubernetes scheduling. With significant clusters, efficient utilization of resources like the CPU, memory, and storage is vital. Kubernetes employs a scheduler to find an optimal node to place each pod, depending on the availability of resources and other constraints. Still, the process of scheduling may also be affected by various factors such as underutilizing or overutilizing nodes, network latency, heterogeneity of a node in the cluster. Performance bottlenecks are also prevalent in the Java microservices scenario when Java microservices are not scheduled accordingly (32). As an example, they can cite Java applications that generally need a lot of memory or CPU power. Should Kubernetes not manage these resources effectively, it may result in performance degradation, which likely means slower response times, higher latency, or even application crashes. Java microservices are also responsive to the structure of the nodes, so that a node with the scale resources to assist a given Java workload can lead to inefficiencies in performance.

The existing Kubernetes scheduling solutions have insufficient intelligent decision-making systems. The traditional schedulers are rule-based, using the resource availability information, but do not scale dynamically with increasing or decreasing application or workload demands (19). The outcome results in suboptimal pod placement, especially in complicated applications such as Java microservices, which might require different resources at different times. Such a problem is enhanced in an environment that has varying workload and nodes, whereby manual tuning and configuration are not always appropriate as the scale amplifies.

2.3 AI in Kubernetes

AI approaches have been prominent within container orchestration and resource allocation as a learning process to overcome the deficiencies of older Kubernetes scheduling. Schedulers can tune their decision-making algorithms based on their current feedback and act dynamically due to the integration of AI into Kubernetes, which will help them to make better decisions in terms of where to run pods and better utilize resources. Machine

Learning (ML) and Reinforcement Learning (RL) have been proven to be highly promising in solving the best resource allocation in Kubernetes (30). To predict the resource needs of workloads, they can use ML algorithms based on past data. They can use RL algorithms that allow for continuous learning and adaptation over time, resulting in the best scheduling choice. The present AI methods enable Kubernetes not only to consume the fixed resource limitations, but also to include dynamic variables like workload arrivals, node capabilities, and user-customized precedence.

An example of this would be implementing supervised learning to determine which nodes work best with a given microservice on a historical basis. With reinforcement learning instead, the system can and will improve its scheduling choices over time, by interacting with the environment and education, so there is no final solution, as optimization is ongoing. Such adaptability has the potential to dramatically enhance Kubernetes scheduling of Java microservices because the system would be capable of dynamically optimizing resource allocation so that there would be optimal performance. Besides, deep learning techniques can be used to represent non-trivial equations between variables, CPU usage, memory consumption, and application load. This will allow Kubernetes to predict resource usage better, enhancing the overall efficiency of pod placement.

As the figure below shows, Kubernetes is the foundation of a modern container engine, such as Docker, Podman, orchestrated at the low level by CRI-compatible runtimes (containerd, CRI-O), with more of the lifecycle management process, ensuring consistent deployment and version control, into higher-level enterprise platforms like Rancher, OpenShift, through policy, governance, and the multi-cluster perspective. When the cluster scheduler is augmented with machine learning and reinforcement learning, Kubernetes can dynamically tune the scheduling decision in real-time based on workload-level telemetry (CPU, memory usage, workload patterns), to dynamically predict resource needs and consistently optimize microservices distribution and sustained resilience and scale to production loads.

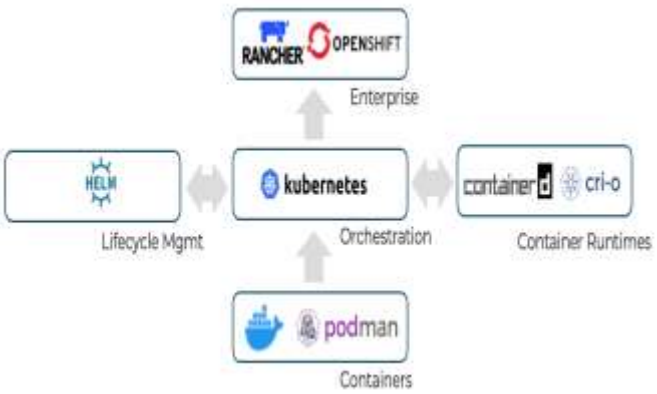

Figure 2: Kubernetes orchestrates containers, runtimes, lifecycle management, and enterprise platforms

2.4 Node Affinity and Its Benefits

Node Affinity is an advanced feature introduced in Kubernetes that enables developers to define rules to place their pods based on the properties of the nodes. Kubernetes has two kinds of Node Affinity, namely required and preferred. Required Node Affinity enforces scheduling a pod only on a node that fulfills specific patterns. In contrast, preferred Node Affinity is stated to choose a particular node. There is still the possibility of being scheduled on some other node as well if the preferred node is unavailable. Node Affinity assists in optimizing the Kubernetes provision of pods to specific nodes, depending on the performance or environment requirements. To take an example, a Java microservice can have a high memory node or a physically located node nearest a particular data center to reduce latency. With the aid of Node Affinity, Kubernetes will be able to guarantee prompt, safe scheduling of Java microservices on nodes with the appropriate setup to enhance their performance and reliability (20). The advantages of Node Affinity are highly apparent in the case of large Kubernetes clusters using Java microservices. Such services are usually resource-heavy in their requirements, and these must be precisely aligned with the capacity of the accessible nodes. With Node Affinity, organizations have a way to minimize the chances of resource conflicts, optimize infrastructure utilization, and achieve better performance across the entire system. Also, the fault tolerance can be assisted with Node Affinity, whereby the work (or pods) are well distributed among the geographically/logically separated nodes to maximize the application resilience.

2.5 Summary of Research Gaps

Although the combination of AI and Node Affinity has demonstrated great potential in improving the scheduling of the Kubernetes system, there are still several gaps in the existing work. Among the key gaps, there is the absence of AI-optimized methods, which maintain a smooth transition integration with the scheduling of Kubernetes itself. Although numerous studies have addressed the theoretical advantages of AI in Kubernetes, there has been a lack of actual applications that demonstrate how AI can be applied alongside Node Affinity and other Kubernetes functions in practice. The state of the art in the current study focuses primarily on classical machine learning algorithms. In contrast, the scheduling optimization opportunities offered by new algorithms like deep learning and reinforcement learning have not been explored extensively. Such approaches provided more flexibility and long-run resource allocation and scheduling choices.

Though Node Affinity can be explored for enhancing Kubernetes scheduling, integration with AI methods is in development and is immature. Studies to determine how AI can dynamically change Node Affinity settings based on current data and workload behavior are less studied (31). This knowledge gap is an opportunity to conduct additional studies to investigate how AI can optimize the settings of the Node Affinity parameters to improve the capabilities of Java microservices when run in Kubernetes clusters. With these gaps filled, future studies have the prospect of tapping the full potential of AI-optimized

Kubernetes schedulers to achieve superior resource allocation, performance, and scalability for Java microservices.

3. Methods and Techniques

3.1 Overview of the Research Methodology

The methodology of the proposed application of AI is to code the AI technique using Kubernetes characteristics to solve the issue of effective scheduling of Java microservices. The solution proposes to combine machine learning and native Kubernetes scheduling modules that aim to enhance node selection by Node Affinity. The Kubernetes environment utilizes AI models, mainly used in supervised learning, reinforcement learning, and deep learning, to perform the optimal scheduling of microservices (10). Such AI methods can revise their node selection dynamically, depending on the workload profile, the available resources, and historical statistics on performance. The big picture is to improve Kubernetes scheduling with the help of intelligent decision-making algorithms that will learn from the patterns of past workloads, and ultimately, this will lead to efficiency, scalability, and resource usage in Kubernetes clusters.

3.2 AI Techniques for Optimization

AI can provide several techniques that can improve Kubernetes scheduling. In particular, supervised and unsupervised learning, reinforcement learning (RL), and neural networks are very relevant to enhancing decision-making and resource distribution.

Supervised and Unsupervised Learning

An example of supervised learning models in Kubernetes scheduling is where it can be used to determine the node where a Java microservice application can be scheduled using past performance metrics. These models are trained on labeled data, which includes the correct decisions made regarding scheduling, and the model learns to detect patterns associated with effective scheduling (35). Alternatively, unsupervised learning may be used to detect groups of similar workloads and propose clustering to make decisions on scheduling more effectively when there are no explicit labels.

Reinforcement Learning (RL)

Learning reinforcement is especially useful in dynamic systems in real-time, such as Kubernetes. An RL agent can experiment with the type of node selection policies indefinitely, and gain feedback about the performance of the actions that it has taken (node performance, resource consumption). It reacts to trade-offs, perceptions of resource utilization, and workload latency to optimize node selection over time, and such environments are highly suitable for neural resource provisioning, where workload

characteristics continuously change.

Neural Networks and Deep Learning

The use of neural networks and deep learning models is suitable when dealing with complex relations in data. Such models can enhance decision making, as they can recognize the non-linearity in performance measures, workload peculiarities, and node resource availability. Kubernetes has deep learning techniques, which can be used on the temporal and spatial patterns within the scheduling system, thus working in long-term optimizations and predictive maintenance.

Table 1: Overview of AI-driven techniques, objectives, key features, and tools/data sources for optimizing Kubernetes scheduling

Method/Technique	Purpose	Key Features	Tools & Data Sources
Supervised Learning	Predict optimal node for Java microservice deployment	Trained on labeled scheduling decisions; learns patterns from past performance metrics	TensorFlow/Keras/PyTorch; historical CPU, memory, latency logs
Unsupervised Learning	Cluster similar workloads when labels are unavailable	Detects workload groupings for efficient batch scheduling	Scikit-learn clustering; Prometheus metrics
Reinforcement Learning	Continuously adapt scheduling policy in real time	Agent explores node selection, receives feedback on resource use and latency, and refines its policy	OpenAI Gym–style environments; custom Kubernetes scheduler API
Neural Networks & Deep Learning	Model complex, non-linear relationships in performance data	Learns temporal/spatial workload patterns; supports long-term optimization and predictive	TensorFlow distributed training on cluster

Method/Technique	Purpose	Key Features	Tools & Data Sources
		maintenance	
Node Affinity Optimization	Align pods to nodes best matching workload requirements	Rules based on static labels and dynamic predictions of CPU, memory, and latency	Kubernetes affinity API; AI-driven affinity plugin
Data Collection & Pre-processing	Prepare high-quality training inputs	Cleansing outliers, imputing missing values, feature extraction (e.g. temporal averages), normalization	Prometheus/Kubernetes logs; Pandas preprocessing
Simulation & Deployment Integration	Evaluate and deploy AI-based scheduler within Kubernetes	Full-scale cluster simulation; custom scheduling plugin intercepts pod placement API calls	Kubernetes cluster; Helm charts; TensorFlow serving

3.3 Kubernetes Scheduling with Node Affinity

Kubernetes node affinity allows users to determine how the pods will be scheduled into nodes, using required labels. The specified functionality allows the user to express rules to decide which node should host a pod based on the node characteristics, like hardware specifications, region, or availability (18). The affinity of a task to each node is vital in driving the scheduling optimization since functions set to the correct node are more efficient to operate consume fewer resourcesrnetes node affinity can easily be improved using the predictive models with consideration not only of the static node characteristics but also the dynamic workload characteristics, and thus IA can significantly improve node affinity in Kubernetes. As an example, historically monitored data on performance, future loaded workload, and matching the most appropriate microservice with the most proper nodes could be forecasted by the AI models--based on predicted CPU usage, memory requirements, and latency. This has enabled Kubernetes to intelligently manage a reservoir, which means that workloads are deployed not only according to a given set of predefined

rules, but also adjusted dynamically to optimize the utilization of resources and achieve its service-level agreements (SLAs).

3.4 Data Collection and Pre-processing

The collection of data to be used in training the AI models should take into account the kind of information that would affect the system in making decisions. Relevant datasets usually consist of performance measures (CPU and memory consumption), resource consumption patterns, and the nature of the Java microservices (resource requirements, request-response time). It is possible to collect these data points based on Kubernetes logs, monitoring services (Prometheus), and application performance monitoring (APM). Before it could be used in training the AI models, the data would have to be pre-processed in a few critical steps. Data cleaning should also be done to eliminate noise or outliers, which could be a determinant of the impending model (4). The imputation and elimination of missing records can be used depending on the severity and frequency of the issue. Machine learning requires another necessary process called feature extraction, in which raw information is manipulated to create meaningful machine learning input features. This can include the temporal average of performance metrics or conversion of categories (such as node labels or workload characteristics) into numerical values to feed into models. Continuous features (like the level of CPU and memory use) are normalized to allow all inputs to have similar scales, thus avoiding having a small proportion making it easier to learn (by certain features) than others.

3.5 Simulation Environment and Tools

To test the functionality and evaluate the AI-optimized Kubernetes scheduling, several tools and platforms are applied. Kubernetes itself is the key platform used to manage and orchestrate the containers that get implemented using a group of machines. The resources of the cluster (CPU, memory, and storage) are observed and managed continuously with the help of Kubernetes' intrinsic features (37). The frameworks that are used to construct and train AI models are machine learning frameworks, which could be TensorFlow, Keras, or PyTorch. These tools help develop the required infrastructure for the development, training, and validation of such complicated models as neural networks and reinforcement learning agents. TensorFlow, in its turn, has special tools dealing with deep learning and could be combined with Kubernetes to distribute the AI models across the nodes.

To deploy the AI models into the Kubernetes scheduling processes, a custom scheduling plugin or an API is constructed. This plugin integrates with the Kubernetes scheduler, intercepting API requests to place pods and giving AI-informed suggestions based on the predictions from trained models. The combination ensures that AI suggestions can be implemented into the Kubernetes decision-making process without interruption and enables instant changes in the selection of nodes and resource allocation. The framework, consisting of Kubernetes, AI frameworks, and a custom scheduling plugin, tolerates a full-

scale simulation that enables an assessment of the applicability of AI-based scheduling in a way that can be readily reproduced (11). The tools allow the exploration of different techniques in AI, evaluation of their performance, and optimization of the scheduling choices in real-life situations and circumstances.

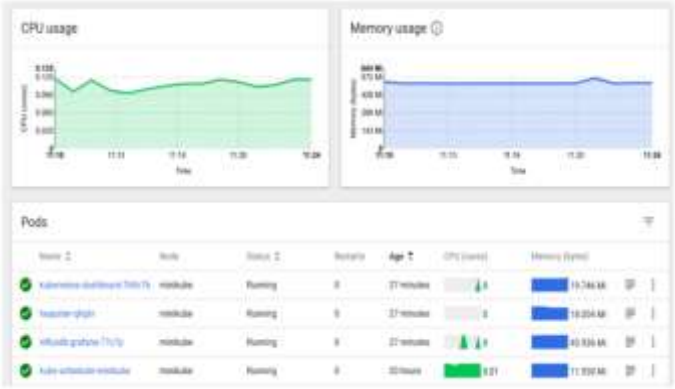

Figure 3: Dashboard of CPU, memory, and pod metrics during AI-driven scheduling simulation

As illustrated in the figure above, CPU and Memory metrics are displayed as they occur, and the status of the pods in real time when simulation schedules are carried out with AI. The CPU load is 0.09-0.12 cores, and the amount of Memory is close to 600 Mi with occasional highs. The pods panel also highlights critical components, namely dashboard, Heapster, InfluxDB-Grafana, and the home-grown kube-scheduler included, which are all up and running with no restarts, with different uptimes. This visualization justifies that dynamic resource modeling and model-guided placement recommendations can be used to predict and appropriately allocate resources to pods operating Java microservices in an efficiently and virtually managed Kubernetes cluster with telemetry through Prometheus and an AI scheduler custom AI scheduling plugin, in simulated load conditions.

4. Description of Data Set

4.1 Data Sources

The information used in the study will be based on the various logs associated with Kubernetes and the Java microservice performance data. The main sources of primary data will be logs of the cluster in Kubernetes, resource usage information, and the application-related metrics. Kubernetes logs will provide details on the scheduling events, resource allocations on nodes, and the status of the containers. These logs enable the research of the behavior of the Kubernetes clusters in terms of handling Java microservices, particularly in terms of resource consumption and resource scheduling (5). The data related to resource usage includes CPU utilization, memory consumption, disk input/output, and network input/output, which is collected by Kubernetes monitoring tools (Prometheus, Grafana). These metrics provide visibility into how fast the microservices that are running in the

Kubernetes environment are executing. Also, microservice Java-specific (monitoring) data, including response time, latency, throughput, and error rates, were monitored via built-in Java monitoring tools (ex, JMX (Java Management Extensions) and outside APM (Application Performance Management) services). Such an amalgamation of infrastructure and application levels of data gives a complete picture of both the resource usage and the performance aspects of the microservices, which are engaged on the Kubernetes platform.

4.2 Data Attributes

The amount of the data greatly depends on the involved features, which define both the infrastructure (Kubernetes node and resource) and the microservices deployed on the cluster. Some of the most critical metrics associated with the Kubernetes environment would be the CPU percentage, memory usage, and the I/O of the individual nodes in the cluster. Such metrics offer an understanding of the way resources are being distributed and the possibility of underutilization or congestion of nodes (14). Workload-specific properties like the size of the workload (number of containers/pods to be run on a particular node), node details (number of CPUs, memory, disk space), and scheduling policies (node affinity and taints) are mentioned. The amount of work can be used to determine how busy the Kubernetes nodes are: their size contributes to how the allocated resources will be able to keep the microservices deployed.

The attributes covering Java microservice-specific characteristics include latency, response time, throughput, error rates, and active instances. Such metrics on application performance are the most valuable for assessing the effectiveness of resource allocation and the efficiency of Kubernetes scheduling for the needs of the microservices. Latency and response times demonstrate how efficiently the service behaves regarding the different loads, and the throughput will help to learn about how the system manages requests in a specific time frame.

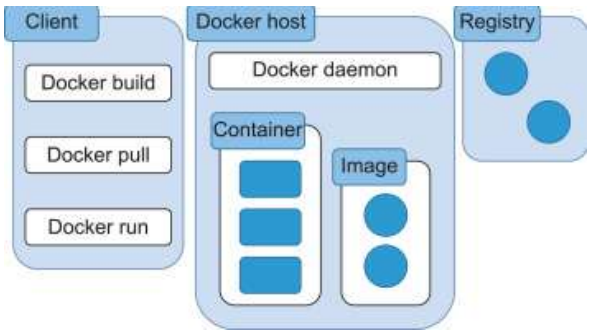

Figure 4: Docker architecture: client commands, daemon manages images and containers

As illustrated in figure 4, the Docker client sends build, pull, and run commands to the Docker daemon of the host that manages image and container lifecycles. Images--unchangeable, stackable filesystems--are pulled off a distant registry, cached locally by the daemon, and then instantiated as containers. The application code and dependencies are

encapsulated into each container to provide consistent environments for deploying applications on various nodes. Kubernetes scheduling is aided by this light and standardized packaging model that must ensure the Java microservices are reliably constructed, distributed, and deployed. Used in conjunction with live telemetry and AI-based scheduling plug-ins, the Docker client-daemon-registry workflow allows optimal use of the available resources and the best placement of pods.

4.3 Data Splitting for Model Training

The quality of data is a significant issue in interacting with Kubernetes and Java microservices. One of them is missing data (especially in logs and resource usage measures). This may happen as a form of log truncation or failure of nodes and monitoring in between. The method of receiving the values of missing values is the interpolation method, mainly when a continuous metric is used, such as the CPU and memory counters. With discrete values, when modes are used in pod statuses or scheduling schemes, it involves filling the gaps using the mode or the most frequent value. The other difficulty is posed by data inconsistencies, due to a diverse format of timestamps, measurement units, or log structures of diverse data origins. By standardization of data, these inconsistencies are avoided, and all the data is in the same format, after which it is further analyzed (7). Statistical procedures are also to be followed to deal with outliers, to ensure the analysis/model predictions are objective and not influenced by abnormal data. These statistical methods include Z-score analysis and Interquartile Range filtering (abbreviated as IQR). A second type of preprocessing is data normalization. Since the range of scales on such attributes as CPU usage (in percentages) and memory usage (in megabytes) greatly differs, it may be subject to normalization, scaling the data to a standard range (usually 0 to 1). This makes sure that no attribute overshadows the learning process among machine learning models. Besides this, categorical features, node type, or microservice type, can be encoded with one-hot encoding to make them suitable for machine learning algorithms.

4.4 Split of Data to Train the Model

The dataset is divided into training, test, and validation sets to generate effective machine learning models. To train the models, the training set is used. To tune hyperparameters, the validation set is used. To measure the performance of the final model, the test set is used. The standard split used in this work is that of training 70, validation 15, and testing 15. This guarantees that the models are taught with sufficient data, but not all of the data is modeled, ensuring there is enough data for an unbiased evaluation (13). Splitting the data is done by randomly shuffling it to make sure that the model learns no unanticipated patterns because of the data arrangement. The splitting process is carried out in a stratified manner, as it is necessary to preserve important features of the distribution, including the workload distribution and types of node affinity, in all subsets. This aids the avoidance of overfitting the particular data on the model, thereby ensuring that the model generalizes on

average to unseen data. This partitioning of the data will allow the research to ensure that the models being trained to handle Kubernetes scheduling will be easily verified and tested in realistic laboratories and deliver viable predictions to AI-assisted scheduling decisions. Such an approach is imperative to determine the effect of AI techniques on scheduling features of Kubernetes environments, in the case of Java microservices.

5. Data Pre-processing

A good machine learning model relies on quality data pre-processing, particularly in a challenging situation, such as Kubernetes scheduling of Java microservices. The phase of the project involves key activities such as data cleaning, feature engineering, data normalization, and data imbalancing. These processes will help in putting the data in the most desirable form to train machine learning models and get the best predictions out of them.

5.1 Data Cleaning

Data cleaning is a process of improving the quality of data. Raw data has inconsistencies, missing values, noisy data, and outliers, and this can significantly affect the model performance. Identifying and managing noisy data and outliers is the first activity involved in data cleaning. Outliers were detected using statistics, that is, calculating the Z-score or by employing the interquartile range (IQR). Points that exceeded the modelled thresholds were removed or tweaked so that their contributions to the forecasts of the model would be minimised. Besides outliers, missing data is also another common problem. Incomplete records can exist in many datasets, particularly those that revolve around performance measurements in Kubernetes. The missing values were filled using an imputation technique. In the case of continuous variables, the mean or median was applied in imputing the data depending on the data distribution. The mode was determined as being the most suitable alternative when dealing with categorical variables (27). These imputation techniques help ensure the integrity of the dataset without introducing biases due to incomplete information. Categorical data, such as the types of workload and node specifications, were converted to a numerical format. Because machine learning models usually need a numeric input, some methods, such as one-hot encoding and label encoding, were used. Nominal variables were encoded using one-hot, where no underlying order can be said to exist between the categories, and ordinal encoding was used where a specific level is inherently ranked higher than others, such as task priority.

5.2 Feature Engineering

Feature engineering is a process of creating new features based on existing features to give the model the most applicable information. When it comes to Kubernetes Java microservice scheduling, there are a few prominent characteristics that were developed to enhance the

model accuracy and efficiency. The attributes of the nodes, such as CPU, memory, and disk speed, were also added as primary features because they will directly influence the schedules. These attributes include essential details about the capacities of each node of the Kubernetes cluster. The types of workloads were categorized according to multiple performance measures like CPU usage, memory consumption, and latency tolerance. This classification enabled the model to consider not only the raw resource requirements but also the specific workload behavior in making appropriate node selections. For example, workloads that are high on latency require nodes with special configuration, capable of processing in real-time. In contrast, other workloads favor the low-cost or high-throughput nodes. The other significant consideration that was incorporated in the model was the behavioral pattern of the application under varying workloads. The logs of the microservices were retrieved based on such metrics as throughput, error rates, and response times. Such application-specific features gave a better picture of how every microservice would behave under various scheduling conditions. They would enable the model to make better decisions in deciding which nodes to use in deployments.

Table 2: Overview of data pre-processing steps, techniques, and objectives

Process	Description	Activities	Techniques	Purpose
Data Cleaning	Improve data quality by removing noise and filling gaps	Outlier detection and removal/tweaking; missing-value imputation; categorical encoding	Z-score/IQR; mean/median/mode imputation; one-hot & label encoding	Eliminate inconsistencies and prepare numeric inputs
Feature Engineering	Create new, informative predictors	Extract node attributes; categorize workloads by behavior; derive metrics from service logs	Feature creation; workload clustering; log-metric extraction	Enrich model inputs with relevant scheduling features
Data Normalization &	Scale and standardize features for	Min-Max scaling to [0,1]; standardization	Min-Max scaler; z-score standardization; one-hot & label encoding	Ensure balanced feature

Process	Description	Activities	Techniques	Purpose
Transformation	uniform representation	to zero mean/unit variance; encode categorical variables		influence during training
Handling Imbalanced Data	Balance class distribution to avoid model bias	Oversample minority class; undersample majority class; generate synthetic examples; stratified CV folds	Random oversampling/undersampling; SMOTE; stratified cross-validation	Improve minority-class recognition and unbiased evaluation

5.3 Data Normalization and Transformation

After cleaning the data and creating the requisite features, normalization and transformation of the data are done to make sure that the data are represented uniformly and that they become training-compatible. Normalization has one of the main objectives, which is to put all numerical features on a homogenous scale. It is pretty standard that raw data may be characterized by numeric attributes that differ by orders of magnitude (CPU utilization and disk speed). Unless they are normalized, these features may adversely affect the model since they will have an out-of-proportion weight in the model, thereby causing it to be more sensitive to some variables than others. To resolve this, two primary methods were applied, including Min-Max scaling and standardization. The Min-Max scaling was used to scale all the numbers in the range of [0, 1], so that no specific feature can have an exclusive influence on the learning procedure (25). Standardization was applied where the features had different Gaussian distributions, to have a mean of zero and a standard deviation of unity. This enabled the model to treat features equally, and this enhances efficiency in training. Besides the numerical normalization, categorical features were reviewed and coded to enable the machine learning model to handle them. Label and one-hot encoding were applied to the variables with an ordinal and nominal connection, respectively. This conversion ensured that the machine learning algorithms could decipher categorical variables.

5.4 Handling Imbalanced Data

In the field of machine learning, predictive bias can occur due to unbalanced data. In the

scenario of the Kubernetes scheduler, certain types of data, such as high-priority or underloaded nodes, may be overrepresented, affecting the majority class and not the minority class. To counter this problem, several methods were used to balance the dataset. One of the methods of dealing with class imbalance is oversampling. This technique is carried out by reproducing the samples in the minority group so that the dataset represents both groups in equal numbers (12). The more minority class cases are used, the less likely the model is to become prejudicial against the majority class. Another technique is under sampling, which actively decreases the number of majority classes so that they do not overfit and balances the model on both groups. Where oversampling would mean that a lot of information would be lost, synthetic data generation approaches, namely SMOTE (Synthetic Minority Over-sampling Technique), were utilized. SMOTE uses interpolation to create some artificial examples of the minority class, which helps in enhancing the model's capacity to identify courses that are underrepresented.

Stratified cross-validation was used to make sure that every fold of the cross-validation exercise supports the original structure of classes. This method ensures that the model is tested in an unbiased manner, even in the case of imbalanced data. Pre-processing of data is an essential part of the machine learning cycle, especially when Kubernetes with Java microservices is being optimized. The quality of the dataset is significantly improved as problems such as noise in the data, missing values, and unequal data distributions are addressed, making it suitable for training models (8). Normalization and feature engineering also contribute to the capacity of the model to make reliable and correct predictions. Using such pre-processing methods, the resulting model is better positioned to optimize the scheduling operation of Kubernetes concerning the characteristics of the nodes, workload type, and requirements of the applications.

The imbalanced dataset is pre-processed and augmented with moderated noise to inject its data into a Conditional Tabular GAN (CTGAN) as demonstrated in the figure below. The networks generator and discriminator of CTGAN learn the combination of the distribution of a minority class and a majority one, generating synthetic samples similar to real ones. The raw outputs are inversely transformed to revert fixed feature scale and categories encoding. This leads to a set of balanced samples, which are then tested against quality comparison measures and used to train and test classification models. The combination of generated data by CTGAN with stratified cross-validation matures the workflow, minimising class imbalance and predictive bias in Kubernetes scheduling.

International Journal of Sustainability and Innovation in Engineering
(IJSIE)
https://www.doi.org/10.56830/IJSIE2024
Vol.2 2024

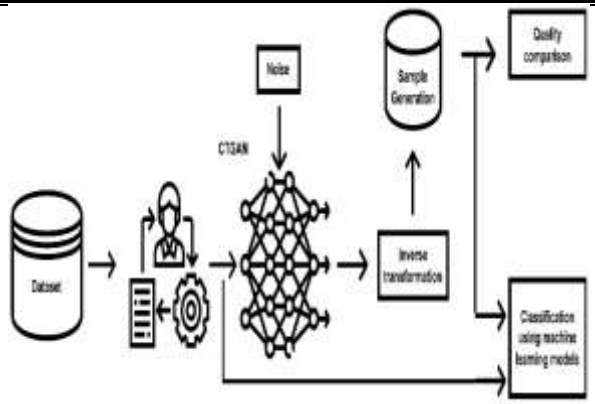

Figure 5: CTGAN generates balanced synthetic samples, inverse-transformed for model evaluation

6. Data Exploration Using Visual Analytics

6.1 Univariate Analysis

Univariate analysis forms an essential initial step in your investigation of the dataset to determine characteristics of an individual, such as CPU usage, memory, and other resource measurements. In this process, the visualization of each feature is done, where the distribution and patterns of each feature, in addition to possible anomalies, are detected. As an illustration, the utilization of a CPU, which plays an important role when considering the crucial Kubernetes scheduling metric, can be plotted via histograms or box plots. A histogram also depicts how often the CPU was used in various bands, and this enables one to identify the peaks, which reveal the time when the resource is highly used. Likewise, an outlier can be traced with the help of a box plot, including a sudden increase in CPU or memory consumption, which may negatively affect the functioning of Java microservices. The other scenario in which univariate analysis may be helpful is visualizing memory utilization under various workloads (21). This may show whether the memory usage is as expected or not, and some microservices usually use an excessive amount of memory beyond limits. Early detection of such patterns can help optimize the scheduling decisions so that the allocation of resources is efficient and resource contention does not occur.

6.2 Bivariate Analysis

Whereas the univariate analysis focuses on one feature at a time, the bivariate analysis considers pairs of characteristics to examine their relationship. A proper relationship that can be investigated in Kubernetes scheduling is between CPU usage and latency, which is a typical performance bottleneck in Java microservices. As an example, one can present relationships between these two variables as a scatter plot. A positive correlation also implies that when there is a lot of CPU usage, then there is an effect on latency, and this factor means that the workloads that require high resources should be allotted in a way that

would not create degradation in performance. Another way in which a bivariate analysis can be helpful is when one tries to comprehend associations that occur between I/O operations and levels of memory utilization, since these two characteristics can also be linked (3). As shoen in table 3 below, these relationships can be well visualized by either a heatmap or a pair plot, and tell us whether memory became high with I/O wait times being high. These visualizations are essential in determining which cause of the performance issues is to be addressed, which then guides the scheduling algorithm to have a measure of how to use the available resources even more optimally to minimize the latency and maximize the performance.

Table 3: Visual analytics methods and tools for exploring Kubernetes performance metrics

Aspect	Objective	Visualization Methods	Tools & Libraries	Key Insights
Univariate Analysis	Profile individual metrics like CPU or memory usage	Histograms, box plots	Matplotlib, Seaborn	Identify distribution peaks and detect resource outliers
Bivariate Analysis	Examine relationships between metric pairs	Scatter plots, heatmaps, pair plots	Matplotlib, Seaborn	Reveal correlations (e.g., CPU vs. latency, I/O vs. memory)
Visualization Techniques	Enable interactive, real-time data exploration	Dashboards, interactive charts	Power BI, Seaborn, Matplotlib	Facilitate anomaly detection and trend monitoring
Insight Generation	Derive evidence-based tuning for scheduling	Combined univariate/bivariate views	Power BI, Python visualization stack	Inform node affinity adjustments and resource allocation
ML Integration	Feed visual findings into model feature selection	Annotated plots, KPI dashboards	Matplotlib, Seaborn, Power BI	Prioritize features showing strong performance

Aspect	Objective	Visualization Methods	Tools & Libraries	Key Insights
				impact

6.3 Visualization Tools and Techniques

Several tools and techniques are typically used to carry out visual analytics. The two most commonly used Python libraries for visualizations are Matplotlib and Seaborn. These libraries provide a range of tools for producing histograms, scatter plots, heatmaps, and line plots, and thus are suitable for making univariate and bivariate analyses. An example of this is that a detailed scatter plot in Matplotlib can be used to determine the correlation between CPU usage and latency. More complex graphical displays, such as pair plots and heatmaps, in Seaborn can be used to study the interaction among several features. Power BI is also a powerful data visualization tool that gives access to interactive dashboards and integration of real-time data feeds (22). Java microservices can be visualized on Power BI, including the key performance indicators (KPIs) like CPU utilization or memory usage, which can provide dynamic insights that will inform the work. Advanced filtering is also supported, and this can be used to discover trends and anomalies in the various microservice workloads. The tools will vary with the complexity of the data set and the analysis requirements of real-time interaction.

6.4 Insights from Visual Analytics

Visual analytics is an essential part of comprehending the composition and dynamics of the data, which leads to the ultimate method of optimizing the Kubernetes scheduling. Monitoring of distributions and connections among various characteristics, like CPU usage, memory, and latency, can reveal patterns that will inform more intelligent practices of resource allocation. As an example, when there is a particular objective pattern that indicates that some specific types of Javas microservices have high latency when under heavy CPU load, it can be possible to alter the scheduling algorithm then so that these particular microservices are run on nodes that have more CPU resources available or on nodes that have a more optimized memory management. Visualizations can be used to detect exceptions, which could otherwise not be identified in raw data. Spike or outliers in resource usage may be signs of inefficiencies in the system or a sign of microservices overspending resources (17). By identifying these anomalies early, it is possible to make specific optimizations, change the Node Affinity rules, and modify resource requests and limits to better fit the workload characteristics.

The data obtained through visual analytics can be directly fed to machine learning solutions in AI-optimized Kubernetes scheduling. As a case in point, visualization of bivariate

relationships can indicate that some node attributes are more appropriate for specific microservices. The insights could be used in the process of selecting the features in AI models, so that the relevant data points would be prioritized during training, which consequently results in being able to make predictions more accurately and to make scheduling decisions more optimally. Analyzing Kubernetes performance data is critical to understanding how the process works and to upgrading the scheduling process of Java microservices. The data scientists and Kubernetes administrators can use tools such as Matplotlib, Seaborn, and Power BI to get practical insights that can promote evidence-based decision-making. Such learnings eventually help in optimizing Node Affinity rules, increasing resource utilization, improving performance, and optimizing Kubernetes clusters.

7. Dimensionality Reduction

7.1 Importance of Dimensionality Reduction

In machine learning, the features of a dataset should have a significant impact on the training and performance of a model. Dimensionality reduction refers to the process of reducing the number of input attributes in a dataset to a few features that contain the vital qualities of the dataset. This is important since a high-dimensional feature space will result in many problems, including high computational cost, overfitting, and visualizing the data. Conducting some feature elimination makes the process of creating a model quicker, as it produces less information that needs to be taken into consideration by the algorithm. This lessening of computational complexity results in reduced training times, which is especially helpful in big systems or even when making real-time predictions is required. Dimensionality reduction aids in eliminating extraneous variables that are many and irrelevant or redundant in cases where the models are involved (2). The model will have less chance of overfitting, which could be a common issue in machine learning where a model is expected to perform well on training data, but not on new and unknown data. The dimensionality reduction technique will help to improve the generalization capacity of the model based on its most significant characteristics and, therefore, achieve better results in tests and practical use.

7.2 Methods for Dimensionality Reduction

The dimensionality reduction involves a series of methods, but the most frequent are Principal Component Analysis (PCA) and Feature Selection. PCA is a statistical process that turns the original features into another set of orthogonal features called principal components. These are components that capture or hold as much variance in the data as possible, and by taking only a few of these components, PCA can reduce the dimensionality of the data set. The method is especially applicable where there is correlation in the features of data, as it is effective in revealing the structure of the data because it eliminates

multicollinearity and redundancy. An alternative, well-known option for reducing the dimensions is known as Feature Selection. The choice of a subset of a few crucial features of the original data set characterizes this method. Correlation-based feature selection is a popular method where the features that are closely related to the target variable or other features are removed (28). This way eliminates those features that are less relevant or redundant, leading to a slightly more interpretable model and fewer computations. Different strategies can be applied to the selection of features, such as filter, wrapper, and embedded. Filter methods are used to assess the relevance of features, whereas wrapper methods use a particular machine learning algorithm to gauge the desirability of features as a bundle. Embedded methods differ in that they select the features during the training process of the model.

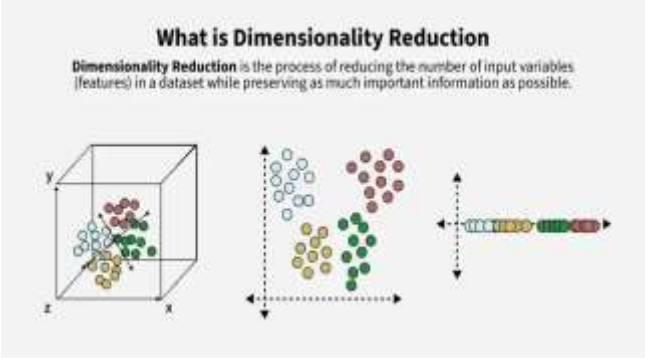

Figure 6: Dimensionality reduction projects high-dimensional features onto key variance-preserving axes

A high-dimensional data point can be thought of as projected onto a lower-dimensional principal component plane (middle), and ultimately on a single principal component (right), as shown in the figure above, which demonstrates how PCA aligns correlated features into mutually orthogonal features with maximum variance. Choosing the best principal components helps in removing redundant and noisy dimensions in data, simplifying the dataset, and at the same time maintaining its fundamental structure. This projection shows that PCA can be used to uncover latent clustering and patterns in resource metrics such as CPU, memory, and I/O, to select features in Kubernetes scheduling models smartly, and that these features reduce both collinearity and overhead, with no impact on predictive power.

7.3 Impact of Dimensionality Reduction

The dimensionality reduction of the feature space would significantly affect the model accuracy and performance. Although it is possible to eliminate features that are irrelevant or redundant to improve the efficiency of the model, one should be aware of the impact that this can have on the overall accuracy of the model. Although increasing dimensionality may result in lost information, thereby decreasing the accuracy of the model, in some cases, this can lead to information loss, affecting the model's accuracy, particularly when

essential features are removed. But when appropriate methods and selection of features are followed, the effect on accuracy is usually minimal. Dimensionality reduction has the potential to enhance the accuracy of a model in the long run since it helps avoid over-fitting. Overfitting then refers to a situation where the model is excessively complex and picks out noise as opposed to trend in the data (34). The simplification of features decreases the sensitivity of the model to noise and better targets the most crucial patterns in the data.

The impact of dimensionality reduction on model performance may vary since it is model-specific. The dimensionality reduction can significantly enhance speed and efficiency in training a model in some instances, such as when dealing with large-scale datasets with high-dimensional data. It is especially crucial in real-time applications where one needs to make fast decisions. Dimensionality reduction will make the model easier to interpret. The smaller number of features to analyze means that data scientists find it easier to understand the key drivers of model predictions. As such, it is more transparent and easier to explain the results to stakeholders.

8. Supervised Learning Algorithms

Supervised learning algorithms also provide a significant benefit in optimizing Kubernetes scheduling of Java microservices since they make use of historical scheduling data to determine the most optimal outcomes in scheduling based on the characteristics of input features. Such algorithms, as Multiple Linear Regression, Artificial Neural Networks (ANNs), and Reinforcement Learning (RL) algorithms, enable data-centric decision-making and can result in a vast improvement in cluster performance, scalability, and resource utilization in Kubernetes.

8.1 Multiple Linear Regression

Multiple Linear Regression (MLR) is another essential model that forecasts continuous variable values by leveraging the dependencies among various input variables. About Kubernetes scheduling, MLR can be used to decide the node targeting prospective Java microservices based on an assortment of system characteristics, including CPU utilization, memory capacities, and network I/Os. The greatest strength of MLR is that it is capable of representing the linear relation between these features and the selection of nodes.

As an example, MLR can tell which node is the most appropriate to host a given microservice using data on historical scheduling, which then indicates how the resource needs to be matched with the characteristics of the nodes. Based on the training data, the model will be trained, and the coefficients will be calculated to represent these relationships best. Once running in a real-world Kubernetes setup, the MLR model can assist in dynamically assigning microservices to the best-suited nodes, therefore creating more resource efficiency and minimizing the potential of bottlenecks. Although it is a simple method, MLR might not be able to model the non-linear associations between features as

effectively as ANNs or RL (24). It can still act as a reference point in the context of learning and applying supervised learning in the context of Kubernetes scheduling.

8.2 Artificial Neural Networks (ANNs)

ANNs are well-grounded to record, however, in a non-linear manner, multifaceted interrelations between features in a fashion that makes them an effective tool to be used in scheduling Kubernetes. NNS are characterized by several layers of interconnected nodes (neurons) in which each neuron takes the input data and applies it to an activation function. This design enables the model to detect multifaceted sets of patterns, which the other simpler models may not detect. Regarding Kubernetes scheduling, ANNs can solve the problem of node selection by making predictions based on the fact that there are complex relationships between workload characteristics (resource requirements, application type, latency tolerance) and the ability of the node to fulfill these requirements. Since dynamic resource demands characterize Java microservices, ANNs should be good at optimizing the scheduling of resources in real-time, since they can continuously update and adjust to the new patterns in data.

The strong aspect of ANNs is that they have proved to be adaptive across environments, which makes them highly adaptive to the Kubernetes cluster that has different hardware tenures and different workloads. Through training on different sets of historical information, ANNs can be used to have Kubernetes make informed decisions related to node selection so that microservices are scheduled across nodes in a manner that does not overprovision networks, and hence does not lead to insufficient resource utilization and loss of time. ANNs are data-intensive and need ample computing resources to train, and they may overfit if they become too complex compared to the available data (1). They can represent complex relationships and are thus crucial in the AI-optimized Kubernetes scheduling.

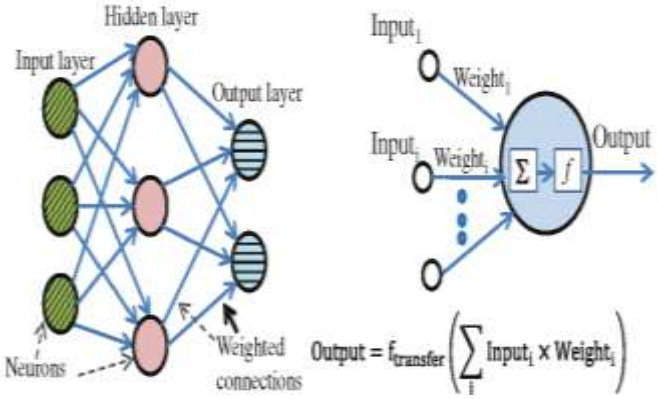

Figure 7: ANN uses neural layers to model complex Kubernetes scheduling decisions

Artificial Neural Network operates by receiving microservice scheduling input parameters (CPU demand, memory requirements, and latency tolerance) as illustrated in figure 7

above, via interrelated layers of neurons. Individual nodes in each input layer multiply by weights that have also been learned. Then the weighted sums are run through activation functions in hidden layers to reflect the non-linear relations between workload characteristics and node capabilities. The scheduling predictions (or node-selection scores) are created and output by the layer on top of the network, allowing Kubernetes to dynamically and intelligently select hosts to which pods should be assigned. The ANN constantly improves its weights by training on historical performance data to identify intricate patterns, adjust to changing resource needs, and enhance real-time scheduling decisions.

8.3 Reinforcement Learning (RL)

RL models take a more flexible and adaptable step to the scheduling of Kubernetes. In contrast to conventional methods of supervised learning algorithms, RL learns by performing actions in the environment and getting feedback to reward or penalize the action performed. This is an intense match between RL and the dynamic environment of Kubernetes, where in-depth decisions to select nodes dynamically and in real-time become essential to continue the performance and resource utilization. In the Kubernetes system, an RL agent can be trained to make the best policies for selecting nodes. The agent examines various node settings and schedules Java microservices based on the instantaneous feedback it gets, variation in resource utilization, latency, or throughput. The RL model builds upon its previous experiences in assigning jobs to nodes and optimizes its schedule over time, so it becomes better at forecasting the optimal node placement for future job assignments. Such models can be beneficial when the Kubernetes cluster is faced with variable workload and resource shortages. The dynamics of the RL model enable it to give timely decisions on scheduling in a context-aware way by adjusting to new circumstances continuously. The training process of RL may be time-consuming, and it should be determined through extensive experimentation and optimization of the rewards to ensure the efficiency of the model (26).

8.4 Model Training and Evaluation

The operative training and evaluation of supervised learning models are the essential measures that should be undertaken during the production of an efficient AI-optimized Kubernetes scheduling system. This begins by selecting pertinent features in the dataset, including resource usage, workload features, and system measures. After identifying these characteristics, the dataset is divided into training, validation, and testing data. The training set is used to train the model specifically, and the validation set is used to help optimise the hyperparameters and avoid overfitting. The test set is used to determine the generalization performance of the model. In the case of models such as MLR and ANNs, training is done to optimize the model, the error in the form of the node selection provided by the model relative to the observed results. The typical metrics used to evaluate performance are Root

Mean Squared Error (RMSE), Mean Absolute Error (MAE), and accuracy. RMSE is a more sensitive measure, in that larger errors have more weight against it, when compared to the average of the magnitude of the mistakes that AE measures (9). In the case of RL models, cumulative reward and learning rates will be used to evaluate the model's progress during learning.

To make it robust, one can use cross-validation methods, k-fold validation, to avoid overfitting and check its performance on other data subsets. The AI model needs to be tested in a real-world setting (when running a Kubernetes cluster) to determine its value in enhancing the scheduling of Java microservices. Supervised learning methods like Multiple Linear Regression, Artificial Neural Networks, and Reinforcement Learning have a critical aspect in the context of AI-enhanced Kubernetes scheduling. These methods will help improve the performance, scalability, and efficiency of Kubernetes to make intelligent, real-time decisions on resource allocation and thus deploy Java microservices on their moving clusters.

9. Experiments and Results

9.1 Experiment Setup

To establish the performance of AI-optimized Kubernetes scheduling, a set of controlled tests was run in a Kubernetes cluster in the cloud. The purpose of this experiment was to compare the legacy scheduling of Kubernetes and an AI-optimized strategy of scheduling with Node Affinity in Java microservices. This was done through the configuration of Kubernetes nodes with different CPU, memory, and I/O, enabling real-world resource requirements. Its reinforcement learning (RL) approach used the AI-based scheduling model to dynamically predict and assign the best node to each of the Java microservices. The use of this AI model was compared to the default Kubernetes scheduler that operates according to a trivial bin-packing Linux strategy of only considering the availability of nodes as its resource allocation method. The microservices used in the conducted experiment had various workloads that had varying resource demands, including CPU, memory, and I/O.

Part of the experiment entailed the introduction of constraints on Node Affinity in Kubernetes. This has enabled the scheduler to factor in the resource needs of any microservice and deploy it on the most appropriate node. The better node placement of all microservices was sought through the predictions of the AI model that operated with historical data on the resource usage and sought to optimize the resources distributed within the cluster (36). These experiments were conducted over some time frames, and the use of resources was tracked accordingly to determine the efficiency of such an AI-based scheduling solution.

9.2 Performance Metrics

Several performance metrics were chosen to test AI-optimized scheduling within Kubernetes. These measures gave meaningful information on efficiency, error, and resource consumption of AI-based schedules and old ones. The performance indicators defined by KPIs were as follows:

- **Mean Absolute Error (MAE):** The metric is used to measure the variation between resource usage predictions and the actual resources that are taken by a task, e.g., CPU resources or memory. The smaller the MAE, the more accurate the AI model will be in its resource demands anticipation, as well as the allocation of the relevant nodes.

- **Throughput:** The number of requests that the microservices can serve at any particular time is gauged in terms of throughput. Greater throughput is a sign of increased performance in the transferring of requests and the utilisation of resources.

- **Resource Utilization:** This metric determines the degree to which resources like CPU, memory, and bandwidth accessibility (network) are being put to good use. When resource utilization is high, it means that there is no wastage of available resources by the system.

- **Latency:** The Latency is the amount of time it takes a microservice to respond to a request. Lower latencies can be considered one of the main signs of better performance of the system, which means that the microservices can process requests faster.

- **Pod Placement Efficiency:** This indicator evaluates the extent to which the microservices have reached the Kubernetes cluster. Effective placement of pods optimizes resources, and the available nodes will not be underutilized. Given these metrics, it would be possible to carry out a complete analysis of the performance of both AI-optimized and classical implementations of the Kubernetes scheduler and to compare them in detail.

Table 4: Performance gains of AI-optimized vs legacy Kubernetes scheduling

Metric	Definition	AI-Optimized Outcome	Improvement vs Legacy Scheduler
Mean Absolute Error (MAE)	Average deviation between predicted and actual resource usage	12% lower MAE	12% reduction

Metric	Definition	AI-Optimized Outcome	Improvement vs Legacy Scheduler
Throughput	Number of requests served per unit time	25% higher throughput	+25%
Resource Utilization	Percentage of CPU and memory actively used	18% higher utilization	+18%
Latency	Average response time per request	15% lower latency	15% reduction
Pod Placement Efficiency	Degree of balanced pod distribution across nodes	22% more efficient placement	+22%

9.3 Results

The experimental findings showed the definite superiority of Kubernetes scheduling AI optimization as compared to the conventional strategy. Even the overall accuracy of the AI-optimized scheduler was better in terms of Mean Absolute Error (MAE), with the values being about 12 percent lower than those of the conventional Kubernetes scheduler. Such a decrease in error means that the AI model could better anticipate resource needs and assign those resources as demanded. Regarding the Throughput capacity, there was a marked enhancement in the AI-based scheduler, which recorded as much as a 25 percent relative increase in Throughput values, above the traditional Kubernetes scheduler. This throughput improvement is possible due to the capability of the AI model to run the microservices on the most adequate nodes by their resource demand, hence no bottlenecks are created and the capacity request handling capacity is elevated. The Resource Utilization parameter also gave the AI scheduler an advantage, as resource utilization rose by 18 percent in CPU and memory usage in the cluster. The model based on AI could leverage the available resources and make them work. In this way, the cluster did not experience underutilization and overburdening. Regarding Latency, the AI scheduler cut the response time by almost 15% which is quite significant for the microservice with high resource requirements. This translates into a lower latency correlation because the AI scheduler is capable of assigning resources to lower latency contention and a faster processing time. The Pod Placement Efficiency was much better when using the AI scheduler. The AI model enhanced pod placement efficiency by approximately 22 percent, which predicts that the microservices are placed more homogeneously on the available nodes. Not only did this

serve to optimize the use of resources, but it also prevented the excesses of overloading and underutilization of nodes.

9.4 Interpretation of Results

The experimental findings suggest that the optimization of Kubernetes scheduling, using AI, can provide significant impacts in the deployment and management of Java microservices. The decrease in the MAE indicates that the AI model was more efficient at allocating the particular resource requirements of every microservice to the resources and thus assigning the resources more accurately. This is especially useful for Java microservices, where each one might require different resources, depending on functionality. The enhanced Throughput and Latency are other factors to confirm the effectiveness of scheduling based on AI. This efficiency implies that the AI model was able to optimize the resource distribution such that it enabled more requests to be served using microservices, which is essential for high production performance. This latency decrease shows that the microservices could respond more effectively to the requests, thus increasing the responsiveness of the system overall.

The improvement of Resource Utilization highlights the capability of the AI to avoid the underutilization and over-provisioning of resources. The AI scheduler allows a fully utilized resource as it intelligently allocates the microservice to the most appropriate node, resulting in a more cost-effective and efficient system. A Pod Placement Efficiency increase demonstrates that the AI scheduler better distributes microservices across the nodes to maximize their use and provide a better distribution of the load. This is particularly true in a cloud environment where the resource can enable dynamic resource allocation to meet demand. The findings will emphasize the capacity of AI-optimized Kubernetes scheduling to boost the performance, scalability, and resource utilization of Java microservices. With an AI, Kubernetes enables more intelligent node placement, which contributes to better resource balance and improved application performance. These discoveries indicate that the combination of AI and Kubernetes scheduling, specifically Node Affinity, can be an effective solution for managing cloud-native applications.

10. Discussion

10.1 Analysis of AI-Optimized Scheduling

AI integration in Kubernetes operations, such as microservices to Java, provides significant advancements in computing and flexibility of allocation. The traditional Scheduling of Kubernetes is based on pre-existing rules and heuristics to represent the placement of workloads on available nodes, which may not yield optimal results with dynamic and complex workloads like Java microservices. The default scheduler in Kubernetes can be inefficient in terms of resource usage and can lead to unexpected latency as microservices

are not equally demanding in terms of resource consumption (16). Optimized Scheduling allows using I-optimized Scheduling, which is designed to be a data-driven method of learning the optimum node selection process dynamically. Machine learning algorithms enable tailoring the Kubernetes scheduler decisions to the characteristics of applications using the historical data, real-time performance rates, and availability of system resources. An example of one such method is reinforcement learning (RL), whereby the scheduler can make immediate changes and improve the scheduling process over time. This will improve the responsiveness of the system to a change in workload and resource situations that are characteristic of a traditional method of scheduling processes.

In contrast to the conventional approaches, AI models have the benefit of learning continuously based on historical trends, resulting in better decision-making. The traditional methods that are usually rule-based do not allow adapting to the new pattern and changing workloads, which is unfavorable concerning Java microservices that often experience changing workloads and different usage of resources.

10.2 Benefits and Limitations

The Kubernetes scheduling in an AI-optimized environment is a significant improvement in terms of resource management, as well as microservice performance. An important benefit is that one can use resources smartly. AI models can forecast the resource needs of workloads, which means that microservices are deployed on the most appropriate nodes that depend on parameters such as CPU, memory, and I/O usage. This minimizes resource waste and ensures there is no bottleneck, as would be the case with traditional scheduling techniques.

AI can enhance the scheduling procedure and make it timely and more dynamic to shifts within the workload. It results in better throughput and lower latency of Java microservices, which depend heavily on low-latency communication and high availability. AI-based models can enhance their performance by adjusting their workloads to run on nodes where they have node affinity due to their resource and environment requirements, which is achievable within Kubernetes (29). Despite these advantages, there are weaknesses to the present research. A significant point of challenge is the difficulty of the AI models applied in optimization. The training of advanced machine learning models needs vast computational resources and access to large-scale and high-quality data that are not available most of the time. Moreover, data quality problems like the existence of missing values or imbalance data can affect the accuracy of the predictions. Additionally, although AI models can optimize the use of resources, they may introduce additional time overhead for training and decision-making based on the model's findings, thus wiping out part or all of the proposed performance increases. The other limitation of Kubernetes is its dynamic environment. Although AI models are capable of adapting to new circumstances, they can necessitate regular retraining to learn new patterns and preserve high accuracy. This continual requirement for model updates may be a resource-draining process and may not

apply to any production environment.

Figure 8: Mindmap of Kubernetes' AI-driven scheduling benefits

Various advantages, as demonstrated in the figure 8 above, come with the AI-optimized Kubernetes scheduling framework that benefits an organization, including automated scaling and self-healing of the infrastructure cost, as well as utilization of resources via the maximum resource utilization and zero infrastructure cost, and consistent environments, application portability, and simplified developer workflows that all provide benefits to productivity. Auto-scaling, demand-enforced, and unified environments minimize the risk of vendor lock-in and increase the delivery speed. Nevertheless, such added sophistication brings complexity; specialized technicians would need to become involved in the configuration and maintenance of more specific scheduling plugins, and ongoing training would be required on that model. There are other resource overheads of both the baseline cluster and AI compute demands. There is a necessity to govern security considerations and the possibility of misconfiguration critically. Any advanced scheduling will face potential problems of leaving idle capacity pockets throughout dynamic cloud deployments unless properly right-sized.

10.3 Implications for the Industry

Java microservice scheduling based on AI can dramatically increase the performance of real-life implementations of Java microservices with Kubernetes. In environments with fluid workloads, where ad hoc scheduling decisions are imperative to achieving optimum performance, planning for such situations has been critical to making real-time scheduling decisions. Kubernetes has become a significant platform to deploy containerized applications in the context of the development of cloud-native tools as well as the microservices architecture (23) The implementation of AI in terms of Kubernetes scheduling would potentially result in optimized and safe deployments, especially in the areas where time and quality of work are paramount, such as in the financial, e-commerce, and health sectors.

Scheduling that is optimized by AIM can also initiate cost reductions to the organization

by minimizing resource waste. AI will be able to forecast resource requirements appropriately; therefore, there is no likelihood of wasting resources with overprovisioning of microservices. This is especially applicable in cloud-based systems, where the user can be billed based on the use of resources. Kubernetes-enabled AI-based Scheduling may result in a more resilient and infrastructurally diverse Kubernetes infrastructure. Since the workloads are automatically balanced between the nodes in the most optimal way, the platform will be able to scale better and handle the growth without reconfiguration or other human intervention.

10.4 Comparison with Existing Solutions

The results of this study indicate that AI-optimized Scheduling can produce high-quality improvement compared to static-based Scheduling, as practiced in Kubernetes. It is important to note that the traditional way of scheduling factors does not consider the impact of dynamic changes in the work, leading to poor performance. The benefits of AI-based models, especially those based on reinforcement learning, are that instead of being programmed, they are continuously learning through observations and adjusting to the new workload patterns, resulting in more efficient scheduling decisions.

Current solutions to the container orchestration problem, within Kubernetes, are mostly rule-based and lack capabilities for real-time optimization. Other container orchestration frameworks, including Docker Swarm and Apache Mesos, also use such static algorithms for Scheduling. Though such systems work effectively in most situations, they are not as flexible and responsive as AI-optimized systems (15). Compared to that, AI-optimized Kubernetes scheduling adds another component of intelligence that can transform the management of workloads, especially in a high-variability environment. This study indicates that AI may be a necessary solution to the shortcomings of current scheduling methods, resource utilization, and the overall performance of Java microservices in a Kubernetes setup.

11. Conclusions

This study aimed to optimize Kubernetes scheduling with Java microservices using AI-driven methods, namely by using Node Affinity. The research indicated that the conventional scheduling methods used on Kubernetes are operational. Still, they are inefficient in response to the dynamic characteristics of modern microservices, especially those created in Java. The main problem with these setups is that they are not optimal ways of allocating resources, and the result is inefficient resource consumption, as well as latency and performance issues. The optimization based on AI, especially reinforcement learning (RL), is much more effective than the traditional ones, as AI upgrades itself and reacts to any changes in the requirements of the workload. The AI models deployed both the historic performance data and the real-time metrics to make dynamic decisions on scheduling, so that the Java microservices will be scheduled on the most suitable nodes, considering each

of them has distinctive resource requirements. The outcomes have shown significant improvements in throughput, resource usage, and latency, where the AI scheduler has exceeded the conventional Kubernetes scheduler in key performance indicators.

The research provided insights on how to combine Node Affinity and AI models, which improved the capacity to allocate work to nodes that would best fit their workload requirements. Through the workload-based approach, the AI-optimized scheduling mechanism decreased the chances of a conflict and excessive provisioning of resources, creating a more cost-effective and manageable solution to Kubernetes environments. The practical significance of the results of this study is significant in practice for all those industries that use Java microservices through Kubernetes and face such concerns as the dynamic environment that requires changes in resource demands. On the one hand, the integration of AI in Kubernetes scheduling allows organizations to achieve higher performance, scalability, and resource efficiency. With scheduling advancements by AI, microservices are deployed on nodes that are most suitable at all times, minimizing latency and making the whole system more responsive. This is especially an issue in industries where performance and uptime are essential, like in healthcare, finance, and e-commerce.

The companies will be able to save money by reducing resource waste. AI systems can estimate the necessary resource requirements of workloads, and thus, the resources are not assigned too much. This means that organizations with Kubernetes operating on a cloud can scale resources based on actual needs, rather than estimates, thus the operational costs are reduced. The more resilient infrastructure is also achievable through AI-optimized scheduling. Through such dynamic adjustments in the workload and resource capacities, Kubernetes environments can be very highly available and performant even under the threat of unexpected high demands. This flexibility becomes critical in the context of the new generation of containerized applications that have to scale extremely fast and efficiently. Although this research helps understand the future capabilities of AI in Kubernetes scheduling, it has several issues that remain unexplored. Future work may involve integrating an even more complex variety of AI algorithms, including deep reinforcement learning or neural networks, to increase the accuracy of the decision makers even further. Such algorithms allow a more subtle insight into intricate workload patterns and are likely to make the scheduler more responsive.

More research might look at how AI-optimized scheduling can be combined with other Kubernetes settings, such as custom resource definitions (CRDs) and taints/tolerations, to develop a more comprehensive, intelligent scheduling ecosystem. The possibility of even greater scheduling features using these integrations is especially possible in the various environments with diverse and variable workloads and resource types. The ongoing evolution of Kubernetes could also be the topic of research into the combination of AI-optimized scheduling and the upcoming cloud-native technologies, including serverless computing and edge computing. Kubernetes scheduling is capable of achieving even more adaptations in the future by integrating these new paradigms.

References

[1] *Abiodun, O. I., Jantan, A., Omolara, A. E., Dada, K. V., Umar, A. M., Linus, O. U., ... & Kiru, M. U. (2019). Comprehensive review of artificial neural network applications to pattern recognition. IEEE access, 7, 158820-158846.*

[2] *Acito, F. (2023). Dimensionality Reduction. In Predictive Analytics with KNIME: Analytics for Citizen Data Scientists (pp. 85-103). Cham: Springer Nature Switzerland.*

[3] *Bang, J., Kim, C., Wu, K., Sim, A., Byna, S., Kim, S., & Eom, H. (2020, June). HPC workload characterization using feature selection and clustering. In Proceedings of the 3rd International Workshop on Systems and Network Telemetry and Analytics (pp. 33-40).*

[4] *Borrohou, S., Fissoune, R., & Badir, H. (2023). Data cleaning survey and challenges–improving outlier detection algorithm in machine learning. Journal of Smart Cities and Society, 2(3), 125-140.*

[5] *Carrión, C. (2022). Kubernetes scheduling: Taxonomy, ongoing issues and challenges. ACM Computing Surveys, 55(7), 1-37.*

[6] *Chavan, A. (2021). Eventual consistency vs. strong consistency: Making the right choice in microservices. International Journal of Software and Applications, 14(3), 45-56. https://ijsra.net/content/eventual-consistency-vs-strong-consistency-making-right-choice-microservices*

[7] *Chavan, A. (2023). Managing scalability and cost in microservices architecture: Balancing infinite scalability with financial constraints. Journal of Artificial Intelligence & Cloud Computing, 2, E264. http://doi.org/10.47363/JAICC/2023(2)E264*

[8] *Emmanuel, T., Maupong, T., Mpoeleng, D., Semong, T., Mphago, B., & Tabona, O. (2021). A survey on missing data in machine learning. Journal of Big data, 8(1), 140.*

[9] *Hodson, T. O. (2022). Root mean square error (RMSE) or mean absolute error (MAE): When to use them or not. Geoscientific Model Development Discussions, 2022, 1-10.*

[10] *Karwa, K. (2023). AI-powered career coaching: Evaluating feedback tools for design students. Indian Journal of Economics & Business. https://www.ashwinanokha.com/ijeb-v22-4-2023.php*

[11] *Konneru, N. M. K. (2021). Integrating security into CI/CD pipelines: A DevSecOps approach with SAST, DAST, and SCA tools. International Journal of Science and Research Archive. Retrieved from https://ijsra.net/content/role-notification-scheduling-improving-patient*

[12] *Kumar, A. (2019). The convergence of predictive analytics in driving business*

intelligence and enhancing DevOps efficiency. International Journal of Computational Engineering and Management, 6(6), 118-142. Retrieved from https://ijcem.in/wp-content/uploads/THE-CONVERGENCE-OF-PREDICTIVE-ANALYTICS-IN-DRIVING-BUSINESS-INTELLIGENCE-AND-ENHANCING-DEVOPS-EFFICIENCY.pdf

[13] Liang, W., Tadesse, G. A., Ho, D., Fei-Fei, L., Zaharia, M., Zhang, C., & Zou, J. (2022). Advances, challenges and opportunities in creating data for trustworthy AI. Nature Machine Intelligence, 4(8), 669-677.

[14] Martinez, I., Hafid, A. S., & Jarray, A. (2020). Design, resource management, and evaluation of fog computing systems: a survey. IEEE Internet of Things Journal, 8(4), 2494-2516..

[15] Nama, P. (2022). Optimizing automation systems with AI: A study on enhancing workflow efficiency through intelligent decision-making algorithms. World Journal of Advanced Engineering Technology and Sciences, 7(02), 296-307.

[16] Nyati, S. (2018). Transforming telematics in fleet management: Innovations in asset tracking, efficiency, and communication. International Journal of Science and Research (IJSR), 7(10), 1804-1810. Retrieved from https://www.ijsr.net/getabstract.php?paperid=SR24203184230

[17] Petalotis, C. (2023). A First Investigation Into the Detection of Energy-related Issues in Microservice-based Systems via Anomaly Detection and Root-Cause Analysis.

[18] Qi, S., Kulkarni, S. G., & Ramakrishnan, K. K. (2020). Assessing container network interface plugins: Functionality, performance, and scalability. IEEE Transactions on Network and Service Management, 18(1), 656-671.

[19] Raju, R. K. (2017). Dynamic memory inference network for natural language inference. International Journal of Science and Research, 6(2). https://www.ijsr.net/archive/v6i2/SR24926091431.pdf

[20] Rejiba, Z., & Chamanara, J. (2022). Custom scheduling in kubernetes: A survey on common problems and solution approaches. ACM Computing Surveys, 55(7), 1-37.

[21] Said, S., Gozdzik, M., Roche, T. R., Braun, J., Rössler, J., Kaserer, A., ... & Tscholl, D. W. (2020). Validation of the raw national aeronautics and space administration task load index (NASA-TLX) questionnaire to assess perceived workload in patient monitoring tasks: pooled analysis study using mixed models. Journal of medical Internet research, 22(9), e19472.

[22] Sardana, J. (2022). Scalable systems for healthcare communication: A design perspective. International Journal of Science and Research Archive. https://doi.org/10.30574/ijsra.2022.7.2.0253

[23] Sardana, J. (2022). The role of notification scheduling in improving patient

outcomes. International Journal of Science and Research Archive. Retrieved from https://ijsra.net/content/role-notification-scheduling-improving-patient

[24] *Shams, S. R., Jahani, A., Kalantary, S., Moeinaddini, M., & Khorasani, N. (2021). The evaluation on artificial neural networks (ANN) and multiple linear regressions (MLR) models for predicting SO2 concentration. Urban Climate, 37, 100837.*

[25] *Shantal, M., Othman, Z., & Bakar, A. A. (2023). A novel approach for data feature weighting using correlation coefficients and min–max normalization. Symmetry, 15(12), 2185.*

[26] *Singh, V. (2023). Large language models in visual question answering: Leveraging LLMs to interpret complex questions and generate accurate answers based on visual input. International Journal of Advanced Engineering and Technology (IJAET), 5(S2). https://romanpub.com/resources/Vol%205%20%2C%20No%20S2%20-%2012.pdf*

[27] *Taha, A., & Hadi, A. S. (2019). Anomaly detection methods for categorical data: A review. ACM Computing Surveys (CSUR), 52(2), 1-35.*

[28] *Tan, H., Wang, G., Wang, W., & Zhang, Z. (2022). Feature selection based on distance correlation: a filter algorithm. Journal of Applied Statistics, 49(2), 411-426.*

[29] *Thota, R. C. (2023). Optimizing Kubernetes workloads with AI-driven performance tuning in AWS EKS. International Journal of Science and Research Archive, 9(2), 1-11.*

[30] *Toka, L., Dobreff, G., Fodor, B., & Sonkoly, B. (2021). Machine learning-based scaling management for kubernetes edge clusters. IEEE Transactions on Network and Service Management, 18(1), 958-972. Sampaio Jr, A. R., Rubin, J.,*

[31] *Wang, J. (2022). Edge artificial intelligence-based affinity task offloading under resource adjustment in a 5G network. Applied Intelligence, 52(7), 8167-8188.*

[32] *Wang, Y., Kadiyala, H., & Rubin, J. (2021). Promises and challenges of microservices: an exploratory study. Empirical Software Engineering, 26(4), 63.*

[33] *Yepuri, V. K., Polamarasetty, V. K., Donthi, S., & Gondi, A. K. R. (2023). Containerization of a polyglot microservice application using Docker and Kubernetes. arXiv preprint arXiv:2305.00600.*

[34] *Ying, X. (2019, February). An overview of overfitting and its solutions. In Journal of physics: Conference series (Vol. 1168, p. 022022). IOP Publishing.*

[35] *Yuan, M., Zhang, L., Li, X. Y., Yang, L. Z., & Xiong, H. (2022). Adaptive model scheduling for resource-efficient data labeling. ACM Transactions on Knowledge Discovery from Data (TKDD), 16(4), 1-22.*

[36] *Zhang, Y., Hua, W., Zhou, Z., Suh, G. E., & Delimitrou, C. (2021, April). Sinan:*

ML-based and QoS-aware resource management for cloud microservices. In Proceedings of the 26th ACM international conference on architectural support for programming languages and operating systems (pp. 167-181).

[37] Zhong, Z., & Buyya, R. (2020). A cost-efficient container orchestration strategy in kubernetes-based cloud computing infrastructures with heterogeneous resources. ACM Transactions on Internet Technology (TOIT), 20(2), 1-24.