ISSN: 2583-9888
DOI: doi.org/10.51219/JAIMLD/chandra-sekhar/172

URF PUBLISHERS
connect with research world

Journal of Artificial Intelligence, Machine Learning and Data Science
https://urfpublishers.com/journal/artificial-intelligence

Vol: 1 & Iss: 1 *Research Article*

Predictive and Corrective Machine Learning for Seamless Auto-Scaling and Auto-Deployment in Cloud-Native Microservices

Chandra Sekhar Veluru*

Citation: Veluru CS. Predictive and Corrective Machine Learning for Seamless Auto-Scaling and Auto-Deployment in Cloud-Native Microservices. *J Artif Intell Mach Learn & Data Sci 2022*, 1(1), 674-679. DOI: doi.org/10.51219/JAIMLD/chandra-sekhar/172

Received: 02 August, 2022; **Accepted:** 18 August, 2022; **Published:** 20 August, 2022

*__Corresponding author:__ Chandra Sekhar Veluru, USA, E-mail: chanduveluru@gmail.com

A B S T R A C T

Resource utilization and scalability are the two most prominent features in the dynamism of cloud-native environments. This white paper introduces predictive machine learning for auto-scaling and corrective machine learning for auto-deploying systems in connection with microservice-based software applications. Particular attention is given to seamless zero-downtime operation, assured with the help of state-of-the-art machine learning algorithms, Kubernetes, and an orchestration framework. The practical implementation architecture here will hopefully serve as a guide toward applying those concepts in real-world scenarios.

1. Introduction

Resource management is vital, considering cloud-native applications' current complexity and demand-driven nature. Traditional approaches to scaling usually waste many resources or need more capacity to handle load peaks. This white paper presents how integrating predictive corrective machine learning with microservices, Kubernetes, and a state-of-the-art orchestration framework can help with such challenges.

2. Predictive Machine Learning for Auto-Scaling

In predictive machine learning, the historical data analyzed is used to project future resource needs. By identifying patterns and trends in the data, machine learning models can predict or foresee when an application requires more resources and scale on demand.

Key Components

- **Data Collection**: Gathering metrics on CPU usage, memory consumption, network traffic, user requests, thread waiting times, and thread processing times.
- **Feature Engineering**: Identifying relevant features that influence resource demand.
- **Model Training**: Using historical data to train machine learning models that predict future resource needs.
- **Prediction and Scaling**: Implementing a feedback loop where predictions trigger auto-scaling actions.

Benefits

- **Efficiency**: Optimizes resource allocation, reducing costs associated with over-provisioning.
- **Performance**: Ensures applications have the necessary resources to handle peak loads.
- **Scalability**: Facilitates seamless scaling to accommodate varying workloads.

3. Corrective Machine Learning for Auto-Deployment

Corrective machine learning focuses on identifying and rectifying system anomalies that could lead to downtime. Corrective machine-learning models can automatically deploy corrective actions by continuously monitoring application performance and health.

Key Components

- **Detect deviations**: Using machine learning algorithms to detect deviations from normal behavior.
- **Root Cause Analysis**: Identifying the underlying causes of anomalies.
- **Automated Responses**: Deploying predefined corrective actions, such as restarting services or deploying additional instances.
- **Continuous Learning**: Refining models based on new data to improve accuracy and response times.

Benefits

- **Reliability**: Minimizes downtime by quickly addressing issues before they impact users.
- **Autonomy**: Reduces the need for manual intervention, allowing IT teams to focus on strategic initiatives.
- **Adaptability**: Continuously learns and adapts to changing application dynamics.

4. Implementation

An e-commerce microservice application that handles UI and API requests was considered to implement predictive and corrective auto-scaling. This application consists of multiple microservices, such as:

- **UI Service:** Manages the front-end user interface.
- **API Service:** Handles backend API requests for product information, user authentication, and order processing.
- **Order Service:** Manages order creation, updates, and status.
- **Product Service:** Handles product catalog and inventory management.
- **User Service:** Manages user data and authentication.

The sections below provide the bare minimum implementation of the white paper, which was used to implement predictive and corrective machine learning-based auto-scaling.

4.1 Data Collection

Zabbix: Collects metrics from each microservice.

Metrics include CPU usage, memory consumption, network traffic, user requests, thread waiting times, and thread processing times.

```
:zabbix_agent
enabled: true
host: zabbix.example.com
```

Fluentd: Aggregates logs from each microservice.

```
<source>
type forward@
port 24224
<source/>
<** match>
type file@
path /var/log/fluentd
<match/>
```

4.2 Feature Engineering

Apache Spark: Processes large-scale data for feature engineering.

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName(“FeatureEngineer-
()ing”).getOrCreate
(“logs = spark.read.json(“hdfs://path/to/logs
(“metrics = spark.read.json(“hdfs://path/to/metrics
Combine logs and metrics #
(“combined_data = logs.join(metrics, “timestamp
```

Pandas: Handles data manipulation and analysis

```
import pandas as pd
(“df = pd.read_json(“combined_data.json
[‘df[‘feature’] = df[‘metric1’] / df[‘metric2
```

Scikit-learn: Facilitates feature extraction and preprocessing.

```
from sklearn.preprocessing import StandardScaler
()scaler = StandardScaler
([[‘features = scaler.fit_transform(df[[‘feature1’, ‘feature2
```

4.3 Model Training

TensorFlow/PyTorch: Train machine learning models using historical data.

```
import tensorflow as tf
])model = tf.keras.models.Sequential
,(‘tf.keras.layers.Dense(128, activation=’relu
,(‘tf.keras.layers.Dense(64, activation=’relu
(tf.keras.layers.Dense(1
([
model.compile(optimizer=’adam’, loss=’mean_squared_
(‘error
(model.fit(features, labels, epochs=10
```

Apache Kafka: Streams data for real-time training and updates.

```
:kafka
bootstrap_servers: kafka.example.com:9092
topic: metrics
```

4.4 Prediction and Scaling

Kubernetes: Manages containerized applications.

```
apiVersion: apps/v1
kind: Deployment
:metadata
name: ecommerce-api
:spec
replicas: 3
:selector
:matchLabels
app: ecommerce-api
:template
:metadata
:labels
app: ecommerce-api
:spec
:containers
name: api -
image: ecommerce-api:latest
:ports
containerPort: 80 -
```

KubeFlow: Deploys machine learning workflows on Kubernetes.

```
apiVersion: kubeflow.org/v1
kind: TFJob
:metadata
name: ecommerce-predictor
:spec
:tfReplicaSpecs
:Worker
replicas: 3
:template
:spec
:containers
name: tensorflow -
image: tensorflow/tensorflow:latest
```

HPA (Horizontal Pod Autoscaler): Automatically scales the number of pods.

```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
:metadata
name: ecommerce-api-hpa
:spec
:scaleTargetRef
apiVersion: apps/v1
kind: Deployment
name: ecommerce-api
minReplicas: 3
maxReplicas: 10
targetCPUUtilizationPercentage: 50
```

4.5 Anomaly Detection

Elasticsearch, Logstash, Kibana (ELK Stack): Ingests, stores, and analyzes logs.

```
} input
} file
"path => "/var/log/fluentd/*.log
"start_position => "beginning
{
{
} filter
} grok
match => { "message" => "%{COMBINEDAPACHEL-
{ "{OG
{
{
} output
} elasticsearch
["hosts => ["localhost:9200
"index => "ecommerce-logs
{
{
```

TensorFlow: Detects anomalies in the system metrics.

```
])model = tf.keras.models.Sequential
tf.keras.layers.LSTM(128, input_shape=(timesteps,
,((features
(tf.keras.layers.Dense(1
([

model.compile(optimizer='adam', loss='mean_squared_er-
('ror
(model.fit(training_data, labels, epochs=10
```

4.6 Root Cause Analysis

Neo4j: Stores and analyzes relationships and dependencies.

```
(MATCH (s:Service)-[r:DEPENDS_ON]->(t:Service
'WHERE s.name = 'ecommerce-api
RETURN t
```

TensorFlow: Identifies patterns and correlations.

```
from tensorflow.keras.models import Model
intermediate_layer_model = Model(inputs=model.input,
(outputs=model.get_layer('intermediate_layer').output
intermediate_output = intermediate_layer_model.predict(-
(data
```

4.7 Automated Responses

Kubernetes: Orchestrates automated responses.

```
apiVersion: v1
kind: Pod
:metadata
name: ecommerce-api-restart
:spec
:containers
name: api -
image: ecommerce-api:latest
:lifecycle
:preStop
:exec
["command: ["/bin/sh", "-c", "sleep 10
```

Terraform: Manages infrastructure as code.

```
} "resource "kubernetes_deployment" "ecommerce_api
} metadata
"name = "ecommerce-api
{
} spec
replicas = 3
} template
} metadata
} = labels
"app = "ecommerce-api
{
{
} spec
} container
"image = "ecommerce-api:latest
"name  = "ecommerce-api
} port
container_port = 80
{
{
{
{
{
{
```

Ansible: Automates configuration management and deployment tasks.

```
name: Restart ecommerce API service -
hosts: all
:tasks
name: Restart Kubernetes pod -
command: kubectl rollout restart deployment ecom-
merce-api
```

4.8 Continuous Learning

TensorFlow: Continuously updates models with new data.

```
model.fit(new_data, new_labels, epochs=10, initial_ep-
(och=model.epochs
```

Apache Kafka: Streams new data for real-time model updates.

```
:kafka
bootstrap_servers: kafka.example.com:9092
topic: new_metrics
```

Kubernetes: Deploys updated models.

```
apiVersion: apps/v1
kind: Deployment
:metadata
name: ecommerce-model
:spec
replicas: 1
:template
:metadata
:labels
app: ecommerce-model
:spec
:containers
name: model -
image: ecommerce-model:latest
```

5. Results and Discussion

Table 1: Collected Metrics shows the raw data collected by Zabbix and Fluentd, including CPU usage, memory usage, network traffic, user requests, thread waiting times, and thread processing times.

Timestamp	CPU Usage (%)	Memory Usage (MB)	Network Traffic (MB)	User Requests	Thread Waiting Time (ms)	Thread Processing Time (ms)
7/4/22 10:00	45	1024	500	2000	50	100
7/4/22 10:05	50	1100	520	2100	55	110
7/4/22 10:10	60	1150	540	2200	60	120
7/4/22 10:15	70	1250	560	2300	65	130
7/4/22 10:20	80	1350	580	2400	70	140

Table 2: Feature Engineering illustrates how raw metrics are transformed into features that can be used for machine learning model training.

Timestamp	CPU to Memory Ratio	Requests per Thread	Process Time per Request (ms)
7/4/22 10:00	0.044	40	0.05
7/4/22 10:05	0.045	38	0.052
7/4/22 10:10	0.052	37	0.054
7/4/22 10:15	0.056	35	0.057
7/4/22 10:20	0.059	34	0.058

Table 3: Model Predictions presents the predictions made by the trained models, including predicted CPU usage, memory usage, and user requests, along with the corresponding scaling actions (e.g., scaling up or down the number of pods).

Timestamp	Predicted CPU Usage (%)	**Predicted Memory Usage (MB)**	**Predicted User Requests**	**Scaling Action**
7/4/22 10:25	85	1400	2500	Scale Up to 5 Pods
7/4/22 10:30	90	1450	2600	Scale Up to 7 Pods
7/4/22 10:35	88	1420	2550	Maintain 7 Pods
7/4/22 10:40	75	1300	2400	Scale Down to 6 Pods
7/4/22 10:45	70	1250	2300	Scale Down to 5 Pods

Table 4: Anomaly Detection captures the detection of anomalies in the system metrics, root cause analysis results, and the corrective actions taken to resolve the issues.

Timestamp	Metric	Observed Value	Anomaly Detected	Root Cause Analysis	Corrective Action
7/4/22 10:30	Thread Waiting Time	100 ms	Yes	High load on API Service	Restart API Service
7/4/22 10:35	Memory Usage	1600 MB	Yes	Memory leak detected in Product Service	Adjust Memory Allocation
7/4/22 10:40	CPU Usage	95%	Yes	High CPU usage due to inefficient query	Optimize Database Queries
7/4/22 10:45	User Requests	3000	No	-	-
7/4/22 10:50	Network Traffic	700 MB	Yes	Unusual traffic spike indicating potential DDOS	Deploy Additional Instances

Implementing predictive and corrective machine learning for the e-commerce application significantly improved resource utilization, scalability, and overall performance. The system collected comprehensive metrics such as CPU usage, memory consumption, network traffic, user requests, thread waiting times, and thread processing times using Zabbix and Fluentd. These metrics were processed using Apache Spark for feature engineering, resulting in feature sets that included historical usage patterns, workload characteristics, and application performance metrics.

The predictive models, trained with TensorFlow, used these features to forecast future resource demands. The models continuously received real-time metrics via Apache Kafka, allowing them to update predictions dynamically. When the predicted CPU utilization exceeded 70%, or memory consumption was projected to surpass 75%, the system triggered the Horizontal Pod Autoscaler (HPA) in Kubernetes. These thresholds were determined based on historical data and model training, ensuring proactive scaling before performance degradation occurred.

During a simulated peak load event, the predictive model anticipated a spike in user requests and increased CPU usage. As a result, the HPA scaled the number of API service pods from 3 to 7, ensuring sufficient capacity to handle the increased load. This scaling action occurred seamlessly, maintaining application performance without any downtime.

TensorFlow models identified deviations in system metrics for anomaly detection that indicated potential issues. For instance, an unusual increase in thread waiting times was detected, suggesting a potential bottleneck. The anomaly detection triggered root cause analysis using Neo4j, pinpointing a specific microservice experiencing a high load. Automated responses orchestrated by Kubernetes and managed through Terraform and Ansible restarted the affected microservice and adjusted its resource allocation. Continuous learning was achieved by streaming new data into the models via Apache Kafka, allowing them to adapt to changing usage patterns and improve prediction accuracy over time. As a result, the system refined its thresholds and scaling actions based on real-time feedback, enhancing its responsiveness and reliability.

The implementation ensured that the e-commerce application could handle varying loads efficiently. The predictive and corrective machine learning models provided accurate scaling and automated deployment responses, maintaining optimal performance and resource utilization. By dynamically adjusting to workload demands, the system achieved zero downtime, improved user experience, and reduced operational costs. This practical implementation demonstrates the effectiveness of integrating machine learning with modern orchestration tools to enhance the performance and scalability of cloud-native applications.

6. Conclusion

This experimental study has shown that predictive and corrective machine learning-based auto-scaling and auto-deployment in microservice-based e-commerce applications remarkably enhance resource utilization, scalability, and overall performance. The proposed approach combines state-of-the-art machine learning algorithms with Kubernetes and an orchestration framework to efficiently predict resources and respond against anomalies throughout, ensuring zero downtime operations. The practical approach described in this white paper very clearly illustrates how predictive models can be best utilized to scale and correctives for maintaining application health to ensure optimum resource allocation, improved user experience, and reduced operational costs. It would readjust scaling and deployment strategies developed further for incorporating more sophisticated machine learning models, such as reinforcement learning, into the predictive and corrective machine learning system to enhance its capabilities and robustness; furthermore, ensemble learning techniques are likely to improve the accuracy and reliability of predictions and anomaly detections.

One can integrate more diverse data sources, including user behavior analytics, third-party APIs, and real-time market data, to give a more complete view of system performance and external factors influencing resource demand. Feature engineering tools and techniques can be adopted that will more appropriately identify and extract relevant features from the raw data to improve the predictive power of the models. This would be important in ensuring that the data and models are well protected against any possible breach and adherence to the regulations on data privacy. Anomaly detection systems integrated with SIEM tools can automate security responses and mitigate risks.

This can further improve with the adoption of serverless computing and edge computing paradigms, which would give better scalability and lower latency in applications with a geographically spread-out user base. Dynamic scaling and resource allocation would work in conjunction with continuous monitoring and optimization of the underpinning infrastructure. More granular, real-time monitoring and feedback loops would allow adjustments to be made immediately, based on real-time data, to keep the system adaptive and responsive. Performance would be clearly indicated through advanced visualization tools concerning system performance and predictive model accuracy.

7. References

1. Reiss C, Tumanov A, Ganger GR, Katz RH, Kozuch MA. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. Proceedings of the Third ACM Symposium on Cloud Computing 2012.
2. Ding Y, Qin X, Liu L, Wang T. Energy-efficient scheduling of virtual machines in cloud with deadline constraint. Future Generation Computer Systems 2013;29: 2041-2054.
3. Ballani H, Costa P, Karagiannis T, Rowstron A. Towards predictable datacenter networks. Proceedings of the ACM SIGCOMM Conference 2013.
4. Aceto G, Botta A, de Donato W, Pescapè A. Cloud monitoring: A survey. Computer Networks 2013;57: 2093-2115.
5. Zaharia M, Xin RS, Wendell P, et al. Apache Spark: A unified engine for big data processing. Communications of the ACM 2016;59: 56-65.
6. Delimitrou C, Kozyrakis C. Paragon: QoS-aware scheduling for heterogeneous datacenters. Proceedings of the eighteenth international conference on architectural support for programming languages and operating systems 2013.
7. Di S, Kondo D, Cirne W. Characterization and comparison of cloud versus grid workloads. 2012 IEEE International Conference on Cluster Computing 2012; 230-238.
8. Marinos A, Briscoe G. Community cloud computing. 2012 IEEE International Conference on Cloud Computing Technology and Science 2012; 133-142.

9. Columbus L. Roundup of machine learning forecasts and market estimates, 2020. Forbes 2020.
10. Dean J, Ghemawat S. MapReduce: Simplified data processing on large clusters. Communications of the ACM 2008;51: 107-113.
11. Warneke DT, Kao O. Exploiting dynamic resource allocation for efficient parallel data processing in the cloud. IEEE Transactions on Parallel and Distributed Systems 2011;22: 985-997.
12. Dastjerdi AV, Buyya R. An autonomous agent-based engine for cloud resource provisioning. Int J Computational Intelligence and Applications 2011;10: 151-172.