Predictive Auto-Scaling for Cloud Applications

Machine Learning Driven Framework using E-Commerce Data Workloads

Chaithra Jagannatha Rao Telkar | MSc In Applied Computer Science | SRH University Heidelberg

Introduction

This project focuses on improving how cloud systems manage their computing resources automatically. It aims to make cloud applications more efficient, reliable, and responsive to changing user demands. Instead of reacting after performance issues occur, the system predicts changes in workload in advance. By doing so, it ensures that enough resources are always available without wasting capacity. The approach promotes smarter decision-making and stable application performance. Continuous monitoring helps maintain transparency and trust in system behavior. Overall, the project highlights how predictive intelligence can make cloud environments more adaptive. It contributes to building faster, cost-effective, and sustainable cloud operations.

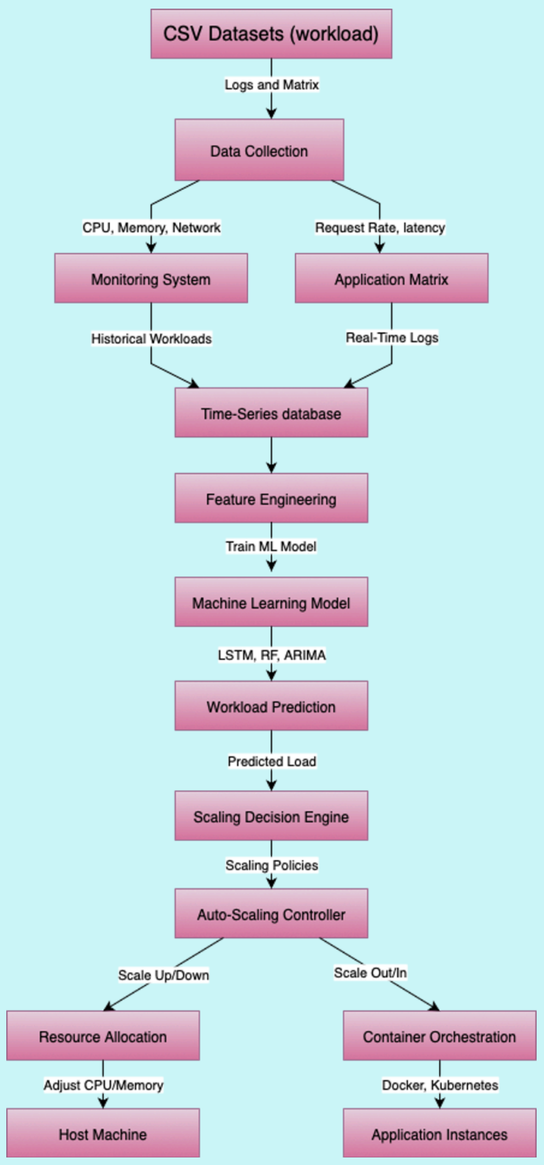
Research Objective

Traditional auto-scaling methods rely on reacting to predefined metric thresholds, which often results in delayed or unstable scaling decisions. These reactive approaches can cause latency violations, inefficient resource usage, and service degradation when workloads change rapidly. The main limitation lies in the absence of predictive intelligence that can anticipate workload fluctuations before they occur. Existing systems typically lack the integration of machine learning based forecasting with real-time scaling control. Therefore, there is a strong need for a predictive, uncertainty aware auto-scaling framework that ensures stability, efficiency, and adaptability in modern cloud environments.

Technology Used

- **Programming Language:** Python 3, Yaml, Javascript
- **Machine Learning Libraries:** TensorFlow / Keras (LSTM), Scikit-learn (Random Forest), Statsmodels (ARIMA)
- **Data Handling:** Pandas, NumPy, Joblib
- **Monitoring & Visualization:** Prometheus (metrics collection), Grafana (dashboard visualization)
- **Containerization:** Docker & Docker Compose (to deploy controller, Prometheus, Grafana, and exporters)
- **Logging & Observability:** Structured logs via controller service; Prometheus metrics endpoint
- **Framework Integration:** Custom Python-based Predictive Scaling Controller connected with ML pipeline

System Architecture



The above architecture illustrates the predictive auto-scaling workflow in cloud environments. It begins with data collection from workloads, monitoring systems, and application metrics, which are stored in a time-series database for feature engineering and machine learning–based workload prediction (using LSTM, RF, ARIMA). The predicted load guides the auto-scaling controller to adjust resources or orchestrate containers dynamically for efficient performance.

Future Work

- Future work includes integrating the framework with Kubernetes using containerization, enabling seamless deployment and intelligent scaling through HPA/VPA integration.
- Additionally, optimization will be extended beyond p95 latency to include cost, error rate, and CPU utilization, ensuring more comprehensive and efficient resource management.

Results & Evaluation

```
Data processing completed ✅
Feature engineering started ...
[features] X shape=(2796, 22), y len=2796
Feature engineering completed ✅
Model training started ...
[LSTM] saved: artifacts/lstm_model.keras
[RF] saved: artifacts/rf_model.pkl
[ARIMA] saved: artifacts/arima_model.pkl (order=(2, 1, 2))
Model training completed ✅
Prediction started ...
[predict:LSTM]  next=93.2476
[predict:LSTM-MC] mean=92.7540 sigma=1.8729
[predict:RF]    p50=93.0661  p90=93.0703  p95=93.0703  (point=93.0624)
[predict:ARIMA] next=93.0420
📝 saved predictive signal -> artifacts/predictive_signal.json
Prediction completed ✅
Evaluation started ...
[LSTM] MAE=1.6196  RMSE=2.1966  N=2776
[RF] MAE=0.0517  RMSE=0.1424  N=2796
[ARIMA] MAE=2.5326  RMSE=3.2867  N=2795
Evaluation completed ✅
```

Fig 1-Execution of Predictive Signal Generation Pipeline

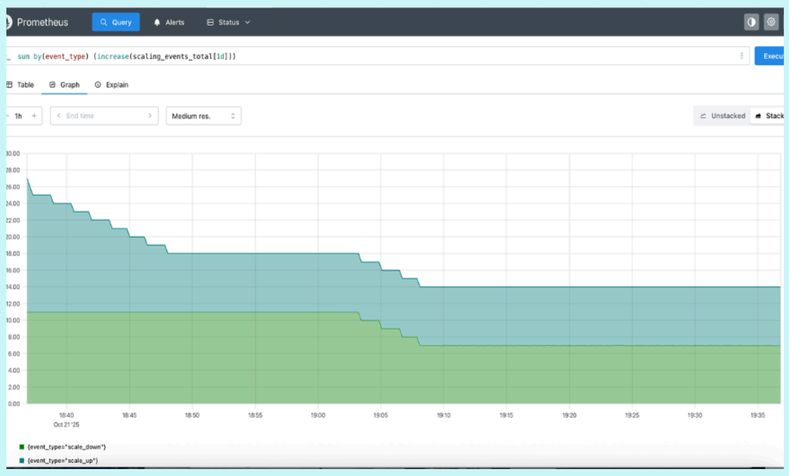


Fig 2-Total Scaling Events (Up/Down)

```
info: predictive read: metric=response_time_p95 mean=0.0928 sigma=0.0019 alpha=1 decision
=0.0946 threshold(SLO p95 (seconds))=0.09 {"service":"auto-scaling-controller","timestamp
":"2025-10-22T22:06:00.478Z"}
info: Initiating scaling to 3 replicas. Reason: predictive_response_time_p95: value=0.094
6 threshold=0.09 (mean=0.0928, σ=0.0019, α=1) {"service":"auto-scaling-controller","times
tamp":"2025-10-22T22:06:00.479Z"}
info: Scaling completed: 2 -> 3 replicas in 2.001s {"service":"auto-scaling-controller","
timestamp":"2025-10-22T22:06:02.482Z"}
```

Fig 3-Controller Log Showing Predictive Scaling Decision and Execution

```
# HELP scaling_decisions_total Total number of scaling decisions made
# TYPE scaling_decisions_total counter
scaling_decisions_total{decision="scale",reason="predictive_response_time_p95: value=0.0941 threshold=0.08 (mean=0.0928, σ=0.0014, α=1)"} 19
scaling_decisions_total{decision="scale",reason="manual_test"} 2

# HELP scaling_events_total Total number of scaling events
# TYPE scaling_events_total counter
scaling_events_total{event_type="scale_up",target="app"} 19
scaling_events_total{event_type="scale_down",target="app"} 2

# HELP current_replicas Current number of replicas
# TYPE current_replicas gauge
current_replicas 10

# HELP target_replicas Target number of replicas
# TYPE target_replicas gauge
target_replicas 10

# HELP scaling_latency_seconds Time taken to complete scaling operations
# TYPE scaling_latency_seconds histogram
scaling_latency_seconds_bucket{le="0.1"} 0
scaling_latency_seconds_bucket{le="0.5"} 0
scaling_latency_seconds_bucket{le="1"} 0
scaling_latency_seconds_bucket{le="2"} 3
scaling_latency_seconds_bucket{le="5"} 21
scaling_latency_seconds_bucket{le="10"} 21
scaling_latency_seconds_bucket{le="+Inf"} 21
scaling_latency_seconds_sum 42.02199999999999
scaling_latency_seconds_count 21

# HELP predictive_mean Predictive mean
# TYPE predictive_mean gauge
predictive_mean 0.09275403594970703

# HELP predictive_sigma Predictive sigma
# TYPE predictive_sigma gauge
predictive_sigma 0.0018729168176651002

# HELP predictive_upper Predictive upper bound
# TYPE predictive_upper gauge
predictive_upper 0.09462695276737214

# HELP predictive_threshold Predictive threshold
# TYPE predictive_threshold gauge
predictive_threshold 0.08

# HELP predictive_wants_scale Predictive scale desire 0/1
# TYPE predictive_wants_scale gauge
predictive_wants_scale 1
```

Fig 4 Prometheus-exposed controller metrics showing predictive, decision, and scaling-related telemetry data

Figure 1: Shows the machine learning pipeline execution, where models (LSTM, RF, ARIMA) predict the target metric response_time_p95. The outputs — mean latency and predictive uncertainty (σ) — are stored and later used in the decision rule **decision=mean+α · σ**

Figure 2: Displays the Prometheus monitoring graph, visualizing the number of scale-up and scale-down operations over time. These scaling actions are triggered when the computed decision value exceeds the latency SLO threshold (SLO_{p95}).

Figure 3: Shows the auto-scaling controller logs, where the controller reads the predicted values (mean = 0.0928 s, σ = 0.0019 s, α = 1.0) and applies the decision rule 0.0928+(1×0.0019)=0.09460.0928+(1×0.0019)=0.0946. Since this is greater than the SLO (0.09 s), the controller initiates a scale-up from 2 to 3 replicas.

Figure 4: Presents the Prometheus metrics exposition, reporting internal variables and counters from the auto-scaling controller. It includes metrics such as:

- predictive_mean, predictive_sigma, and predictive_upper — corresponding directly to the formula components (mean, σ, mean + ασ).
- predictive_threshold = 0.08 (SLO), and predictive_wants_scale = 1, indicating that scaling was triggered.
- scaling_decisions_total and scaling_latency_seconds show how many scaling actions occurred and how long they took to execute.

References

- Prometheus — https://prometheus.io/docs/introduction/overview/
- TensorFlow — https://www.tensorflow.org/
- statsmodels (ARIMA) — https://www.statsmodels.org/
- Kubernetes HPA — https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/
- https://cloud.google.com/stackdriver/docs/solutions/slo-monitoring/concepts