CP395 Weekly Report 3

Sufiya Rahemtulla – 169018559
January 26th, 2026

Week 3 Objectives and Completion Status

- Implemented a reproducible data ingestion pipeline for Google Cluster Trace v3
- Performed Exploratory Data Analysis (EDA) to characterize workload volatility
- Defined input features and target labels based on evidence

Data Preparation Steps and Design Decisions

To address the challenge of data scale (500GB+ trace files), I implemented a streaming ingestion pipeline using Python's gzip and json libraries. Some key decisions:

- **Dataset Selection:** I utilized the instance_usage table from Cell A of the Google Cluster Trace v3. I filtered for a single dominant job_id to isolate a specific microservice workload rather than aggregating unrelated jobs.
- **Granularity (5-Minute Windows):** The raw trace data in microseconds was resampled into 5-minute intervals. (This aligns with the default scrape interval of standard cloud monitoring tools and the typical reaction window of the Kubernetes Horizontal Pod Autoscaler (HPA))
- **Normalization:** Time was normalized from Unix epoch (microseconds) to standard Datetime objects. CPU usage was also normalized to NCUs (Normalized Compute Units) to maintain consistency.

Key EDA Findings (Workload Characterization):

1. **High Volatility and "Reaction Gaps":**

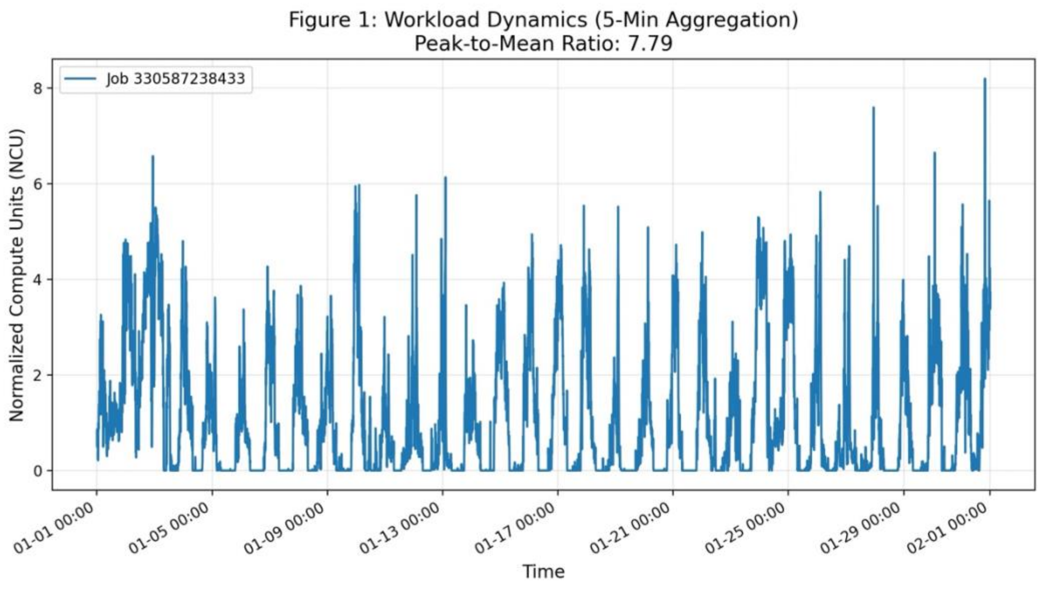

Figure 1: Time-series analysis of aggregated CPU demand (5-minute intervals)

As shown in *Figure 1*, the CPU usage does not follow a smooth pattern; instead, it exhibits sharp, jagged spikes where demand can double or triple within a single 5-minute window. This evidence supports the "Reaction Gap" problem I identified in my literature review. A standard reactive scaler (like the Kubernetes HPA) waits for averages to cross a threshold. Given how fast these spikes occur, a reactive scaler would likely provision resources too late, causing performance issues during the ramp-up of these bursts.

2. Heavy-Tailed Distribution

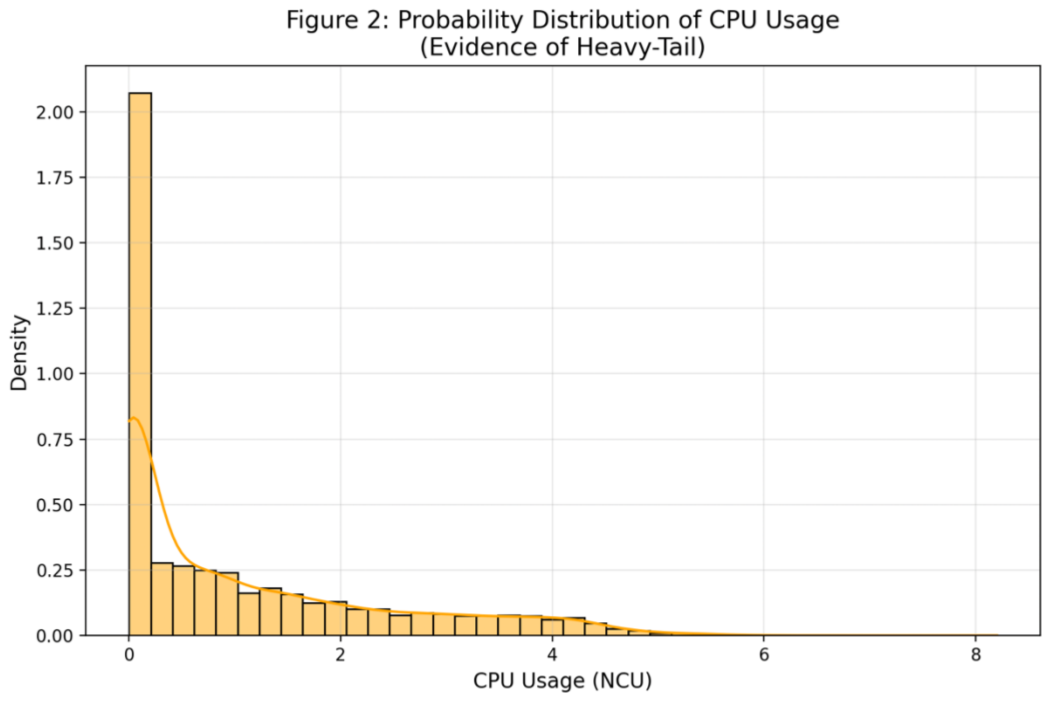

Figure 2: Probability density of CPU usage showing a right-skewed tail

Figure 2 shows that the data follows a "Heavy-Tailed" or Right-Skewed distribution. The tall bar on the left indicates that for the majority of the time, the CPU usage is relatively low and stable. However, the long tail extending to the right shows that extreme high-load events, while rare, do occur.

This confirms that optimizing an autoscaler for the "average" case (e.g., minimizing Mean Squared Error) is dangerous. An autoscaler that fits the average would under-provision during these critical tail events. This validates the need for my proposed *Failover Mechanism*, which is designed specifically to handle these unpredictable outliers that standard models might miss.

Feature and Label Definition:

While predicting the *mean* CPU usage would minimize resource waste (Slack), it ignores the intra-window volatility observed in the EDA. Because the distribution is heavy-tailed (Figure 2), a model that predicts the mean will systematically under-provision during peak bursts, leading to SLO violations.

Since the primary objective of this research is Reliability-Aware scaling, I have chosen to target the Next-Window Maximum. I explicitly accept a higher cost (slack) in exchange for minimizing the risk of under-provisioning during 'Black Swan' events.

Input Features:

- **Lagged Demand (t-1, t-2, t-3):** Captures immediate history. *Figure 1* shows continuity between adjacent time steps, making autoregressive features highly predictive.
- **Rolling Volatility (StdDev 30m):** To address the "Reaction Gap," this feature helps the model distinguish between "steady state" traffic and the onset of "burst" phases seen in the EDA.
- **Seasonality (Hour of Day):** To capture the macro diurnal (day/night) cycles observed in the trace, allowing the model to anticipate expected daily peaks.

Target Label (Next-Window Maximum):

- The model will predict the ***maximum*** **CPU usage** expected in the upcoming 5-minute window.
- *Figure 2* shows a heavy-tailed distribution. Predicting the mean load would statistically **under-provision** during the critical tail events.
- Predicting the *peak* is a safety-first approach required to minimize the SLO violations identified in the problem statement.

Assumptions:

1. **CPU Bottleneck:** I assume that CPU usage is the primary performance constraint. While Memory or I/O could be bottlenecks, the trace data is richest in CPU metrics, and CPU is the standard metric for the Kubernetes HPA baseline.
2. **Stationarity within Windows:** While the workload is volatile, I assume that within short (5-minute) windows, the statistical properties remain stable enough for short-term forecasting models (like ARIMA) to function.

Risks:

1. **Cost Metric Obfuscation:** As noted in previous feedback, exact dollar costs are not available. I will define a "Unit of Waste" metric *(Allocated - Used)* to quantify efficiency without needing currency.
2. **Baseline Selection:** There is a risk that a weak baseline will make my proposed model look artificially good. I will implement the industry-standard Kubernetes HPA logic (reactive threshold scaling) as the primary policy baseline, and a standard time-series model (e.g., ARIMA) as the prediction baseline.

Next Weeks Plan:

- **Baselines:** Implement ARIMA (statistical prediction baseline) and a Reactive Threshold simulation (Scale up >80%, Down <50%) to establish the benchmark for SLO violations.
- **Model Selection:** Utilize PyCaret to rapidly screen and select the best regression algorithms (e.g., Random Forest vs. Gradient Boosting) for the proactive scaling component.