CSC4160: Cloud Computing - Final Project (Midterm Report)

Title: Analyzing public cloud traces using machine learning to improve resource allocation

Team member:
Chua Qin Di, 125400010, 125400010@link.cuhk.edu.cn

1. Introduction

Modern large-scale cloud providers such as Google Cloud, AWS, and Microsoft Azure execute millions of containerized workloads daily on shared clusters. Each submitted job requires users to specify its CPU and memory requirements before execution. However, these user-provided resource requests are often inaccurate and highly conservative. Users tend to over-request resources to be safe, resulting in underutilization and inflated costs. At the same time, aggressive under-requesting of resources would risk out-of-memory (OOM) failures, job evictions, and service-level agreement (SLA) violations.

This project uses the Google Cluster Trace 2019 dataset to analyze how requested resources compare to actual CPU and memory usage and to explore whether machine learning can predict real resource demand more accurately than user guesses.

The midterm report focuses on the following components:

1. **Data Processing Pipeline** - preprocessing of data
2. **Exploratory Data Analysis (EDA)** - characterizing cloud usage patterns, workload, and inefficiencies.
3. **Preliminary Experiments and Results** - benchmarking of baseline predictive models.
4. **Key Findings** - highlighting the potential of the models and the limitations of available features.
5. **Roadmap for the Final Report** - outlining improvements for model and integration of scheduler-compatible constraints as well as issues to be addressed in the final report.

Additional required files may be found at the GitHub repository of the project:
https://github.com/chuaqindi/cloud-computing-project/tree/main

2. Dataset and Data Extraction

2.1 Google Cluster Trace 2019

The Google Cluster Trace 2019 dataset is a large, publicly released trace collected from Google's production data centers. It records how containerized workloads are scheduled, executed, and monitored across thousands of machines in a real cluster environment. At a high level, the **trace** captures:

- **CPU and memory usage** sampled at regular intervals for each container instance.
- **User-requested** CPU and memory for each job.
- **Instance-level** events - start, finish, fail, and evict.
- **Collection-level** metadata - priority, scheduling class, and collection type.
- **Machine identifiers** and scheduling-related attributes.

This dataset reflects real production workloads rather than simulated data, which is ideal for studying workload behavior, inefficiencies in user requests, and the potential for prediction-based resource management.

2.2 Use of Google BigQuery

The raw trace is stored in multiple Google BigQuery tables and is far too large to download and process directly on a local machine. Google BigQuery is a serverless, petabyte-scale SQL analytics engine that allows queries over these tables without materializing the full data locally. In this project, BigQuery is used to select, join, and aggregate the relevant tables:

- **instance_usage**: runtime CPU and memory usage samples.
- **instance_events**: instance lifecycle events (start, stop, fail, evict, etc.).
- **collection_events**: job-level metadata such as scheduling class, priority, and user.

The final dataset (sample of 90,000 rows to reduce dataset size) contains the pre-execution metadata and post-execution usage metrics suitable for exploratory analysis and machine learning.

Nested JSON structures are also flattened and converted into separate columns. The SQL query to generate the dataset is documented in a separate file. (BigQuerySQL.txt)

3. Exploratory Data Analysis

Exploratory data analysis was conducted to characterize workload behavior and to motivate the choice of predictive targets and features.

3.1 Preparation of Analysis Dataset

During EDA, the instance-level and collection-level event JSON fields were parsed to extract key attributes such as requested CPU, requested memory, scheduling class, collection type, and priority. Incomplete rows were removed during preprocessing, and numeric fields were standardized to ensure consistency. Simple derived features such as instance duration,over-request ratios, cpu-to-memory ratio were also added to capture basic workload characteristics.

3.2 CPU Usage Patterns

CPU usage across the cluster is extremely low for the vast majority of container instances. The distribution of average CPU usage per job shows that most jobs consume **less than 1% CPU on average**, and over 90% of all usage samples fall below **2–3% utilization**. This indicates heavy over-provisioning and a workload mix dominated by lightweight or I/O-bound tasks.

The percentile curves highlight this skew: the median CPU usage is close to **0%**, the 90th percentile reaches only **≈2.35%**, and even the 99th percentile remains under **5%**. Only a small number of outlier samples exhibit higher utilization, reflecting short, bursty activity in a minority of workloads.

The combination of extremely low steady-state CPU usage and occasional tail spikes illustrates why static CPU requests tend to be highly conservative. Most workloads rarely approach their requested CPU limits, reinforcing the opportunity for CPU overcommitment or prediction-assisted right-sizing.

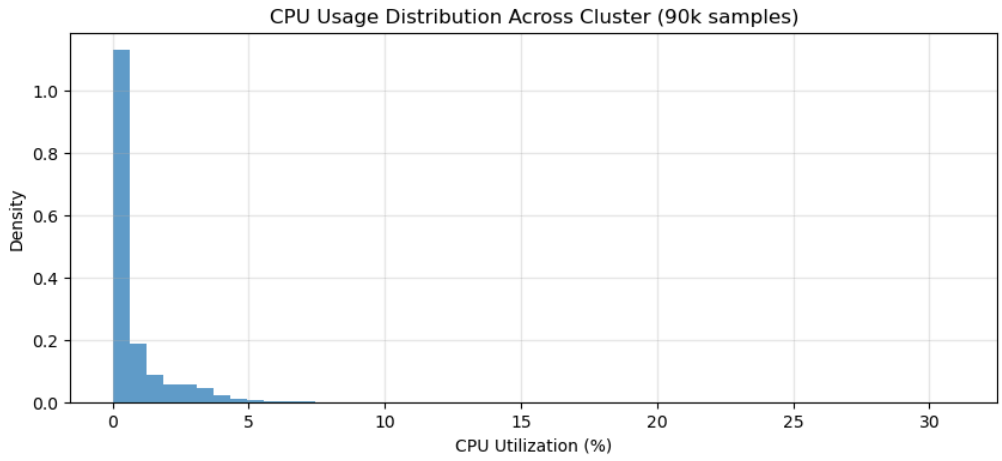

Figure 1: CPU Usage Distribution

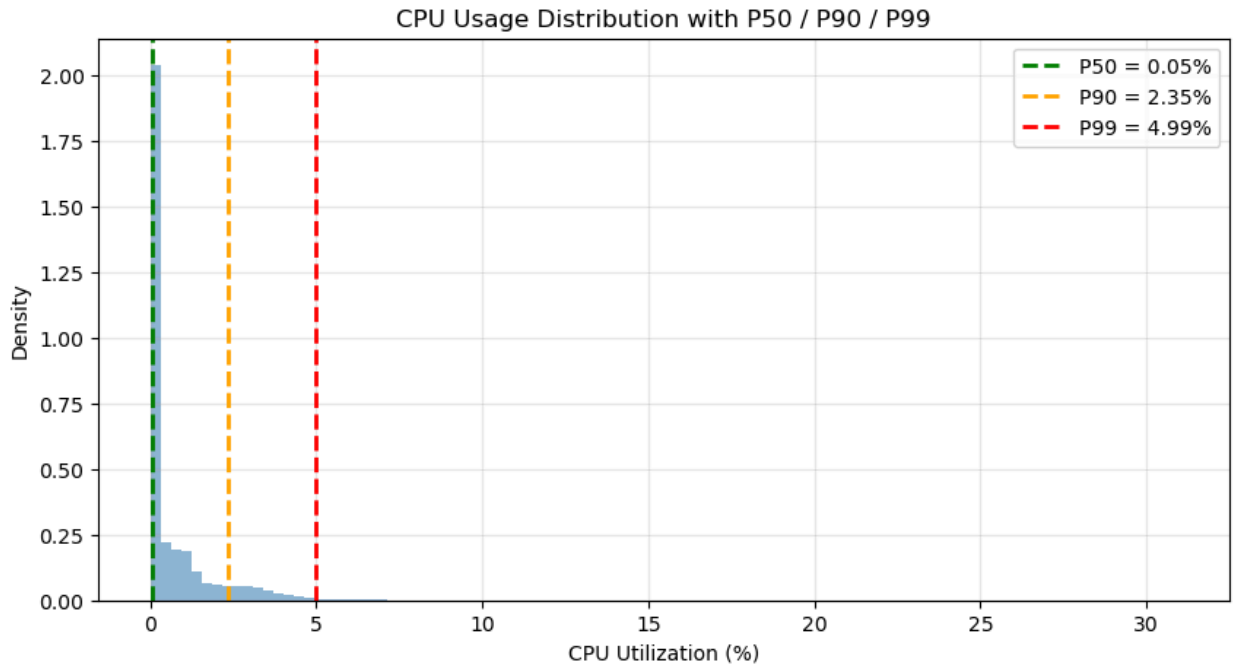

Figure 2: Percentile Distribution

3.3 Memory Usage Patterns

Memory usage demonstrates greater stability compared to CPU usage. Distributions of average and peak memory indicate that most workloads remain within a narrow low-utilization range, with slight variation. The peak-to-average ratio frequently remains below 2x, reflecting the relative absence of transient spikes in memory consumption.

Despite this, requested memory substantially exceeds observed peak usage for most workloads, at times by orders of magnitude. This systematic over-requesting suggests that memory allocations contain significant slack. Given the relatively predictable nature of memory usage, prediction strategies could reduce wastage without significantly increasing the risk of OOM failures, particularly for non-latency-critical jobs.

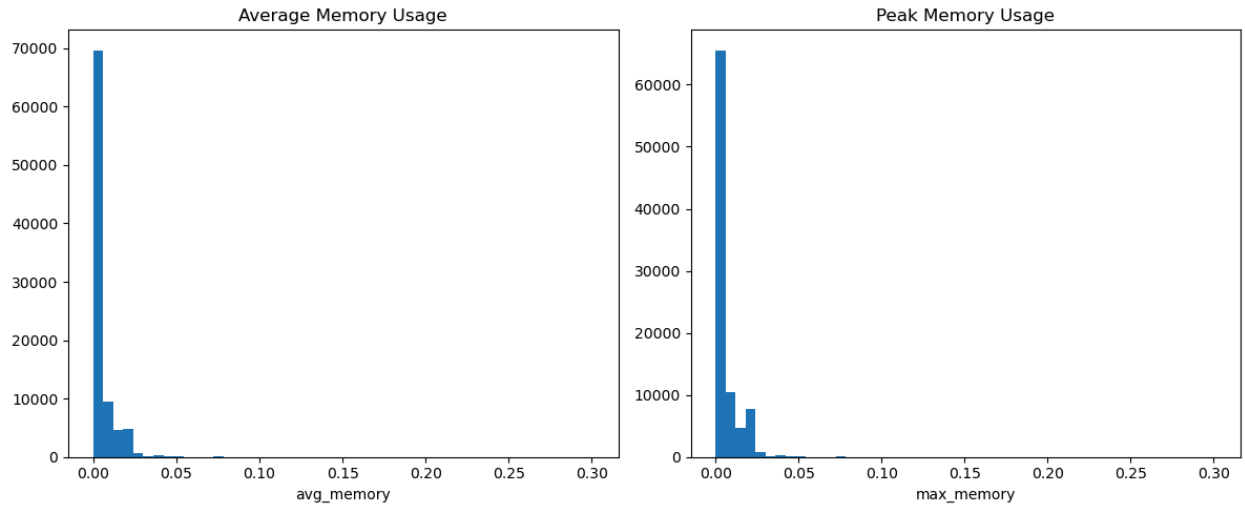

Figure 3: Memory Usage

3.4 Job Failure

Failure and eviction events show clear relationships with both requested memory and scheduling class. Very small memory requests have the highest failure rates, likely due to under-provisioning, while larger memory requests also show increased failures—reflecting the greater runtime and complexity of heavier jobs. Mid-range requests are the most reliable.

Scheduling class further influences reliability: low-priority jobs experience substantially higher failure and eviction rates due to preemption, whereas high-priority tasks are far more stable.

Overall, reliability varies systematically with **resource request size** and **scheduling priority**, rather than occurring randomly.

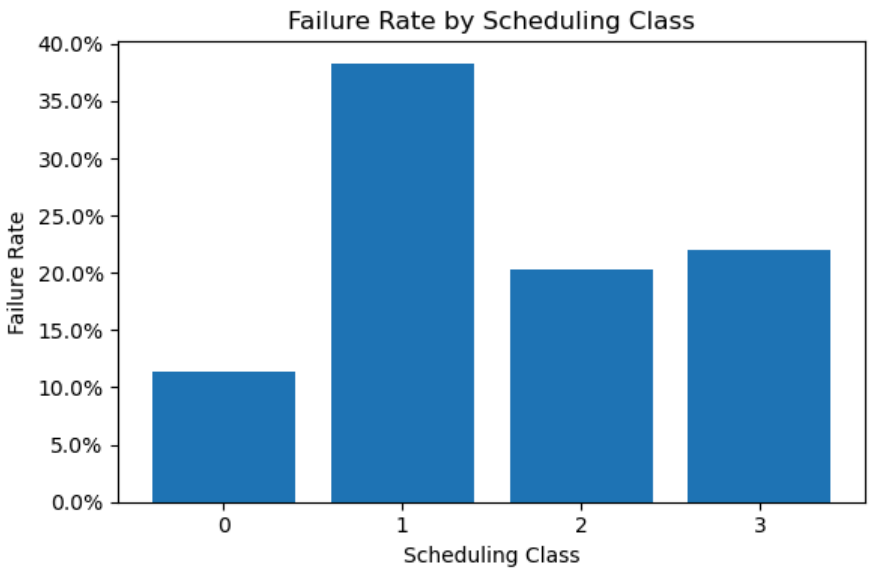

Figure 4: Failure rates when grouped by scheduling class

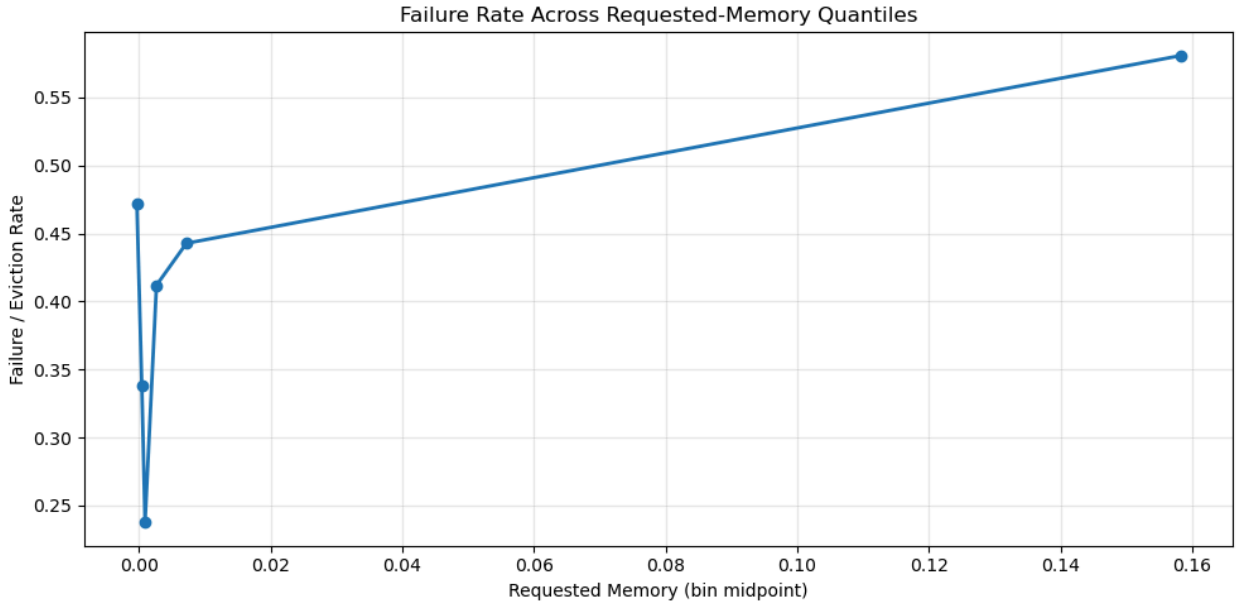

Figure 5: Failure rates as requested memory increased

3.5 Summary of EDA

The exploratory analysis reveals pervasive resource underutilization and clear structure in workload behavior. CPU usage across the cluster is extremely skewed: most workloads consume well under 2-3% CPU on average, with only occasional short-lived bursts reaching higher utilization. In contrast, memory usage is far more stable, but user-requested memory routinely exceeds observed peak usage, often by multiples, indicating systematic over-provisioning. Reliability patterns also display structured trends. Failure and eviction rates increase for both very small and very large memory requests, and lower scheduling-class workloads experience significantly higher disruption due to preemptive policies.

However, these observations are based on a **90,000-row sample**, which represents only a small slice of the full 2019 Google trace. This limited sample constrains the diversity of workload types, reduces the frequency of rare events such as severe CPU spikes or large-scale failures, and may bias the distribution toward lighter workloads depending on the sampled time window and cluster subset. As a result, the current analysis captures broad patterns but cannot yet fully characterize tail behaviors or workload heterogeneity. The final report could address these limitations by incorporating larger samples from BigQuery.

4. Predictive Modeling

The predictive modeling experiments evaluate how effectively machine learning can estimate true resource usage relative to user-provided requests. Several target variables were considered - including average CPU, average memory, and peak memory. Models were evaluated under two settings: an idealized model with **post-execution** metrics, and a **real-time scheduling** scenario limited to submission-time metadata.

4.1 Theoretical Model

In this idealised theoretical setting, the models were trained with scheduler metadata and runtime-derived features such as average memory usage, peak-to-average ratios, and related performance indicators. Under these ideal conditions, tree-based models including LightGBM and XGBoost achieve exceptionally high accuracy, with **R^2 values exceeding 0.98** for peak memory prediction. This demonstrates that cloud workloads exhibit strongly structured and predictable memory behavior when detailed runtime signals are available.

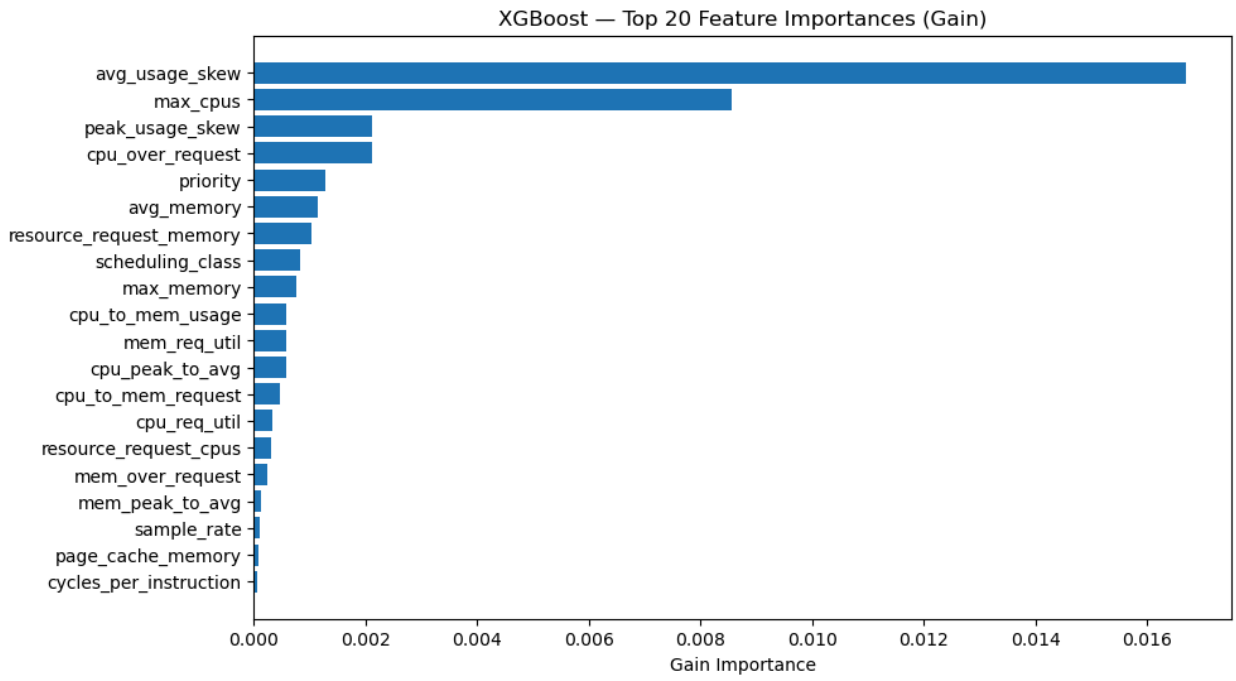

Figure 6: XGBoost Feature Importance

The feature-importance results show that the most predictive signals for CPU usage are **actual workload behaviors**, especially **avg_usage_skew**, **peak CPU**, and **memory usage levels**. This means that tasks with bursty or high peak usage tend to have higher average CPU demand. In contrast, **requested resources** (requested CPUs/memory, over-request ratios) contribute far less, confirming that user-provided requests are poor indicators of true usage.

4.2 Real-Time Scheduling Model

However, the features in **4.1** are only observable after execution and these results do not reflect what a production scheduler can leverage in real time, which is why data centers still have this issue of misallocating resources.

When restricted to **submission-time metadata alone**, predictive performance drops substantially. Using fields such as requested CPU and memory, scheduling class, priority, XGBoost achieves approximately $\mathbf{R^2 \approx 0.48}$ for peak memory prediction. Requested memory emerges as the dominant predictor, while other metadata contributes comparatively little. The large gap between the theoretical and real-time models highlights the severe information constraints faced by operational schedulers and reinforces that static user declarations capture workload behavior only loosely.

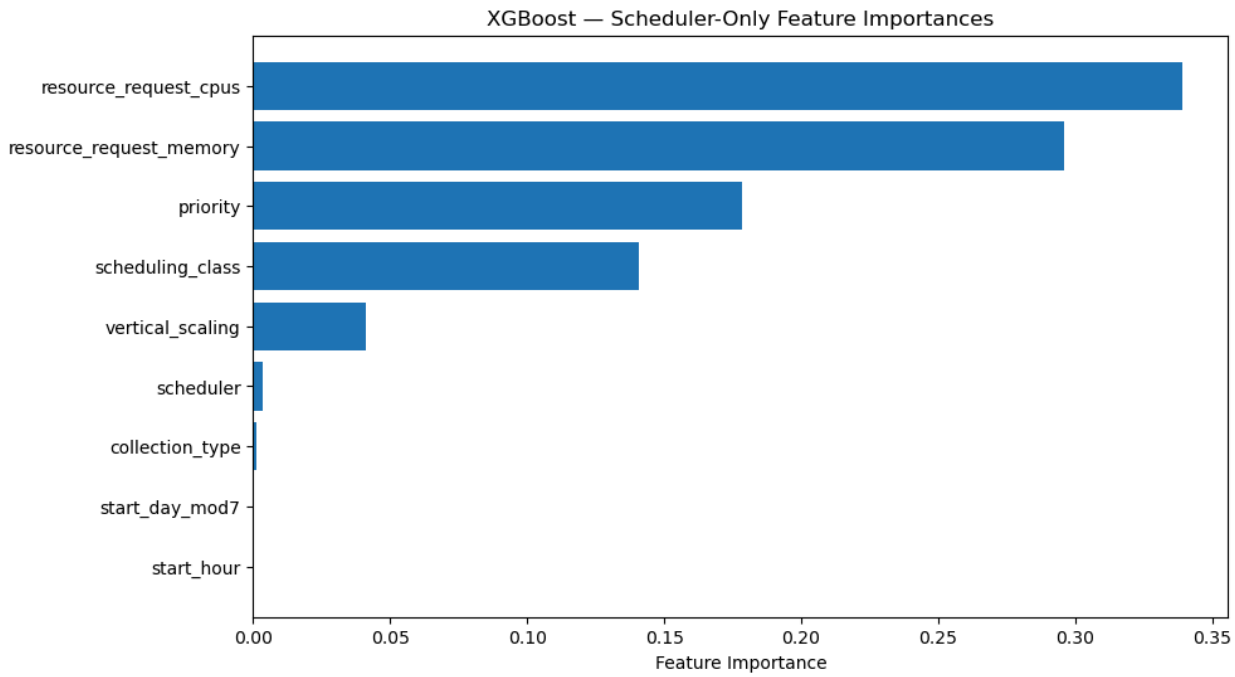

Figure 7: Feature Importance of real-time model

4.3 Key Findings

The contrast between the two modelling results illustrates a fundamental challenge: while workload resource usage is highly predictable when runtime information is known, it is significantly less predictable at submission time, where production schedulers must operate. These findings suggest that enhancing real-time prediction requires incorporating **historical workload behavior, user-level patterns, or job lineage information**, enabling the scheduler to approximate the benefits of post-execution features without violating real-time constraints. Suggestions for the next stage of the project could involve a history-aware, submission-time prediction framework.

5. Challenges and Feasibility

The midterm results highlight several key challenges in developing practical ML-assisted scheduling mechanisms for large-scale clusters.

Information Asymmetry.
A fundamental limitation arises from the gap between what can be learned offline from full execution traces and what a scheduler can observe at submission time. Models leveraging post-execution features achieve near-perfect accuracy but are not deployable, while real-time models operate with restricted visibility and consequently lower predictive power.

Workload Burstiness.
Both CPU and memory exhibit short, concentrated spikes that strongly influence peak usage metrics. These bursts are difficult to anticipate from static metadata alone, complicating accurate prediction of worst-case behavior, particularly for peak memory which governs OOM risk.

Over-Requesting and Safety Requirements.
Although workloads consistently over-request resources, schedulers must prioritize safety and avoid under-provisioning. Any prediction-based right-sizing must incorporate conservative safety margins, especially for latency-sensitive jobs where OOM failures are unacceptable.

Cold-Start Workloads.
Jobs with no historical record present an inherent challenge for history-informed models. Without prior behavioral data, predictions become less reliable, necessitating fallback strategies such as default allocations, user-specific priors, or class-based heuristics.

Despite these challenges, the analysis indicates that workloads follow predictable patterns when adequate information is available.

This suggests that fine-tuning models with only scheduler submission-time input variables could offer a feasible path toward practical, safe, and effective right-sizing strategies.

6. Roadmap for Final Report

The final report will focus on developing a more realistic prediction framework and translating model outputs into actionable right-sizing policies. The first improvement involves improving the real-time scheduling model with **historical signals**, such as prior peak memory usage, over-request ratios, and historical failure rates for similar workloads. These features are observable to a production scheduler and are expected to substantially enhance predictive accuracy relative to the submission-time baseline.

Although ML-assisted right-sizing shows strong potential in offline studies, major cloud schedulers such as Google Cloud and Azure have not widely adopted it due to reliability concerns. Underallocation of memory can result in cascading consequences which is costly for big companies and services.

Model outputs can be converted into practical allocation decisions by applying configurable **safety factors** (1.2×, 1.3× etc.). This will allow systematic evaluation of the trade-off between memory savings and the risk of under-provisioning, measured against ground-truth peak memory. Quantifying these trade-offs is essential for determining whether ML-assisted right-sizing can meaningfully reduce memory waste while maintaining the reliability guarantees required.

The final report will also examine key deployment considerations, including fallback strategies for cold-start workloads that lack historical data, the selection of conservative safety margins for latency-sensitive or mission-critical services, and the interaction of right-sizing recommendations with existing quota, priority, and preemption mechanisms used by modern schedulers.

7. Conclusion

This midterm report summarizes the initial progress of a project that applies machine learning to public cloud workload traces to improve resource allocation. The work established a data extraction pipeline, performed exploratory analysis of CPU and memory behavior, and developed baseline models for predicting resource usage.

The results show that workload behavior is highly predictable when post-execution information is available, but prediction becomes significantly harder when restricted to submission-time metadata - reflecting the real constraints faced by online schedulers.

The final report will focus on building such history-aware models, evaluating their impact on cluster efficiency, and proposing practical policies for deployable, ML-assisted resource allocation in large-scale cloud environments.