Wasted Cycles and Waiting Games: Analysis of HPC Resource Usage Using Production Cluster Data in FRESCO

Aryamaan Ujwal Dhomne*
Joshua McKerracher*
adhomne@purdue.edu
jmckerra@purdue.edu
Purdue University
West Lafayette, Indiana, USA

Saurabh Bagchi
Purdue University
West Lafayette, Indiana, USA
sbagchi@purdue.edu

Abstract

High-Performance Computing (HPC) systems suffer from two persistent inefficiencies: unpredictable queue wait times and resource underutilization. Using the FRESCO dataset—which links 20.9 million production job records with fine-grained performance telemetry across three high performance computing (HPC) clusters—we study (RQ1) what drives queue wait times and whether they can be predicted, and (RQ2) how prevalent and severe resource waste is across workloads. For RQ1, framing wait time as a four-way classification task yields a mean F1-score of 0.92 on a temporal holdout from a production system, with SHAP analyses highlighting submission-time load (pending jobs, free cores) and queue choice as dominant factors. For RQ2, system-level composite waste (equal-weight CPU and memory) averages 29–30% on Conte and Anvil. At the job level, however, waste is highly skewed: in a 10% random sample of jobs (the unit of analysis), 69.3% of jobs exceed 50% composite waste, driven primarily by memory underutilization; waste also increases with job size (from 38.9% for 1–16 nodes to 61.9% for >256 nodes). These results quantify the operational cost of inefficiency and show how linked accounting-and-telemetry data can support user-facing prediction tools and targeted administrative policies.

CCS Concepts

• **Computer systems organization** → **Dependable and fault-tolerant systems and networks**; **Reliability**; **Availability**.

Keywords

high-performance computing, resource utilization, queue wait time prediction, resource waste analysis, job scheduling, machine learning, FRESCO dataset, cluster efficiency, SHAP interpretability, production workload analysis

*Both authors contributed equally to this research.

CODS 2025, Pune, India

ACM ISBN 978-1-4503-XXXX-X/2018/06
https://doi.org/XXXXXXX.XXXXXXX

ACM Reference Format:
Aryamaan Ujwal Dhomne, Joshua McKerracher, and Saurabh Bagchi. 2025. Wasted Cycles and Waiting Games: Analysis of HPC Resource Usage Using Production Cluster Data in FRESCO. In *Proceedings of the 13th International Conference on Data Science (CODS 2025), December 17–20, 2025, Pune, India.* ACM, New York, NY, USA, 9 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

High-Performance Computing (HPC) systems and modern data centers represent significant investments driving scientific and industrial progress. They serve a diverse community spanning academic research, enterprise applications, and cloud services running an equally varied range of workloads, from complex simulations and data analyses to large-scale machine learning models. Consequently, they see a diverse and often unpredictable mix of workloads. A central challenge in managing these systems is the persistent gap between their rated capacity and their operational efficiency as they serve the mix of workloads. While these systems are built for high throughput, their day-to-day operations are frequently marked by inefficiencies that impact both user productivity and institutional return on investment. This paper explores these operational dynamics through a data-driven analysis of job execution records from production HPC environments.

In this paper, we focus on two operational inefficiencies within these HPC systems: job waiting times in queues and resource waste. From the perspective of a user, a persistent annoyance is the opaque nature of job scheduling [9, 18].

Users submit jobs to a queue with little insight into how long they will wait, a delay that directly hinders research progress. This uncertainty often leads users to request more resources than necessary in order to secure a faster allocation. From the administrator's viewpoint, this behavior contributes to the second problem: systemic resource waste. When jobs are allocated more CPU cores, memory, or time than they actually use, those idle resources are unavailable to other users, leading to artificially longer queues and decreased overall system throughput. This cycle of over-requesting and underutilization present a significant hurdle to effective resource management.

Historically, studying these intertwined issues at scale has been difficult due to a scarcity of public operational data from HPC systems. While valuable datasets exist, they often lack fine-grained performance metrics that are linked to job-level attributes across multiple, diverse systems. For instance, valuable public resources like the Atlas cluster trace archive and the Cluster Failure Data Repository (CFDR) [15] have provided important, long-term views

into system workloads and reliability [2, 10]. However, the critical missing component in these archives is the direct linkage between job accounting data and the detailed performance telemetry capturing what resources were actually consumed during execution. To address this, our work leverages the FRESCO dataset, a public, multi-institutional collection of operational data from 20.9 million job records across three academic supercomputing centers spanning 75 months [13]. Our team (with a superset of contributors from this submission's authors) released the latest version of the FRESCO resource in July 2025 [13]. FRESCO's key feature is its linkage of job accounting data with detailed, time-series performance metrics that capture actual resource consumption during a job's execution. This allows for a direct comparison between what was requested and what was used, providing the empirical foundation to analyze both wait times and resource waste.

This study is guided by two core research questions that address the challenges from both the user and administrator perspectives:

- **RQ1**: What job characteristics drive queue wait times in HPC systems, and can we provide users with accurate wait time predictions to inform smarter resource requests and submission strategies?
- **RQ2**: How prevalent and severe is resource waste across different job types and user behaviors, and can we develop systematic approaches to detect and quantify underutilization patterns that administrators can act upon?

In summary, our contributions include: (1) A predictive model achieving 92% F1-score for queue wait times by reformulating the problem as classification with interpretable categories; (2) A methodology quantifying resource waste that reveals 48.5% of allocated resources go unused, with 69.3% of jobs wasting over half their allocations; (3) Evidence that waste increases monotonically with job size (38.9% to 61.9%), suggesting users become more conservative as stakes rise; and (4) An interpretable framework using SHAP values to identify key scheduling factors, built on the public FRESCO dataset to enable reproducible research.

The remainder of this paper is structured as follows. Section 2 provides background on job scheduling and resource utilization and reviews related work in operational data analysis. Section 3 describes the FRESCO dataset and the specific data fields used in our study. Sections 5 and 6 detail the methodologies and results for our analyses of wait times and resource waste, respectively. Finally, Section 7 summarizes our findings, discusses their practical implications, and outlines directions for future work.

2 Background and Related Work

This section situates our research within the broader context of High-Performance Computing (HPC) operations and data analysis. We first provide essential background on the distinct challenges of job scheduling and resource utilization. We then review the historical data limitations that have impeded analyses of these challenges and position our work as a novel contribution that bridges this gap.

2.1 The Challenge of HPC Job Scheduling and Wait Time

A core function of any multi-user HPC system is its job scheduler, a complex software component responsible for allocating computational resources among competing user requests. Schedulers like SLURM, used by the Anvil cluster, and the PBS/TORQUE system, used by Conte and Stampede, are designed to balance multiple objectives, such as maximizing system throughput, enforcing site-specific policies, and ensuring fairness among different users and projects. To manage the continuous stream of submissions, jobs are placed into queues, where they wait for the necessary resources (e.g., CPU cores, memory, GPUs) to become available. This queue wait time is a primary source of frustration for users and a direct impediment to the velocity of scientific discovery. Long and unpredictable wait times not only delay research outcomes but also disrupt workflows, making it difficult for researchers to plan their work effectively.

The complexity of scheduling decisions, which often involves techniques like backfilling (slotting smaller jobs into idle resources reserved for larger, pending jobs), makes it nearly impossible for a user to manually estimate their job's queue wait time. This has motivated a significant body of research aimed at wait time prediction. Early approaches often relied on statistical methods and time-series analysis [4]. More recent efforts have increasingly applied machine learning models, using historical scheduler data to learn the relationships between job characteristics (e.g., number of requested cores, requested timelimit, queue state) and the resulting wait time [16]. While these models have shown promise, their accuracy can be limited by the features available in standard scheduler logs and their generalizability across different systems and workloads remains an open question [3, 17]. Our work contributes to this area by leveraging a rich, multi-system dataset to identify robust predictors of wait time.

2.2 The Pervasive Issue of HPC Resource Underutilization

A related and equally critical challenge in HPC operations is the pervasive issue of resource underutilization, or waste. This occurs when a job is allocated computational resources that it does not actually use. The problem is most commonly observed in three dimensions: CPU/core underutilization (allocated cores remain idle), memory underutilization (allocated memory is not consumed), and wall-time underutilization (jobs finish long before their requested timelimit). This waste is driven by several factors. Users, facing uncertainty about their application's performance and penalized by schedulers for exceeding resource limits, have a strong incentive to be conservative and over-request resources. This "better safe than sorry" approach is a rational response to system policies, but it leads to significant aggregate inefficiency.

When resources are allocated to a job but sit idle, they are effectively removed from the pool available to other users, artificially lengthening queue wait times, and reducing the overall output of the cluster. Previous studies have consistently highlighted this problem [8]. Analyses of various cluster workloads have shown that a substantial fraction of allocated CPU cores often go unused and that jobs, on average, use only a portion of their requested memory [14].

However, quantifying the extent of this problem and identifying its root causes requires more than just scheduler logs. It demands a direct, job-level comparison between the resources a user requested and the resources their application actually consumed, a linkage that has been historically difficult to make at scale.

2.3 The Historical Data Gap in Operational Analytics

The challenges in studying both wait times and resource waste have been compounded by a historical scarcity of suitable public data. For many years, research in this area was heavily influenced by a small number of datasets, most notably the 2011 Google cluster trace [7]. While influential, subsequent work revealed that this single trace was not representative of the diverse workloads found in many academic and research HPC environments [1]. The Atlas project was a significant step forward, releasing a repository of traces from different institutions to address this very issue of workload diversity [2]. Atlas and other valuable resources like the Computer Failure Data Repository (CFDR) made important scheduler and system-level data public.

However, a persistent limitation of many of these datasets was the separation of high-level scheduler data from fine-grained, time-series performance metrics from individual nodes [6]. Scheduler logs contain information on what a user requested (e.g., 128 cores for 24 hours), while node-level monitoring tools capture what resources a job used (e.g., average CPU utilization over time). The difficulty lay in reliably linking these two data sources. Without this linkage, it is extremely difficult to systematically answer a simple but critical question: "For this specific job, how much of the requested resources were actually used?" [5]. This data gap has made it difficult to conduct a unified, large-scale analysis of the relationship between user request patterns, scheduling delays, and resource efficiency.

2.4 Situating Our Analysis with FRESCO

Our research addresses this historical data gap by leveraging the FRESCO dataset, a public resource designed specifically to bridge this divide [13]. FRESCO provides a unified view by integrating job accounting data (resource requests) with fine-grained performance metrics (actual usage) for 20.9 million jobs across multiple HPC centers. This unique, large-scale linkage of requested versus utilized resources provides the empirical foundation to treat wait time prediction and resource waste analysis not as separate issues, but as two interconnected facets of HPC efficiency that can finally be studied together.

3 The FRESCO Resource: A Multi-Institutional View of HPC Operations

The analyses presented in this paper are conducted using the FRESCO dataset, which serves as the empirical foundation for our entire study. This section details the dataset's scope, the diversity of the systems it covers, and the data collection methods used in its construction, establishing its direct relevance to our research questions.

3.1 System Diversity and Generalizability

A key strength of the FRESCO dataset is the heterogeneity of the HPC systems it includes: Purdue University's Anvil and Conte clusters, and the Texas Advanced Computing Center's (TACC) Stampede cluster. These systems represent a diverse range of architectures, scales, and user communities, which enhances the generalizability of our findings. The specific characteristics include:

- `Conte (Purdue)`: A 580-node cluster with Xeon architecture, from which 10.8 million jobs between 2015 and 2017 were captured.
- `Stampede (TACC)`: A large, 6,400-node Intel Xeon E5 system, contributing 8.7 million jobs from the 2013-2016 period.
- `Anvil (Purdue)`: A modern, 1,000-node cluster with AMD EPYC 7763 processors and a separate GPU partition featuring A100 GPUs. It contributes 1.4 million jobs from July 2022 to May 2023.

The inclusion of data from these distinct environments allows our analysis to move beyond the limitations of single-institution studies, ensuring our results are not merely artifacts of a specific system's configuration or workload.

3.2 Data Collection and Integration

The technical contribution of FRESCO is its rigorous integration of two distinct types of data: job accounting records and system performance metrics. For the Conte and Stampede systems, job accounting data was sourced from PBS/TORQUE scheduler logs, while performance metrics were collected using TACC Stats. For the Anvil system, both job accounting and performance data were extracted from the XDMoD monitoring framework, which uses SLURM scheduler data. The crucial step in the dataset's construction was the process of Job Context Association. This process links the fine-grained performance metrics (e.g., actual CPU and memory usage) to the corresponding job's accounting record by ensuring the metric's timestamp falls within the job's execution window and on an allocated node [12]. It is this successful linkage of requested resources from accounting logs to actual measured usage that makes the resource waste analysis in this paper possible.

4 FRESCO Data Schema and Semantics

This section functions as a "legend" for the reader, defining the specific data fields from the FRESCO dataset that are essential for our analyses. A clear understanding of these variables is necessary to appreciate the methodologies used to investigate queue wait times and resource waste. We detail the fields describing the job life-cycle, the classification of job outcomes, and the performance metrics that enable our waste analysis.

4.1 Job Submission and Execution Life-Cycle

To analyze the full life-cycle of a job (from submission to completion) we rely on a set of core temporal and resource allocation attributes present in the unified FRESCO schema. The primary temporal markers include submit_time, start_time, and end_time, which record when a job was submitted to the queue, when it began execution, and when it terminated, respectively. These fields

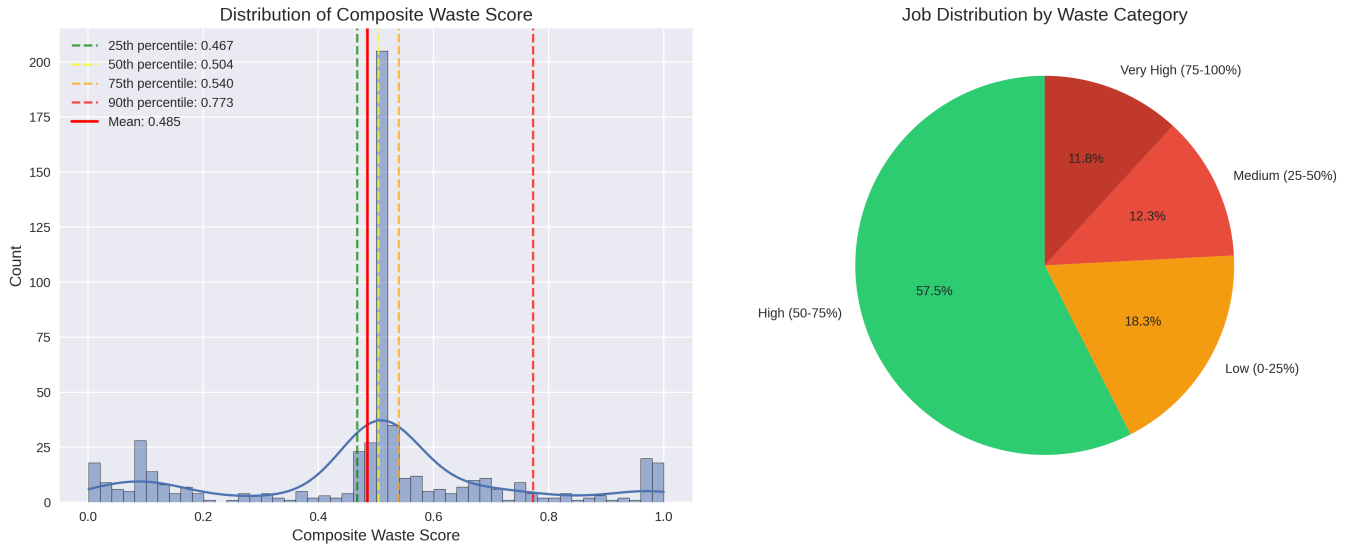

Figure 1: Composite waste score and the distribution of jobs categorized by waste percentage for the Anvil and Conte clusters.

provide the basis for calculating queue wait time (the delta between start_time and submit_time) and job duration.

Alongside these timestamps, our analysis uses the key resource allocation attributes that define a user's request to the scheduler. These include ncores (the number of requested CPU cores), nhosts (the number of requested nodes), and timelimit (the maximum requested execution time). These fields represent the user's intended resource consumption and are the primary features used in our wait time prediction models.

4.2 Job Outcome Analysis and Classification

Understanding how a job terminates is critical for filtering the data and interpreting utilization patterns. The exitcode field, part of the job's execution context, provides this information. Because the raw exit codes can differ between the schedulers used by the source clusters (PBS/TORQUE for Conte and Stampede, and SLURM for Anvil), a crucial step in the FRESCO data integration process was the semantic alignment of these codes. This involved translating the native scheduler exit states into a standardized set of outcomes, such as COMPLETED, FAILED, and ABORTED. This standardized classification allows for consistent, cross-cluster analysis, ensuring that when we compare, for example, the resource waste of FAILED jobs to COMPLETED jobs, the definitions are uniform across all three systems.

4.3 Performance Metrics for Waste Detection

A key counterpart to analyzing wait times is the ability to quantify what happens during a job's execution, which is enabled by FRESCO's fine-grained performance metrics that capture actual resource consumption. For this study, the most critical fields are value_cpuuser, which measures user-space CPU activity as a percentage of the allocated cores, and value_memused, which tracks total physical memory consumption in gigabytes. These time-series metrics are recorded throughout a job's runtime and are associated directly with a specific job ID and host. It is this direct linkage of performance data to job accounting data that allows us to move beyond historical limitations and directly compare the resources a user requested (e.g., ncores) with the resources their job actually used (e.g., value_cpuuser), providing an empirical basis for our resource waste analysis.

5 Analysis 1: Characterizing and Predicting Queue Wait Time

This section addresses our first research question by investigating the factors that drive queue wait times and developing a predictive model to provide users with actionable estimates.

5.1 Methodology

We began our analysis by approaching the problem as a regression task, aiming to predict the exact wait time of HPC jobs. This approach was initially assumed to offer the most utility to users by giving them a precise estimate of how long their job would remain in the queue. However, baseline regression models using standard features yielded poor results. Even after refining feature selection, models continued to perform poorly, which led us to reformulate the problem as a classification task.

In the classification setup, jobs were grouped into four discrete wait time buckets: <10 minutes, 10 minutes–1 hour, 1–6 hours, and >6 hours. These categories were chosen to reflect typical user behavior and job scheduling expectations. A job with an expected wait time under 10 minutes might justify briefly waiting at the terminal, while jobs in the 10 minutes–1 hour range may encourage light context switching. Jobs in the 1–6 hour range may cause users to shift focus to a different task or return after several hours, and jobs expected to wait more than 6 hours are often left unattended until a notification is received.

The dataset consisted of 64,607 jobs submitted between January 2023 and June 2023. Jobs were submitted to a large-scale, production HPC system (Anvil), with 89.85% classified as single-node and 10.14% as multi-node. To remove noise from the dataset, we filtered out 34.8% of jobs with total runtimes under 10 minutes, which were typically test jobs, accidental submissions, or misconfigured workloads.

Jobs were then stratified by host type (single vs. multi-node) and analyzed separately due to their differing scheduling patterns. We further restricted our analysis to jobs submitted to the Anvil cluster, excluding data from other environments for consistency. The dataset was split temporally to better reflect a real-world deployment: the first 80% of jobs by submission time were used for training, and the most recent 20% were held out for evaluation.

Only features available at job submission time were used in modeling. These included time, submit_time, timelimit, nhosts, ncores, account, queue, jid, and username. We excluded jobname due to its extremely high cardinality, which made it impractical for one-hot encoding and likely to introduce overfitting.

All categorical features were one-hot encoded, as most had relatively low cardinality, enabling interpretable expansion without significantly increasing model complexity.

The analysis was conducted using Python 3.13 with key packages including xgboost==3.0.2, scikit-learn==1.6.1, imbalanced-learn==0.13.0, and shap==0.48.0.

5.2 Results and Discussion

Using only the existing features, early results were poor across all models. XGBoost achieved a maximum mean F1-score of 0.64, while Random Forest and logistic regression performed significantly worse, with mean F1-scores of 0.19 and 0.24 respectively. Note that categorical features were encoded using one-hot encoding due to their relatively low cardinality. The underwhelming results from initial models motivated us to develop new features that better captured the structure and dynamics of the system.

We engineered a new set of features designed to reflect system-level load and submission-time context. These included the number of overlapping pending jobs in the same queue or across the entire cluster, the number of free cores at the time of job submission, and the number of busy cores both at the queue and cluster level. We also included features indicating how many jobs were pending for the same user and account. To capture seasonal submission patterns, we extracted time-based features such as whether the job was submitted on a weekend, during the night, in the morning, and the specific hour of the day.

Given the imbalance in class distribution, we applied SMOTE (Synthetic Minority Over-sampling Technique) to the training set to improve model performance. Without applying SMOTE, results remained poor—even XGBoost only reached a maximum mean F1-score of 0.63. After applying SMOTE and completing the full feature engineering pipeline, we tuned model hyperparameters and achieved our best results with XGBoost, which reached a mean F1-score of 0.92. Model hyperparameters were tuned using the Optuna framework, which effectively explored the search space to optimize validation performance. The dataset exhibited a highly imbalanced class distribution, dominated by jobs completing in under 10 minutes, which accounted for approximately 88% of samples. The other classes—jobs taking over 6 hours, between 10 minutes and 1 hour, and between 1 and 6 hours—comprised much smaller proportions, ranging from about 1% to 10%.

The XGBoost model demonstrated strong overall performance, achieving an accuracy of 0.98 on the validation set. Metrics broken down by job class indicated excellent precision, recall, and F1-scores for the majority class (<10min), with scores close to or at 0.99 across precision and recall. For the minority classes, performance was generally good but revealed some challenges. The "10min – 1 hr" class achieved perfect recall but lower precision (0.74), suggesting some false positives in this category. The "1–6hr" and ">6hr" classes had precision and recall values in the low to mid-90s, showing the model's robustness despite the imbalance.

The macro averages for precision, recall, and F1-score were 0.90, 0.96, and 0.92 respectively, which highlight the model's relatively strong but uneven performance across all classes. This underscores that while the model is very effective in identifying the majority class, there remains room for improvement in handling minority classes, potentially through data balancing methods or modified loss functions. When applying the same methodology to multi-node jobs, XGBoost achieved a mean F1-score of 0.90, showing strong generalization across job types.

To interpret the model, we used SHAP (SHapley Additive exPlanations) values to examine feature importance. Features such as `pending_jobs_same_user`, `total_cores_free_queue`, and `cluster_total_jobs_pending` or `cluster_total_jobs_running` had the highest influence—consistent with their direct relationship to system load and scheduling. In addition to expected trends, we also uncovered unexpected insights. For example, jobs submitted under the account GROUP31 were disproportionately classified into the <10-minute category, suggesting that this account might have elevated scheduling priority. Similarly, the queue_shared feature showed high absolute SHAP values. While the SHAP summary plot reflects magnitude alone, the actual SHAP values for queue_shared had a negative influence on longer wait time classifications, suggesting that jobs submitted to the shared queue were typically scheduled more quickly than those submitted to whole-node queues.

5.3 Practical Implications

A user-centered approach to predictive modeling in HPC queue management extends beyond mere accuracy—interpretability and user experience (UX) are central to maximizing the utility of model outputs. By providing explicit, category-based wait time predictions, the models directly support users in structuring their workflows, encouraging productive context-switching and reducing uncertainty inherent to HPC environments.

Crucially, the interpretability of the models enhances this value proposition. Analysis of SHAP (SHapley Additive exPlanations) values reveals the direct impact of individual features on each prediction. For example, features such as system load, the number of competing pending jobs for the same user, and queue selection were consistently identified as having substantial influence over predicted wait time categories. The high importance of these features not only validates their intuitive role in scheduling outcomes

but also arms users and administrators with actionable insights. Users learn which submission-time factors most affect their job's position in the queue, empowering them to adjust submission strategies—such as targeting less congested queues or timing submissions to periods of lower system load—to potentially reduce their wait times.

Furthermore, such transparency fosters trust: when users understand why certain jobs are predicted to experience longer or shorter waits, they are more likely to rely on the system's guidance and less likely to be frustrated by occasional misclassifications. Administrators, in turn, can leverage these insights to refine scheduling policies, ensure fairer resource allocation, and identify possible bottlenecks or systemic inefficiencies reflected in SHAP-derived patterns.

By foregrounding both interpretability and user experience, predictive models in HPC settings move from being "black box" tools to becoming interactive feedback mechanisms that help users make informed choices and extract greater value from shared computing resources.

6 Analysis 2: Detecting and Quantifying Resource Waste

This section addresses our second research question (RQ2) by examining the prevalence and severity of resource waste across different job types and user behaviors in the FRESCO dataset. We present a comprehensive analysis of resource underutilization patterns that can inform both administrative policies and user education initiatives.

6.1 Methodology

We implemented a multi-stage approach that ensures reproducibility and scientific validity.

Artifact Availability. All code to reproduce these analyses is available at: github.com/j-mckerracher/fresco-hpc-analyses/tree/main/comad (tag `1.0.0`, commit `0d6068f`) [11].

6.1.1 Resource Waste Metrics Definition. We defined resource waste through direct comparison of requested versus utilized resources. For each job j, we calculated:

CPU Waste: The fraction of allocated CPU resources that remained idle during job execution:

$$W_{CPU}^{(j)} = 1 - \frac{U_{CPU}^{(j)}}{100}$$

where $U_{CPU}^{(j)}$ represents the average CPU utilization percentage recorded during job execution.

Memory Waste: The fraction of allocated memory that was not consumed:

$$W_{mem}^{(j)} = 1 - \frac{M_{used}^{(j)}}{M_{requested}^{(j)}}$$

where $M_{used}^{(j)}$ is the average memory usage and $M_{requested}^{(j)}$ is estimated based on system-specific memory-per-core allocations: 2.0 GB/core for Stampede and Anvil, 4.0 GB/core for Conte.

Composite Waste Score: An equally-weighted combination of CPU and memory waste:

$$W_{composite}^{(j)} = 0.5 \times W_{CPU}^{(j)} + 0.5 \times W_{mem}^{(j)}$$

6.1.2 Data Processing and Statistical Analysis. To keep the resource-waste computations tractable, we enumerated the dataset across 61,671 hourly Parquet shards and employed a two-stage sampling scheme: we scanned every ∼20th hourly shard to enumerate job IDs, then drew a 10% simple random sample of jobs (the unit of analysis) from that frame, yielding $n = 567$ sampled jobs. Statistical significance was assessed using ANOVA for parametric comparisons and Kruskal–Wallis tests for non-parametric analyses. Confidence intervals were calculated using Student's t-distribution at the 95% confidence level.

6.2 Results and Discussion

Our analysis reveals pervasive resource underutilization across the HPC systems studied, with significant variations based on job characteristics and system configurations.

The distribution of composite waste scores (Figure 1) shows a strongly bimodal pattern. While 18.3% of jobs demonstrated efficient resource usage with waste below 25%, a striking 69.3% of jobs exhibited waste exceeding 50%. This polarization is even more pronounced at higher thresholds: 11.8% of jobs wasted more than 75% of their resources, and 7.4% showed extreme waste exceeding 90%.

6.2.1 Overall Waste Prevalence. The analysis of 567 sampled jobs revealed substantial resource underutilization. CPU waste averaged 29.6% (95% CI: 26.6%-32.6%, median: 8.4%), while memory waste was dramatically higher at 67.4% (95% CI: 63.8%-71.1%). The composite waste score averaged 48.5% (95% CI: 46.6%-50.5%), indicating that nearly half of all allocated resources remained unused.

6.2.2 Economic Impact Analysis. The economic implications of this waste are substantial. The sampled jobs collectively wasted:

- 566,035 CPU-hours (averaging 998 CPU-hours per job)
- 4.28 million GB-hours of memory (averaging 7,553 GB-hours per job)

Extrapolating to the full dataset of 20.9 million jobs suggests tens of millions of wasted CPU-hours annually, representing a significant opportunity cost for these publicly-funded resources.

6.2.3 Waste Patterns by Job Characteristics. Waste patterns vary depending on which characteristic of the job is used for analysis.

Exit Code Analysis: Job termination status showed highly significant differences in waste patterns (ANOVA: F=13.49, p<0.001).

- NODE_FAIL: 63.4%
- TIMEOUT: 58.8%
- CANCELLED: 54.3%
- COMPLETED: 47.6%
- FAILED: 39.1%

Job Size Effects: Resource waste increased monotonically with job size (Kruskal-Wallis: H=98.21, p<0.001):

- Small jobs (1-16 cores): 38.9% waste
- Medium jobs (17-64 cores): 54.7% waste
- Large jobs (65-256 cores): 57.9% waste

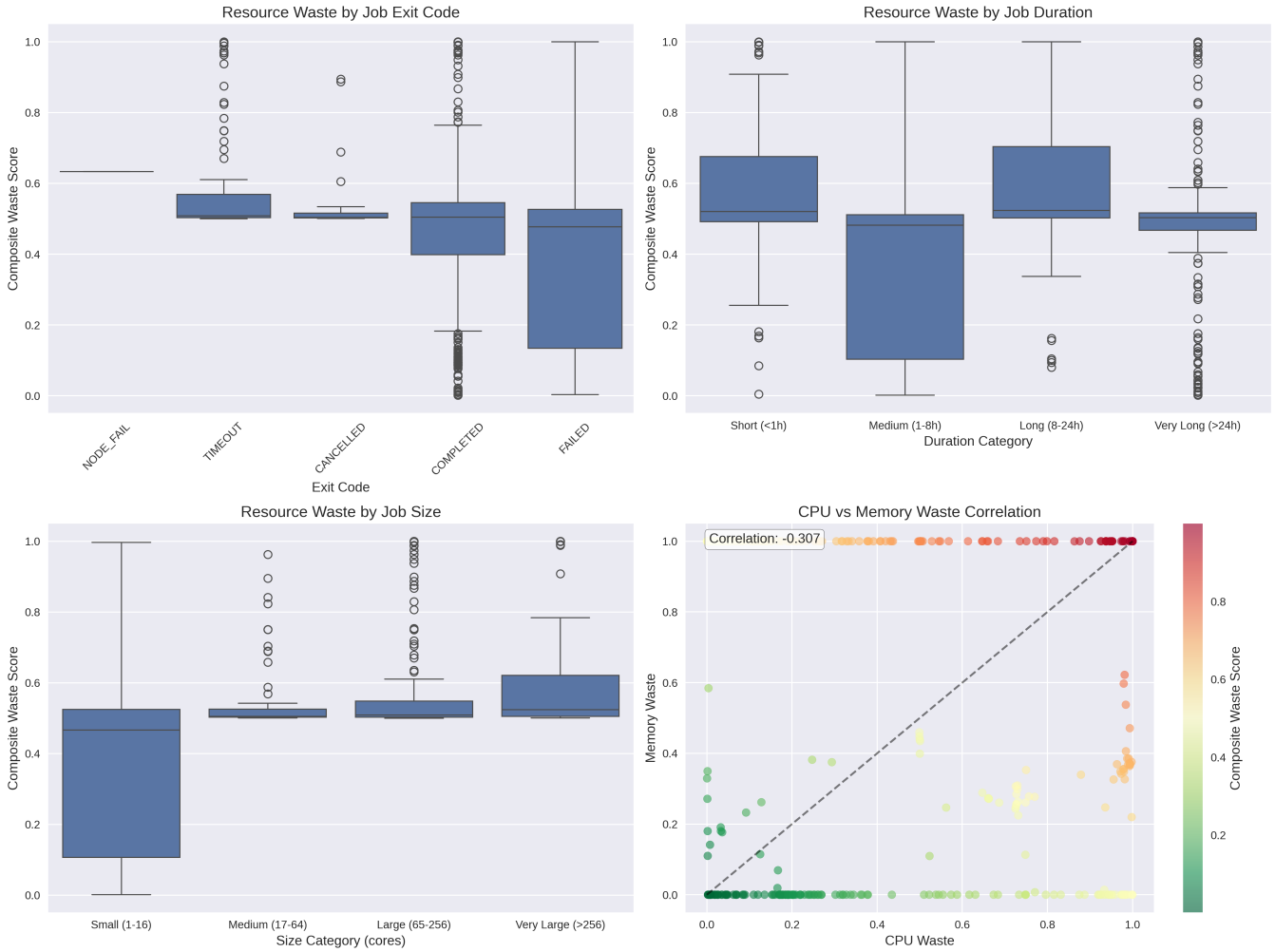

Figure 2: A breakdown of waste by job characteristics along with CPU-memory waste correlation.

- Very large jobs (>256 cores): 61.9% waste

This trend indicates that users become increasingly conservative in their resource estimates as job sizes grow, possibly due to the higher cost of job failures at scale.

Duration Patterns:

- Short duration jobs (<1 hour): 57.4%
- Medium duration jobs (1-8 hours): 35.7%
- Long duration jobs (8-24 hours): 60.1%
- Very long duration jobs (>24 hours): 47.1%

The high waste in short jobs likely reflects fixed initialization overheads, while longer job waste may stem from conservative timelimit requests.

6.2.4 Correlation Analysis. Surprisingly, CPU and memory waste showed a negative correlation (r=-0.31), suggesting that jobs efficiently using CPU tend to over-request memory and vice versa. CPU waste showed minimal correlation with job duration (r=-0.05) or size (r=-0.03), indicating that waste patterns are not simply a function of job scale but reflect more complex user behaviors and application characteristics.

6.2.5 System Comparison.

- Conte: Higher CPU waste (39.8%) but lower memory waste (15.6%), resulting in a 30.1% composite waste.
- Anvil: Balanced waste profile with 35.6% CPU and 19.2% memory waste (29.1% composite).
- Combined dataset: Lower CPU waste (28.4%) but very high memory waste (64.2%), yielding 46.3% composite waste.

These differences likely reflect varying memory-per-core ratios, user communities, and differing workload characteristics across systems.

6.3 Practical Implications

Our findings have several important implications for HPC operations:

6.3.1 For System Administrators.

- Priority Interventions: Focus on large jobs (>256 cores) and short jobs (<1 hour), which show the highest waste levels and represent the greatest optimization opportunities.
- Real-time Monitoring: Implement waste detection systems that flag jobs exceeding 75

6.3.2 For Users.

- Resource Profiling: The negative correlation between CPU and memory waste suggests users need better tools to profile both dimensions simultaneously.
- Size-Aware Guidelines: Provide escalating guidance for resource requests as job sizes increase, addressing the tendency to over-request at scale.
- Targeted Education: Focus training on the 11.8% of jobs with >75% waste, where interventions can have maximum impact.

6.3.3 For Policy Development.

- Waste-Based Scheduling: Consider implementing "waste budgets" that limit users' ability to request resources based on historical efficiency.
- Dynamic Resource Adjustment: Develop mechanisms for runtime resource reduction when sustained underutilization is detected.
- Incentive Alignment: Create reward structures that recognize efficient resource usage, potentially through priority queue access or allocation bonuses.

The pervasive resource waste identified represents a critical challenge and a transformative opportunity. Addressing even half of this waste could effectively increase system capacity by 25% without hardware investment. The central insight is that waste patterns vary dramatically by user, system, and job type, requiring targeted rather than one-size-fits-all solutions. The statistical significance of our findings ($p<0.001$ for major comparisons) provides confidence that these patterns are systematic rather than random, making them amenable to policy and technical interventions.

7 Conclusion and Future Work

This final section summarizes the key findings of our study, reflects on its broader impact and limitations, and suggests promising directions for future research in data-driven HPC operations.

7.1 Broader Impact

Our findings extend beyond the specific HPC systems analyzed, demonstrating how data-driven approaches can transform resource management in shared computational environments. The 48.5% average resource waste represents a significant economic impact. Potentially hundreds of millions of dollars annually if similar patterns exist across academic and national laboratory facilities. The methodological contribution of linking job requests with actual performance metrics provides a template applicable to cloud computing, distributed systems, and other shared resources. Our emphasis on model interpretability through SHAP values demonstrates how machine learning can augment human decision-making in operational settings, providing users with both predictions and actionable insights. Most significantly, our work reveals how data analysis can identify and break negative feedback loops—such as the cycle where wait time uncertainty drives over-requesting—a pattern likely present in other resource allocation contexts from healthcare to transportation systems.

7.2 Limitations

Several limitations bound our findings. The FRESCO dataset, while spanning three systems, represents specific time periods (2013-2023) and academic environments, potentially missing recent AI/ML workload patterns and industrial HPC characteristics. Our waste analysis assumptions—including memory-per-core estimates and equal CPU/memory weighting—may not universally apply, and we omit increasingly important resources like GPU utilization and I/O performance. The classification approach simplifies continuous queue dynamics, and model performance may degrade under different scheduling policies. Critically, we cannot establish causality between observed patterns and their drivers; our hypothesis that uncertainty drives over-requesting remains correlational without controlled experiments. Selection bias may also affect our results, as failed submissions and users who abandoned the systems are not captured in the data.

7.3 Future Work

Future research should pursue several directions. Real-time optimization systems could dynamically adjust allocations for underutilized jobs, though balancing efficiency against stability remains challenging. The analytical framework—comparing requested versus actual usage—applies broadly to cloud platforms and other shared resources. Deep learning approaches using RNNs or transformers could capture complex queue dynamics for improved predictions. Behavioral studies should investigate how information presentation and incentive structures affect resource requests, potentially using game theory to align individual and system-wide efficiency. Expanding analysis to GPU, network, and I/O resources would provide comprehensive efficiency metrics as heterogeneous computing grows. Finally, federated learning could enable multi-site optimization while preserving workload privacy, allowing HPC centers to collaboratively improve operations without sharing sensitive data.

8 Acknowledgements

This work was ably supported by Carol Song of Purdue's Information Technology Department, Stephen Harrell of the Texas Advanced Computing Center (TACC), and Rajesh Kalyanam of Oak Ridge National Laboratory. This material is based in part upon work supported by the National Science Foundation under Grant Numbers CNS-2016704 and CCF-2140139. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsor. Claude.ai was utilized to proofread sections of this work.

References

[1] George Amvrosiadis, Christopher Beck, Gregory R. Ganger, Elisabeth Moore, Jun Woo Park, Chuck Cranor, and Michael Kuchnik. 2017. What Do Cluster Jobs Look Like Outside Google?. In *Proceedings of the 8th ACM European Conference on Computer Systems (EuroSys '17)*. 354–369. doi:10.1145/3064176.3064210

[2] George Amvrosiadis, Michael Kuchnik, Jun Woo Park, Chuck Cranor, Gregory R Ganger, Elisabeth Moore, and Nathan DeBardeleben. 2018. The Atlas cluster trace repository. *Usenix Mag* 43, 4 (2018), 29–35.

[3] Nick Brown, Gordon Gibb, Evgenij Belikov, and Rupert Nash. 2022. Predicting batch queue job wait times for informed scheduling of urgent HPC workloads. *arXiv preprint arXiv:2204.13543* (2022). https://arxiv.org/abs/2204.13543

[4] Allen B. Downey. 1996. *A Parallel Workload Model and its Implications for Processor.* Technical Report. USA.

[5] Todd Evans, William L. Barth, James C. Browne, Robert L. DeLeon, Thomas R. Furlani, Steven M. Gallo, Matthew D. Jones, and Abani K. Patra. 2014. Comprehensive resource use monitoring for HPC systems with TACC stats. In *Proceedings of the First International Workshop on HPC User Support Tools* (New Orleans, Louisiana) *(HUST '14)*. IEEE Press, 13–21. doi:10.1109/HUST.2014.7

[6] Dror G. Feitelson, Dan Tsafrir, and David Krakov. 2014. Experience with using the Parallel Workloads Archive. *Journal of Parallel and Distributed Computing (JPDC)* 74, 10 (October 2014), 2967–2982.

[7] Google Research Blog. 2020. Yet More Google Compute Cluster Trace Data. https://research.google/blog/yet-more-google-compute-cluster-trace-data/. Accessed: 2025-08-15.

[8] Jie Li, George Michelogiannakis, Brandon Cook, Dulanya Cooray, and Yong Chen. 2023. Analyzing Resource Utilization in an HPC System: A Case Study of NERSC Perlmutter. In *International Supercomputing Conference (ISC) (Lecture Notes in Computer Science, Vol. 13948)*. 297–316. doi:10.1007/978-3-031-32041-5_16

[9] librapenseur. 2024. SLURM jobs running much slower under most circumstances. Reddit Post. https://www.reddit.com/r/HPC/comments/1axvogm/slurm_jobs_running_much_slower_under_most/ Accessed: 2025-08-14.

[10] Los Alamos National Laboratory. [n. d.]. Failure Data. U.S. Reliability and Resilience Characterization (USRC). https://usrc.lanl.gov/data%20sources/failure-data.php

[11] Joshua McKerracher. 2025. FRESCO HPC Analyses: COMAD workflows. https://github.com/j-mckerracher/fresco-hpc-analyses/tree/main/comad. Accessed: 2025-08-16; tag 1.0.0, commit 0d6068f.

[12] Joshua Stephen McKerracher. 2025. hpc_transformers.py (Lines 122–219). https://github.com/j-mckerracher/fresco-hpc/blob/main/data-pipeline/hpc_etl_pipeline/src/transformers/hpc_transformers.py#L122-L219. Accessed: 2025-08-16.

[13] Joshua Stephen McKerracher, Preeti Mukherjee, Rajesh Kalyanam, and Saurabh Bagchi. 2025. FRESCO: A Public Multi-Institutional Dataset for Understanding HPC System Behavior and Dependability. In *Practice and Experience in Advanced Research Computing (PEARC '25)*. Association for Computing Machinery (ACM), New York, NY, USA, Article 111, 6 pages. doi:10.1145/3708035.3736090

[14] Ivy Peng, Ian Karlin, Maya Gokhale, Kathleen Shoga, Matthew Legendre, and Todd Gamblin. 2022. A Holistic View of Memory Utilization on HPC Systems: Current and Future Trends. In *Proceedings of the International Symposium on Memory Systems* (Washington DC, DC, USA) *(MEMSYS '21)*. Association for Computing Machinery, New York, NY, USA, Article 14, 11 pages. doi:10.1145/3488423.3519336

[15] Bianca Schroeder and Garth Gibson. 2015. The Computer Failure Data Repository (CFDR). https://www.usenix.org/cfdr

[16] Warren Smith, Ian Foster, and Valerie Taylor. 1998. Predicting application run times using historical information. In *Job Scheduling Strategies for Parallel Processing*, Dror G. Feitelson and Larry Rudolph (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 122–142.

[17] Chiara Vercellino, Alberto Scionti, Giuseppe Varavallo, Paolo Viviani, Giacomo Vitali, and Olivier Terzo. 2023. A Machine Learning Approach for an HPC Use Case: the Jobs Queuing Time Prediction. *Future Generation Computer Systems* 143 (2023), 215–230. doi:10.1016/j.future.2023.01.020

[18] yourradio. 2023. HPC usage etiquette. Reddit Post. https://www.reddit.com/r/HPC/comments/13dm0xt/hpc_usage_etiquette/ Accessed: 2025-08-14.

Received 17 August 2025