AN ANALYSIS OF WORKLOAD PATTERNS IN BORG CLOUD CLUSTER TRACES

An Undergraduate Research Scholars Thesis

by

ZENGXIAORAN KANG

Submitted to the LAUNCH: Undergraduate Research office at
Texas A&M University
in partial fulfillment of the requirements for the designation as an

UNDERGRADUATE RESEARCH SCHOLAR

Approved by
Faculty Research Advisor: Dr. Dilma Da Silva

May 2022

Majors: Computer Science
Statistics

RESEARCH COMPLIANCE CERTIFICATION

Research activities involving the use of human subjects, vertebrate animals, and/or biohazards must be reviewed and approved by the appropriate Texas A&M University regulatory research committee (i.e., IRB, IACUC, IBC) before the activity can commence. This requirement applies to activities conducted at Texas A&M and to activities conducted at non-Texas A&M facilities or institutions. In both cases, students are responsible for working with the relevant Texas A&M research compliance program to ensure and document that all Texas A&M compliance obligations are met before the study begins.

I, Zengxiaoran Kang, certify that all research compliance requirements related to this Undergraduate Research Scholars thesis have been addressed with my Research Faculty Dr. Dilma Da Silva prior to the collection of any data used in this final thesis submission.

This project did not require approval from the Texas A&M University Research Compliance & Biosafety office.

TABLE OF CONTENTS

Page

ABSTRACT

An Analysis of Workload Patterns in Borg Cloud Cluster Traces

Zengxiaoran Kang
Department of Computer Science and Engineering
Department of Statistics
Texas A&M University

Research Faculty Advisor: Dr. Dilma Da Silva
Department of Computer Science and Engineering
Texas A&M University

The drastic emergence of dynamic, heterogeneous, and shared cloud computing clusters has impacted corporations, researchers, and software developers for the past decade. With the aim to improve further the field of cloud data processing and management, it is essential to understand the workload characteristics of large-scale cloud data centers. We analyze the publicly available trace data released by Google in 2019, deriving an approach that can be used on other large-scale trace datasets from industry, such as the recently released data from Microsoft Azure and Alibaba. The notable workload characteristics of heterogeneity in resource types and usage in the Google traces suggest a highly dynamic environment, with varying jobs that demand faster and more scalable scheduling decisions.

In the Google Borg trace dataset, comparing the overall usage of the cluster to its capacity, the average utilization of the cluster for each tier is overcommitted and demonstrates a frequent regulation of preemption to achieve its high utilization. Using K-means clustering and Lasso regression, we gain valuable insights into the characteristics of Borg's scheduling patterns. Moreover, we perform the reverse-engineering technique with supervised classification methods to un-

derstand the key factors being considered in the latest prediction algorithm for resource demands. Since scheduling and utilization datasets are tremendously large, we manually sample them down to an appropriate size. Although the sample size prevents us from generalizing overall trace behaviors, the analytical method we describe nonetheless extracts system design insights that can be useful in scheduling decision-making for large-scale clusters.

ACKNOWLEDGMENTS

Contributors

I would like to thank my faculty advisor, Dr. Dilma Da Silva for her guidance and support throughout the course of this research.

Thanks also go to my friends, colleagues, and the department faculty and staff for making my time at Texas A&M University a wonderful experience.

And last but certainly not least, thanks go to my family for their inspiration, encouragement, and endless patience. Without them, none of this would have been possible.

The data used for *An Analysis of Workload Patterns in Borg Cloud Cluster Traces* was made publically available by Google in May 2019. The analyses depicted in this thesis were conducted and completed by myself, independently through the academic year '21-'22.

Funding Sources

Undergraduate research was supported by Texas Sea Grant College Program at Texas A & M University.

This work was made possible in part by the National Science Foundation under Grant Numbers SPX-1919181 and CCF-1934904. Its contents are solely the responsibility of the authors and do not necessarily represent the official views of the National Science Foundation.

NOMENCLATURE

TiB	Tebibyte
GB	Gigabytes
PB	Petabytes
CPU	Central processing unit
WSC	Warehouse scale cloud
DoS	Denial-of-service attacks
ALTS	Application-layer transport security
VMs	Virtual machines
RAM	Random access memory
RPCs	Remote procedure calls
UI	User interface
GCU	Google compute units
NCU	Normalized Google compute units
SLOs	Service level objectives
GBM	Gradient boosting algorithm

1. INTRODUCTION

The sheer volume of data storage and usage in the cloud has increased at a tremendous scale, ranging from gigabytes to petabytes per day. The computing industry has witnessed an explosion of system log data, sometimes at an overwhelming rate. In contrast, academic researchers are often confronted by a lack of representative information that can be used to evaluate new approaches for system optimization. Our analysis of publicly accessible traces [1] may help researchers to explore patterns underlining workload scheduling in large-scale compute clusters. Through such exploration, researchers may discover novel approaches for further increasing system performance.

The lack of information from real-world, in-production workloads is not surprising, due to commercial corporate concerns about revealing their data. There are severe risks, such as the usage of reverse engineering to discover business secrets (e.g., data center usage, scale, and organization) and the leakage of private user information. A paper by Mogul et al [2] describes the challenges from Google's perspective.

To ensure information security, Warehouse Scale Cloud (WSC) providers, such as Google, Amazon AWS, and Microsoft Azure, deployed technologies at the hardware level, not limited but including a restricted number of permitted personnel onto the data center floor, biometric identification, cameras, laser-based intrusion detection systems, etc. [3]. At the software level, WSC providers employed a custom set of solutions. Some of the must-have standards for trusting workload distribution infrastructure included cryptographic computation, data traffic encryption, internal network protocols, and many specialized constraints (for example, the Google application-layer transport security (ALTS) targeted the response for detecting the ever-present denial-of-service (DoS) attacks [3]).

1.1 Problem Addressed

When researchers are involved in improving the efficiency of WSC systems, traces are generated with anonymization (i.e., a data processing technique that removes or modifies personally

identifiable information) and protected through non-disclosure agreements or employment relationships. Even when these data traces are made public, without the privilege of collaboration agreements and the overall system knowledge that comes from them, there are obstacles for a thorough comprehension of their structure, for instance, which project the data is exactly describing or what specific system design insights can be extracted from machine attributes. Nonetheless, our statistical analysis confirms that traces can reveal many valuable insights about the workload represented.

1.2 Related Work

The problem of how to schedule computing tasks across larger computer clusters to optimize the overall system efficiency has been explored for many decades [4, 5]. The use of traces to identify general and workload-specific opportunities to improve resource management has been studied in the context of multiprocessor systems [6, 7, 8], parallel computing [9], and cluster computing [10]. Our work has a different objective: assess if statistical analysis of a trace dataset can reveal trends in the absence of specific knowledge of the traced systems.

Most work in the literature uses the Google Borg 2019 dataset [1] to evaluate scheduling techniques [11, 12, 13]. Our work takes a different route: as done in [14] for a 2011 dataset, we analyze the dataset in search of patterns not discussed in the original analysis presented by Tirmazi et. al. [15].

2. GOOGLE BORG QUICK OVERVIEW

Google's Borg is a large-scale cluster manager that achieves high utilization of computing resources by combining admission control, efficient task-packing, over-commitment, and machine sharing with process-level performance isolation[16]. The primary users of Borg are internal Google developers and system administrators who deploy and monitor Google's industrial applications and services. Every day, Borg efficiently handles millions of job requests and provides reliable services to thousands of applications across thousands of machines in assembled clusters.

In 2011, Google published a demo trace V1 and a 1-month trace V2 of Borg, enabling external researchers to study the scheduling workloads in computing clusters [17]. The work in this thesis analyzed the latest trace V3 [1], which is a cluster manager log that documents detailed Borg job scheduling information from 8 different Google compute clusters (Borg cells) for the entire month of May 2019 [1]. The following chapter introduces Borg's infrastructure, scheduling process, and the improvements made from V2 and V3 traces to help understand Borg's fundamentals and explain experimental results.

2.1 Borg Units

2.1.1 Tasks & Collections

A job in Borg consists of one or more tasks that all run in the same program, defined by some general properties (e.g., name, owner, number of tasks in a job) and constraints that force particular tasks to run on specific attributes [14]. To avoid extra overhead in virtualization, Borg maps each task directly to a set of Linux processes running in a container on a machine so that the majority of the workload does not need to run inside virtual machines (VMs) [18]. A Borg *alloc* (short for allocation) reserves resources for one or more tasks on a machine in the future and can gather similar tasks from different jobs onto the same machine to optimize efficiency. A group of alloc instances forms an alloc set that reserves resources on multiple machines into which users can schedule additional jobs [14].

From a brevity and similarity perspective, "task" refers to an alloc or top-level task, and "collection" refers to a job or alloc set. Alloc sets only account for 2% of the collections but are important features for production workloads because Alloc sets represent 20% of the total CPU allocations and 18% of the RAM [15]. Jobs inside allocs have a higher average memory utilization (73%) than other jobs (41%) [15].

2.1.2 Cluster & Cells

Each job runs in one Borg cell, which is a single managed unit that consists of a set of machines. In the V3 trace, the average and median cell size is about 10,000 servers. The machines in a cell connected through the high-performance datacenter-scale network fabric belonging to a single cluster [19]. A defined cluster typically hosts one large cell and may optionally equip a few smaller-scale test or special-purpose cells. Each cluster is inside a data center building, and a collection of buildings composes a site. Although machines in a cell have heterogeneous machine features in many dimensions, such as sizes (CPU, RAM) and processor type, Borg hides the complexity of those differences from users by determining where to run tasks, how to allocate resources, installing necessary dependencies, and monitoring tasks' health [16]. Figure 2.1 illustrates the relationship between cells, clusters, and data center sites.

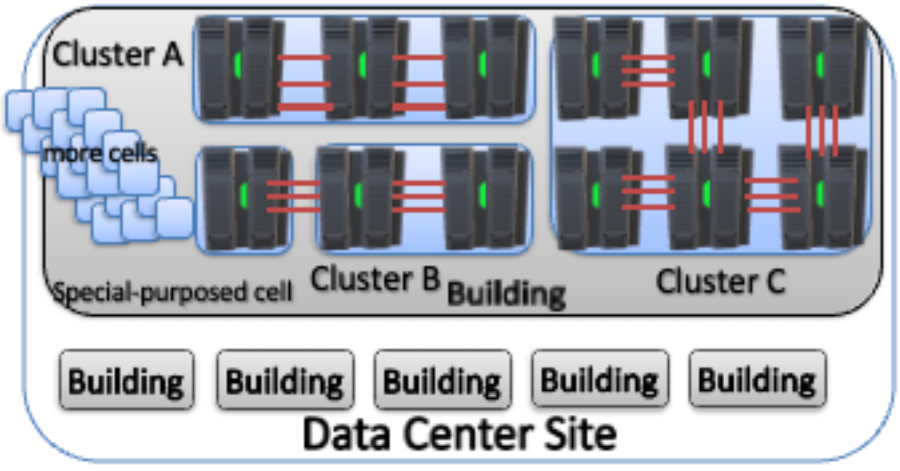

Figure 2.1: A relation overview of Borg machines, cells, clusters, and data center sites.

2.2 Borg Architecture

Cluster managers take into consideration several aspects of the workload and available hardware when optimizing system usage; therefore, they can be quite complex. Borg has been operational for more than a decade and has been evolving to adapt to new trends in the composition and usage of clusters. This section presents the Borg architecture as documented in the literature in 2020.

2.2.1 Infrastructure

Borg has a centralized manager called the *Borgmaster* that is used to control resources on each cell, and an agent process manager, the *Borglet*, that runs on each machine in a cell. A Borglet helps to start, stop, and restart tasks if they fail [16]. Users manage jobs by issuing remote procedure calls (RPCs) to Borg via a web-based user interface (UI) called Sigma, via which users can review the state of all jobs [16].

The Borgmaster handles client RPCs by communicating with Borglets and sending feedback to Sigma. Together they collaborate to mutate states of jobs (e.g., create jobs), manage states of objects in the system (e.g., machines, tasks, allocs, etc.), and provide access to a read-only view of the data (e.g., lookup jobs). In addition, to prevent recovery storms and avoid any manual flow control mechanism, the Borgmaster polls each Borglet every few seconds to retrieve and verify the machine's current state. Therefore, the Borgmaster has full control over the rate of communication [20].

2.2.2 Workloads

Borg cells run a set of heterogeneous workloads with two main parts [16]: (1) Long-running services handling short-lived latency-sensitive requests, particularly applied for end-user-facing products such as Gmail and Google search and (2) Batch jobs that are less sensitive to short-term performance fluctuations present in come-and-go patterns. Many Google essential application frameworks have been built on top of Borg, including MapReduce (system specialized in processing and generating large dataset using a cluster of machines) [21], Flume Java (library used to

develop, test and run code efficiently in parallel pipelines) [22], and Pregel (a large-scale graph processing computational model) [23].

2.2.3 Scheduler

When users submit a job, the Borgmaster keeps the change log persistently in a Paxos-based store [24] and appends the job's tasks to the pending queue. The scheduler operates on the granularity of a task unit; it scans the queue asynchronously and assigns tasks to machines modulated in a round-robin scheme that ensures fairness across users and avoids blocking small tasks behind large ones.

The scheduler algorithm has two parts[16]: (1) feasibility checking finds the machines where a task can run; (2) scoring selects which one of the feasible machines is preferred. In the scoring stage, the criteria were built to minimize the number and priority of preempted tasks by spreading tasks across power and failure domains and packing quality in a mix of high and low tasks onto the same machine. However, in practice, a lower-priority task is often evicted or preempted if a higher-priority task gets assigned to the same resource. Figure 2.2 portrays an overview of the scheduling process.

2.3 Comparison Between Borg Traces V2 and V3

An extensive comparison in the phenomenon, growth and improvement of Borg from trace V2 in 2011 to trace V3 in 2019 has been presented in the literature [15] [25][16][26]. There are two main characteristics to measure and compare CPU and memory resources: requested resource allocation and actual resource usage. The averages have increased over the years on both CPU and memory for the best-effort batch and production-tier jobs. In contrast, resource allocation of free-tier jobs in both scales had a large reduction, while the best-effort batch had a large increment and is about 20% of the cell's overall capacity. [15]. Figure 2.3 and Figure 2.4 also show improvement in the stability of CPU and memory across aggregated time (one-hour averages across millions of machines in 8 cells broken down into tiers.)

As expected, variation in workload across cells and within cells existed. In Figure 2.5 and

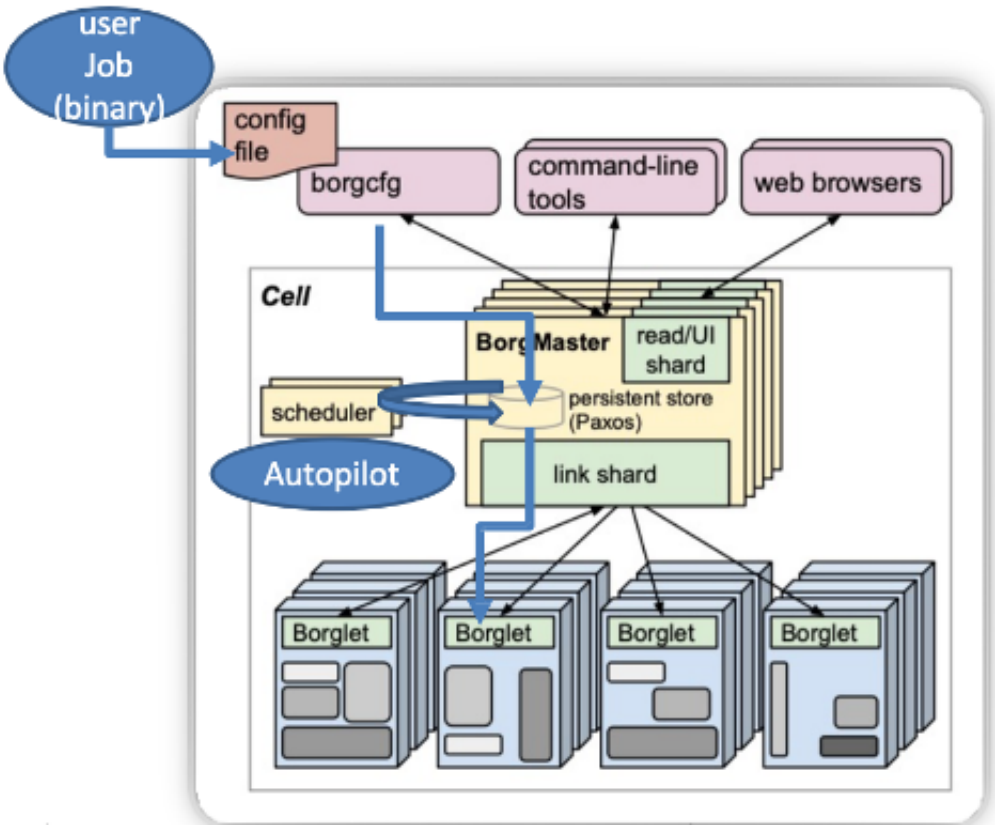

Figure 2.2: An overview of Borg scheduling process (Picture edited from the paper at EuroSys'15 by Verma et al[16]).

Figure 2.6, even though cell *c* has a dominant allocation of jobs in the best-effort tier, around 70% of the average resource usage is consumed by jobs in the production tier. Whereas cell *b* has about 30% average resource allocation of jobs in the best-effort tier, they account for 50% of the average resource usage. Regardless of the internal variations across the 8 cells, every cell in the 2019 dataset has a better resource usage and allocation management than the recorded data from 2011.

In addition, Borg's scheduling workload has increased since 2011. By 2019, the median job arrival rate increased 3.7 times, and the median task scheduling rate increased 3.6 times [15]. However, many scheduling events commit to rescheduling, and the drastic increase in the ratio of the resubmitted tasks rate to the new task rate indicates a "churn" phenomenon in the modern system. The "churn" also suggests Borg's over-commitment of resources to top production jobs during scheduling.

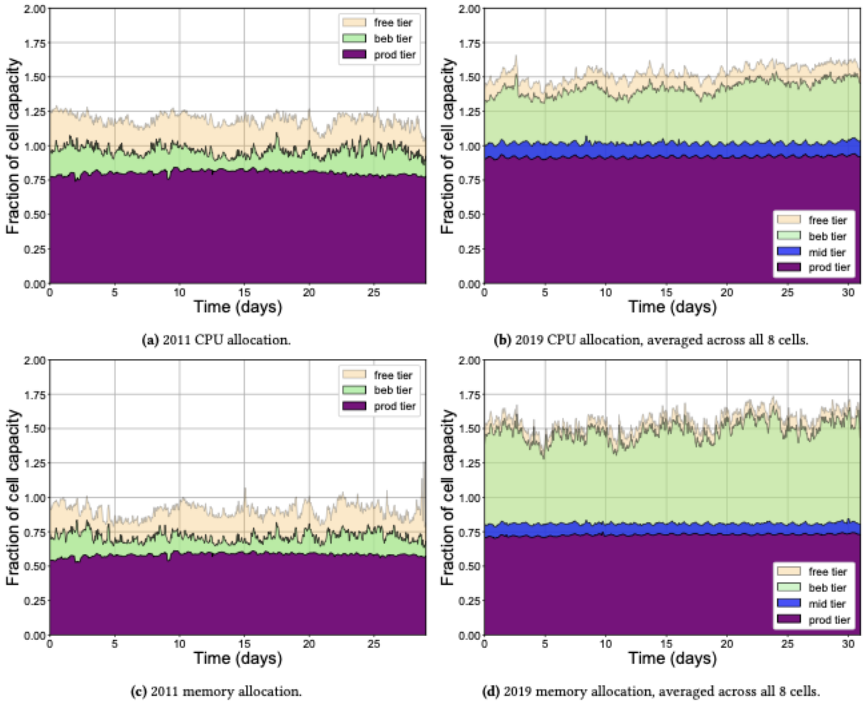

Figure 2.3: The fraction of a cell's total resource capacity **allocated** during each hour-long interval (Picture from paper at EuroSys'20 by Tirmazi et al.[15]).

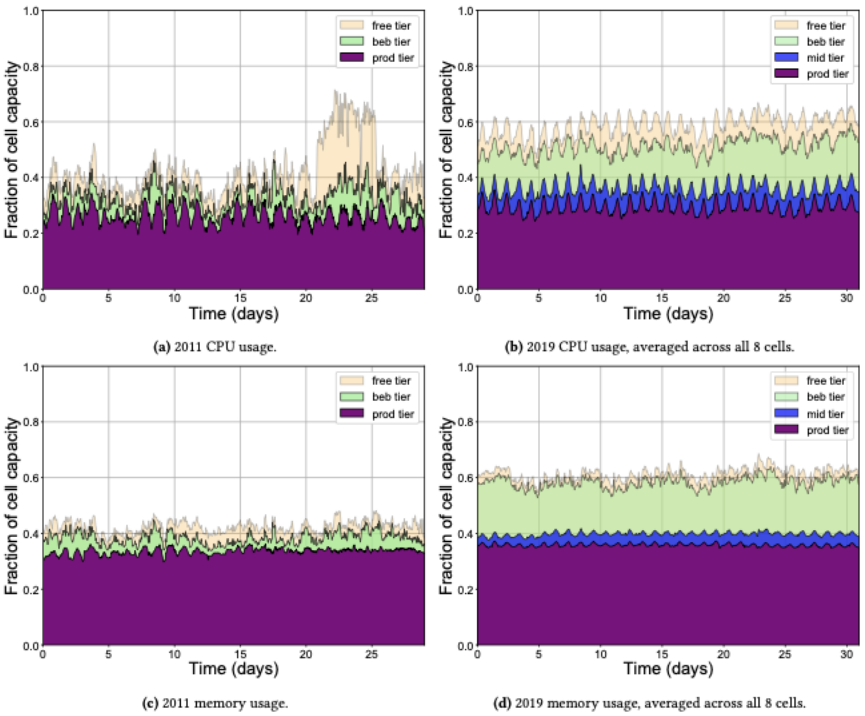

Figure 2.4: The fraction of a cell's total resource capacity **used** during each hour-long interval (Picture from paper at EuroSys'20 by Tirmazi et al.[15]).

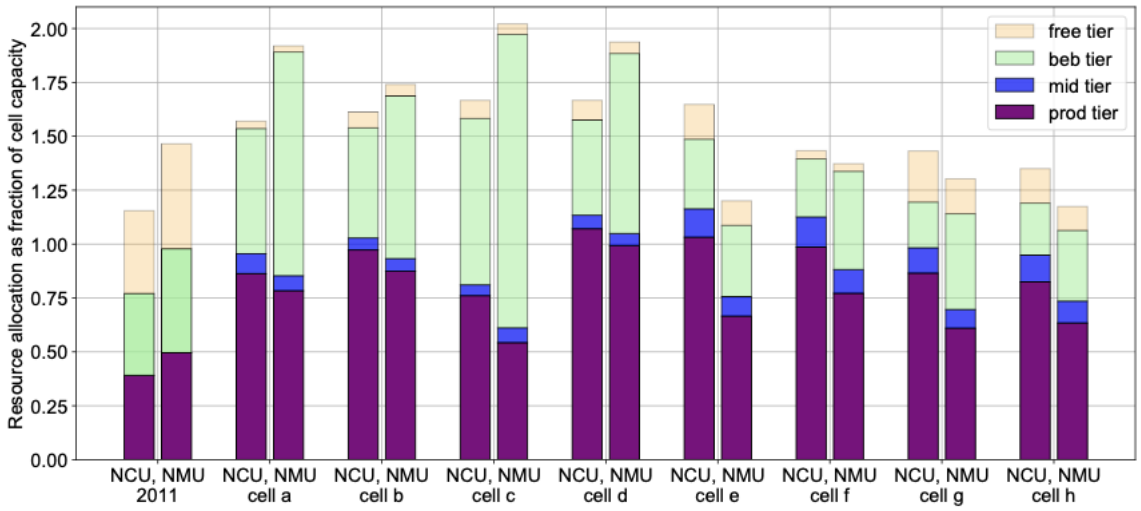

Figure 2.5: Average **allocation** (as a fraction of cell capacity across the entire trace duration) by tier, for 2011 and the 8 cells in the 2019 trace (Picture from the paper at EuroSys'20 by Tirmazi et al.[15]).

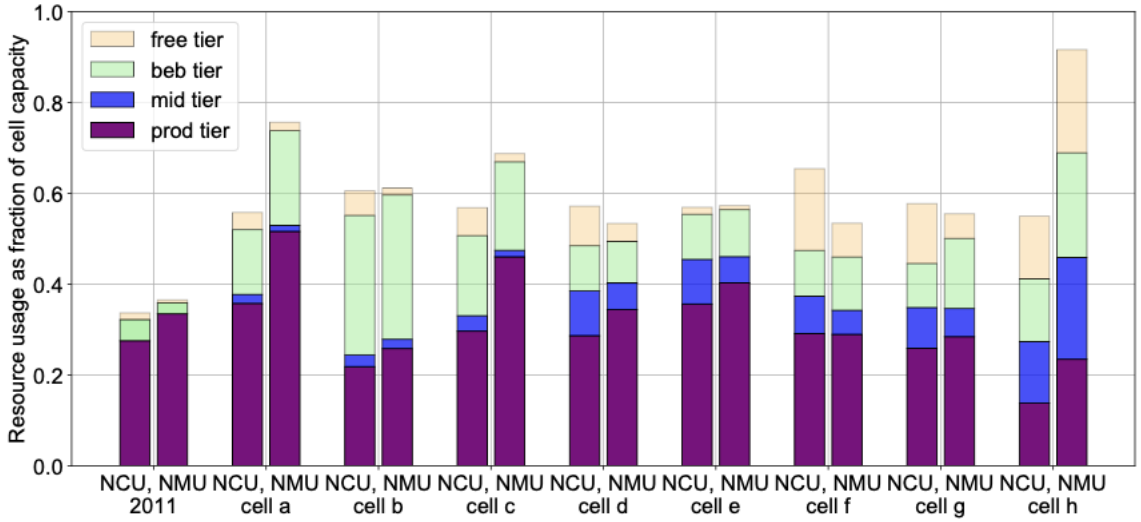

Figure 2.6: Average **utilization** (as a fraction of cell capacity across the entire trace duration) by tier, for 2011 and the 8 cells in the 2019 trace (Picture from the paper at EuroSys'20 by Tirmazi et al.[15]).

3. DATA OVERVIEW

The Google Borg trace V3 is about 2.4TiB compressed and records a wide range of behaviors on machines and their schedulers from 8 different Borg cells for May 2019. Each cell's trace is stored separately in 8 buckets on Google BigQuery objects [27], particularly focusing on resource requests and usage. Due to confidentiality, Google uses obfuscation techniques to protect any sensitive or classified information. Text fields are randomly hashed, numeric resources are linearly transformed, and special values are mapped onto ordered series. However, a thorough understanding of the trace characteristics is held back by such corporate and commercial concerns.

In recent years, some of the most prominent technology companies have released small portions of workload traces that capture the behavior of their system in production: Google opened a repository with three versions (e.g., 7 hr 2011 V1, May 2011 V2, May 2019 V3 [1]) of traces capturing characteristics of the jobs running on Google compute cells managed by the Borg system; Microsoft publicly released five datasets intended to evaluate three major functionalities of Azure (e.g., VM 2017 & 2019, Packing 2020, Function Invocations 2019 & 2021 [28]) for the benefit of the research and academic community; Alibaba also published four versions of traces that capture workload behavior for modern application designs such as micro-services (e.g., VM 2017 & 2018, GPU 2020, Micro-service 2021 [29, 30]) from real production environments.

In this chapter, we present a study the Borg trace V3 and perform an exploratory analysis. We also explain some of the constraints, potential impacts, and adjustments made to adapt to the limited working environment.

3.1 Data Table

Each cell is independently managed by a single Borgmaster and has a corresponding trace organized into five data tables. Each record in the table has a timestamp marked 600 seconds before the beginning of the trace period (recorded in microseconds for consistency.) The accuracy of timestamps is approximately precise, and small disagreements due to clock drift between ma-

chines are tolerated. In the trace, Borg measures and stores memory data by dividing them by the maximum machine memory size across all traces. Borg measures CPU consumption in an internal unit called "Google compute units" (GCU), which represents one CPU-core worth of computation power on a nominal base machine [1]. The trace normalizes GCU with a similar scaling method applied on memory, which refers to as a Normalized Compute Units (NCU). The content of each table is concisely summarized as follows [1]:

1. **Machine attributes:** The table stores key-value pairs describing each machine's characteristics in a cell. The values and names are obfuscated strings intended to describe machines' properties such as the kernel version, external IP address, and other specified constraints. This research does not utilize information inside this table because most values are unrecognizable and infeasible to interpret without internal Google documentation.

2. **Machine events:** This table reflects the normalized physical capacity in each dimension (CPU cores, RAM size) of each machine and their corresponding action events (e.g., Add, Remove, Update) at a timestamp.

3. **Collection events:** The table represents the life cycle of collection instances (jobs or alloc sets). The table includes context-sensitive information but can locate specific collection instances in a cell trace with a unique identifier. The table also describes fields specific to the property for collection instance, such as collection type (a job or alloc set) and vertical scaling type. Vertical scaling is an automatic scheduling service supported by the Autopilot system [31] that can predict and adjust jobs' resource limits based on historical data. The trace indicates if the collection applies no, constrained, or fully automated vertical scaling.

4. **Instance events:** The table provides actions and life cycle information for instances (tasks or allocs). The table indicates resource request and constraint fields that set the limit for the maximum CPU, memory, and machine attributes an instance permitted to use. The table also depicts the behaviors of Borg scheduler over-commitment that often throttle or kill low-priority tasks in preference to maintain good service on high-priority tasks.

5. **Instance usage:** The table records the usage statistics of each task in approximately 5-minutes windows, obtaining the measurements at intervals of one second. Detailed resource usages are captured, including average usage observed within the window, an upper bound of average memory given by the Borglet agent, memory file page cache by the operating system kernel, cycles per instruction, and memory accesses per instruction (i.e., CPU or memory cycles used dividing by the number of executed instructions; only machines equipped with processor performance counters collect those statistical data).

3.2 Data Process Pipeline

Due to the tremendous dataset size stored in Big Query [27], a few processing steps are applied (Figure 3.1) to the original data to accommodate the limited working environment. We deem that the large size of the sample is sufficient to represent the overall trace behaviors and their underlying characteristics.

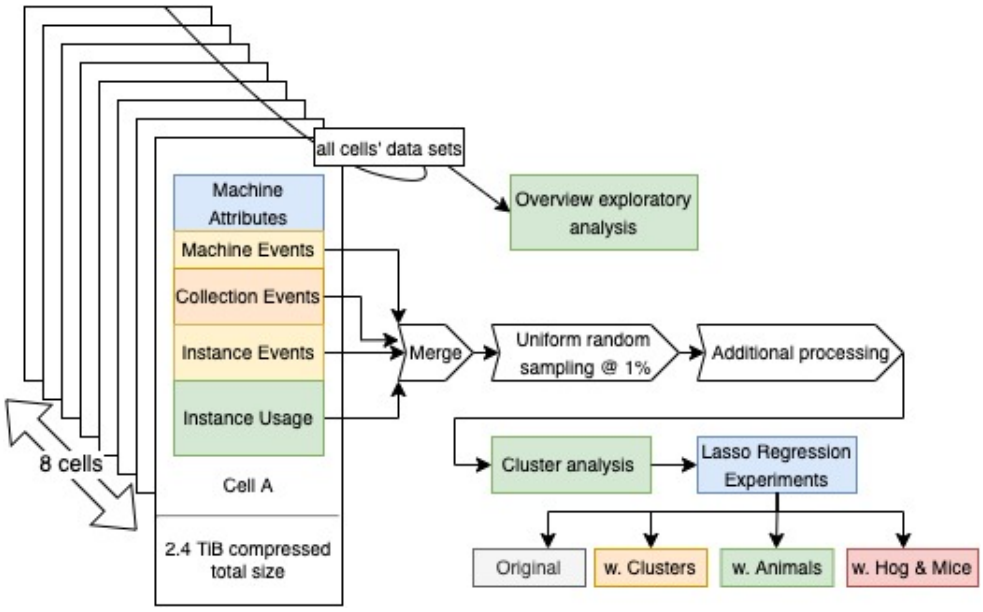

Figure 3.1: An overview of the data process pipeline.

Before sampling, the machine-level variation is preliminarily explored on the original ma-

chine attributes and machine events tables across cells. After that, designated queries are run on Big Query to merge collection events, instance events, and instance usage tables on each job identifier, which is unique to individual cells to obtain information for collection properties, instance requested information, and usage data recorded at run-time. Then a uniform random sampling is applied at 1%, and the sample is fetched for additional data processing and analysis on the local device.

Figures 3.2 and 3.3 summarize the collected sample data. Compared with Figures 2.5 and 2.6, some main characteristics of cells are captured. For example, the best-effort tier jobs in cell c still have about 60% allocation requests and about 40% in resource usage. Some cells have disagreements: for example, the total usage of cell b is surprisingly high compared with other cells. Nevertheless, around 4 million jobs of the sample data should provide enough acknowledgment of the original trace.

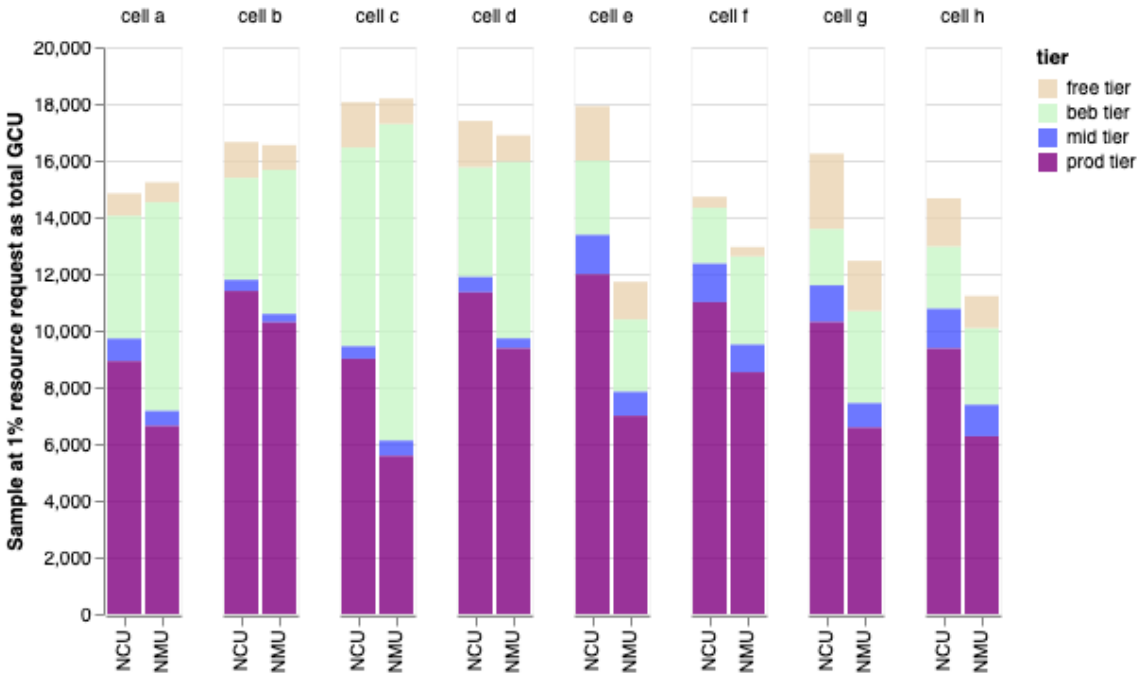

Figure 3.2: Total **requested resources** of the sample as a fraction of 1% cell capacity among sample jobs by tier.

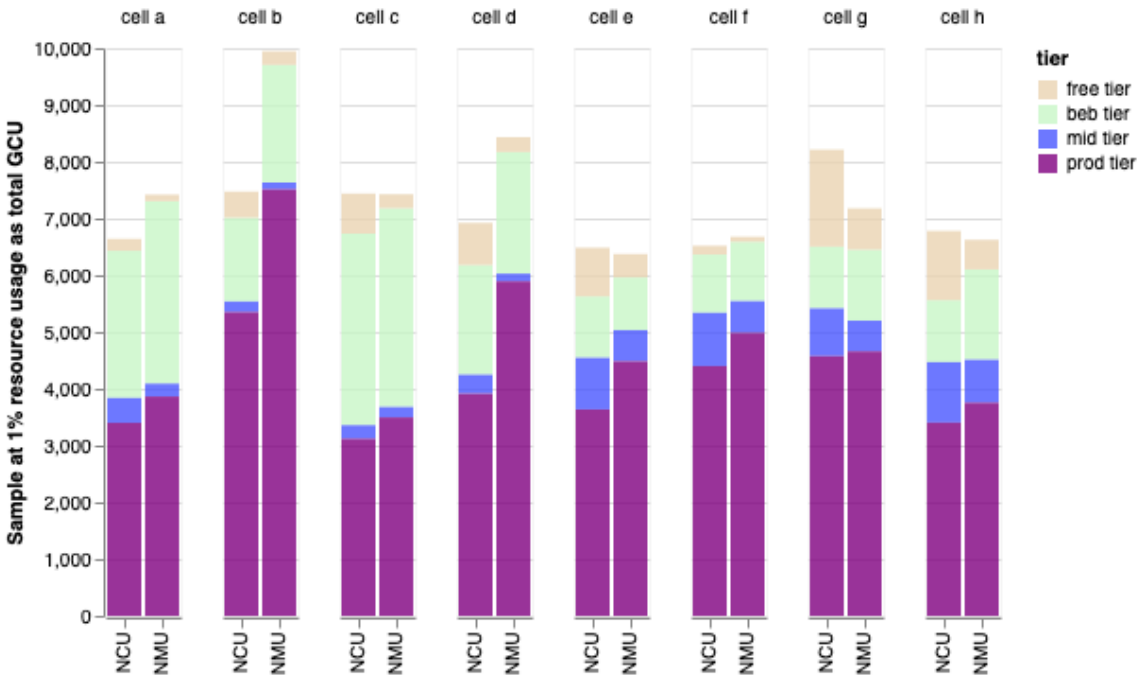

Figure 3.3: Total **resource usage** of the sample as a fraction of 1% cell capacity among sample jobs by tier.

3.3 Data Features Overview

3.3.1 Animals

Previous research discussed a non-negligible phenomenon referred to as "mice and hogs" [15] (or elephant [25]) in workload scheduling. Both CPU and memory resource in usage and allocation follow an extreme heavy-tailed power-law Pareto distribution with the alpha parameter at 0.69 [15]. Log and square root transformations are applied on features to normalized heavy-tailed bias. Figure 3.4 clearly depicts a clear boundary between hog tasks (with heavy CPU usage) and mice ones. We aim to answer whether there exists a necessity to insulate the top 1% from the bottom 99% of jobs. Therefore, segregation labels are assigned to jobs based on their quantiles, and experiments are developed to determine their significance.

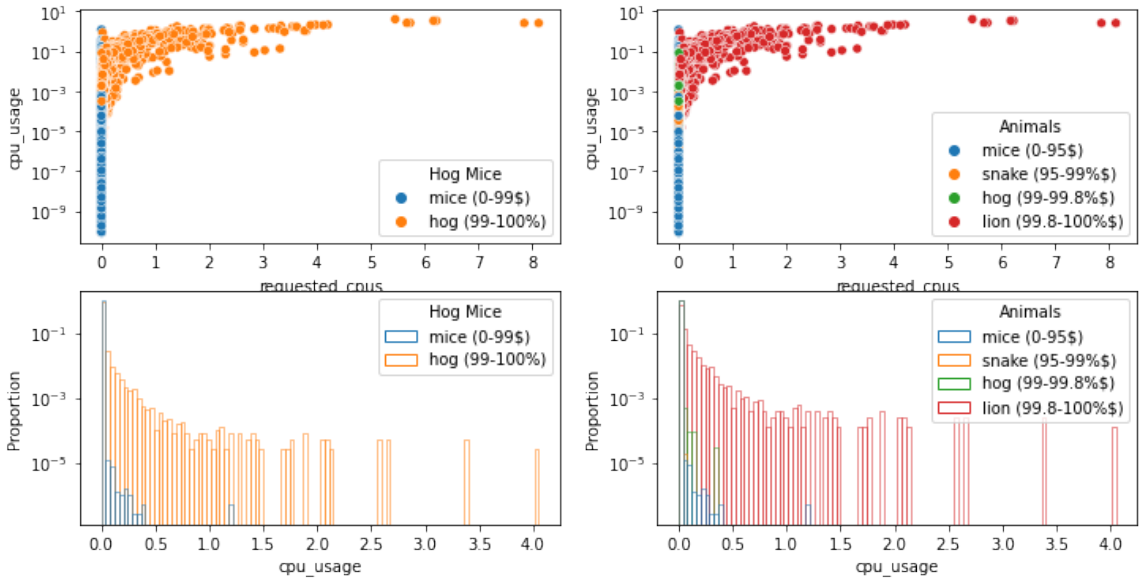

Figure 3.4: The scatter plot (top) and histogram plot (bottom) of the sample at given labels of CPU resource quantiles.

Furthermore, besides hog and mice, two animal categories are subjectively selected (Figure 3.5) based on a primitive intuition of the data distribution: snake and lion. The animals are a more exhaustive classification on higher-level jobs starting above 95% quantiles. Note that production-tier jobs have a ruling population across percentiles.

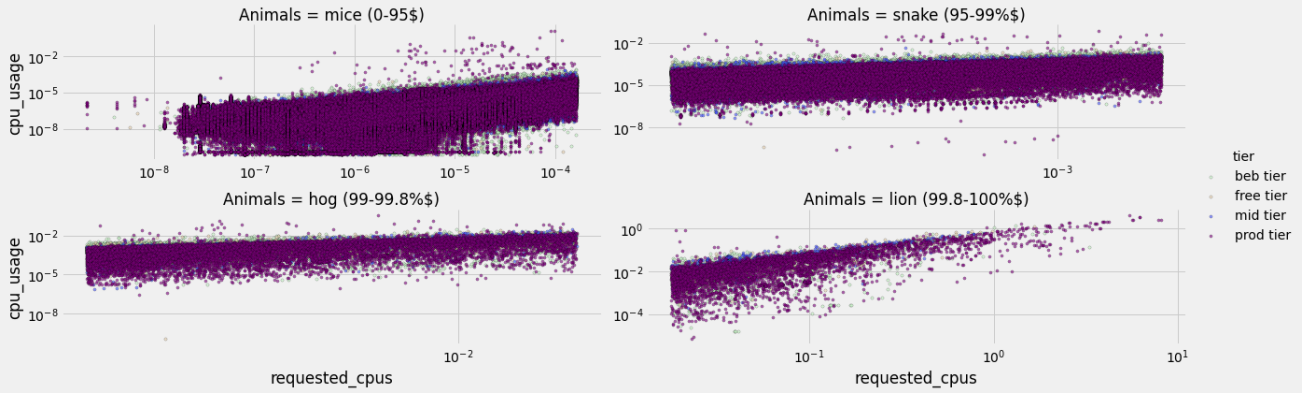

Figure 3.5: The scatter plot of CPU usage vs. request on the sample by the animal label at a log scale. The tier terms (beb, free, mid, prod) are defined in Section 3.3.2.

3.3.2 Handling priority

Every task has a priority level mapping into a set of sorted values, ranging from 0 as a lower priority to 360 or above as a higher priority. The Borg scheduler prioritizes treatment and resource preferences to high-priority tasks over low-priority tasks. Tasks or jobs with similar priority range possess similar properties and are referred to as tiers, in general [1]:

- **Free tier** (priorities <= 99): Jobs with the lowest priorities have no or weak Service Level Objectives (SLOs); they have no expectations regarding obtaining or occupying resources on the machine. Typically, they incur no or little internal charges.

- **Best-effort Batch (beb) tier** (priorities 100 - 115): Jobs are managed by the batch scheduler. They incur low internal charges and have no associated SLOs.

- **Mid-tier** (priorities 116 - 119): Jobs in Mid-tier have lower internal charges and weaker SLOs than jobs in the production tier, and higher internal charges and stronger SLOs than jobs in the free tier.

- **Production tier** (priorities 120 - 359): These are jobs with the highest priorities and availability in the normal setting. Borg schedulers often evict low-priority jobs to transfer storage and networking privilege to high-priority jobs (most times in the production tier) as they expect to receive high-quality services [15]. In particular, the Borg scheduler has a rule-based strategy to prevent latency-sensitive, production-tier tasks from being evicted due to over-allocation at runtime.

- **Monitoring tier** (priorities >= 360): Jobs with the highest priorities are essential to the infrastructure level and intended to monitor the health of critical services. In this work, the monitoring tier is considered a part of the production tier due to its small size. A similar classification has been made in prior analysis work [15].

All tasks or jobs have a scheduling class describing their level of latency-sensitivity in execution. In general, latency-sensitive jobs tend to have higher priorities [1]. Both data collection

events and server instance events impact when a machine schedules a task, and the scheduling class affects the machine-local policy for resource access once the task gets scheduled. The runtime usage and requested events of each instance are used to construct realistic performance benchmarks and help improve scheduling management in the cloud.

3.3.3 Cell Capacity

Each cell may be organized to serve a specialized purpose due to the differences in computing capacity. An underlying pattern on resource demands and allocations may be found if ones dives into the internal variation within cells and machines. However, to the best of our knowledge, no previous work has conducted a detailed study of statistical influences of cells' or machines' variations in Borg datasets. Figure 3.6 describes the variation in the machine shapes across 8 cells. Three kinds of machine configurations occur quite commonly as they can be classified by low, medium, and high CPU capacity. Although more insights may be discovered by comparing and accounting variants at the machine level, this research approximates behaviors in the unit of a cell.

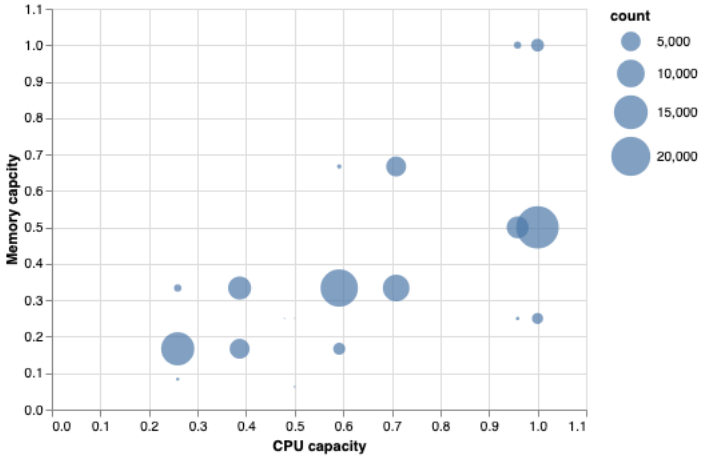

Figure 3.6: Frequency of machine configuration across the 8 cells as a function of CPU and memory (RAM) capacity. The area of each circle is proportional to the number of machines appeared in such configuration.

4. PROPOSED METHODS

This chapter introduces the methods applied to the analysis of of the Borg V3 trace. In general, they are classified into three categories according to their functionality: clustering, regression, and classification.

4.1 Clustering

4.1.1 K-means Clustering

K-means clustering is one of the simplest and most popular unsupervised analytical methods applied in data mining [32]. The unsupervised learning algorithms are capable of capturing underlying patterns and statistical inferences from numeric datasets without labeled or defined outcomes. The objective of the K-means algorithm is to group data points with similar characteristics. K-means can help discover the logistic of allocation of the clusters, which refers to a collection of the aggregated data points. The K-means algorithm starts by randomly picking a given number of centroids. Then, iteratively, the K-means will perform squared Euclidean distance calculations to minimize within-cluster variances and optimize the positions of the centroids until their positions are stable.

4.1.2 K-mode Clustering

K-Prototype clustering is built based on combining K-mode clustering (i.e., another clustering algorithm designed only for categorical features by encoding categorical values into dummy variables and grouping data points with similar modes, instead of clusters' means) and K-means [33]. Since K-means can only have numeric inputs, the K-Prototype is used to observe impacts of categorical features as it is a conventional clustering method suitable for data containing both categorical and numeric inputs.

4.2 Regression

4.2.1 Lasso Regression

Lasso regression, also known as L1 regularization, is an extension of ordinal linear regression that applies a penalty equal to the absolute value of the magnitude of coefficients so that the coefficients of unimportant features shrink down towards a central point [34]. The regularization of Lasso towards variable selection enhances the prediction accuracy, as well as the interpretability of the selected models.

4.2.2 Random Forest Regression

Random forest regression is a supervised learning algorithm that takes advantage of the "wisdom of the crowds". The algorithm builds a collection of trees with specific hyper-parameters that decide the maximum depth level of a tree, the number of constructed trees, and the minimum number of samples at the leaf nodes [35]. Each tree in the forest randomly selects a subset of features as candidates to choose on each level of splits, which prevents trees from overly using the same set of features. Then each tree will form the best tree to separate splits by giving a random sample data from the training set. This step introduces an additional element of randomness to prevent over-fitting the original data. Following such an approach, the algorithm randomly constructs a forest (a set with a predefined number of trees) and asks each tree in the forest to vote based on individual decision criteria. The voting result is averaged across all predictions as an ensemble learning process.

4.3 Classification

4.3.1 XGBoosting Classifier

Boosting is the process of building a strong classifier from a series of weak classifiers. XGBoost takes advantage of boosting that managed both bias and variance aspects under control [36]. The new model on each iteration is adjusted to correct the errors on previous models. The ideal model is formed when no more improvements on accuracy can be progressed when adding new models sequentially. XGBoost is an extension to gradient boosted decision trees (GBM) with

regularization and tree ensemble models to smooth the final learned; practically, XGBoost offers better execution speed and significant improvement in model performance than GBM [36].

4.3.2 Random Forest Classifier

The random forest classification algorithm works in the same way as the random forest regression, except the output response is categorical. The algorithm uses bagging and features randomness on each tree to introduce randomness and prevent overloading. As the algorithm progresses in each tree construction, the unbiased estimate of generalization error is minimized. Moreover, the result decided by the committee of the uncorrelated forest of trees outperforms that of any individual tree [37].

5. RESULTS

This chapter describes the results of our investigation, a quantitative analysis of Borg's policy decision, and an examination of imbalanced operational scheduling experience on a different tier. Each method is applied for a different exploratory purpose based on its classification: clustering methods to observe any underlying patterns occurring in the current workload, regression methods to determine the necessity of isolating workloads for better resource prediction, and classification methods to reverse-engineer the significant factors driving the scheduling adjustment carried out by the Autopilot system.

5.1 Investigation of Underlying Patterns

The K-means clustering can observe the natural aggregation of data points in the Borg scheduling workload. A controlled experiment is then conducted to delve into the impacts of tiers on jobs. Since the K-means algorithm is sensitive towards numeric size and each feature follows a heavy-tailed Pareto distribution [15], the log and square root transformations are applied to adjust the scaling.

In the first set, the scaled priority level is not included. A good model fit is found at 5 clusters. The result has one large group that captures a prevalent number of data points in free, mid, and production tiers that we call the *popular set*. One medium-sized group captures the popular aggregation of latency-sensitive jobs inside the best-effort tier and some extreme jobs in the production tier. The rest of the small groups capture various outliers or influencers. As the number of clusters increases, the medium and large groups remain stable, and small groups are broken down based on cycles-per-instruction services. The phenomenon describes the natural tendency of mice and hog problems [15][25].

When considering the priority level in the clustering process, the model has a good fit at 4 clusters. The support of tier management is significant, but it is possible that the unusual distribution of priority levels may be the deciding factor, as shown in Figure 5.1: a majority of mid-

tier jobs is aggregated into the "normal" group, and the free-tier jobs split into the "normal" and "marginal" groups; both the best-effort tier and production tier jobs are evenly distributed into four groups, with a large population of high-quality demands jobs clustered into the "high-performance" group. All jobs in different tiers have a small set of marginal data points aggregated into the "marginal" and "extreme" group. With the support of the priority level, it is worth mentioning that each group has approximately the same proportion of jobs from different animal labels, implying that Borg uniformly manages assignments across quantiles.

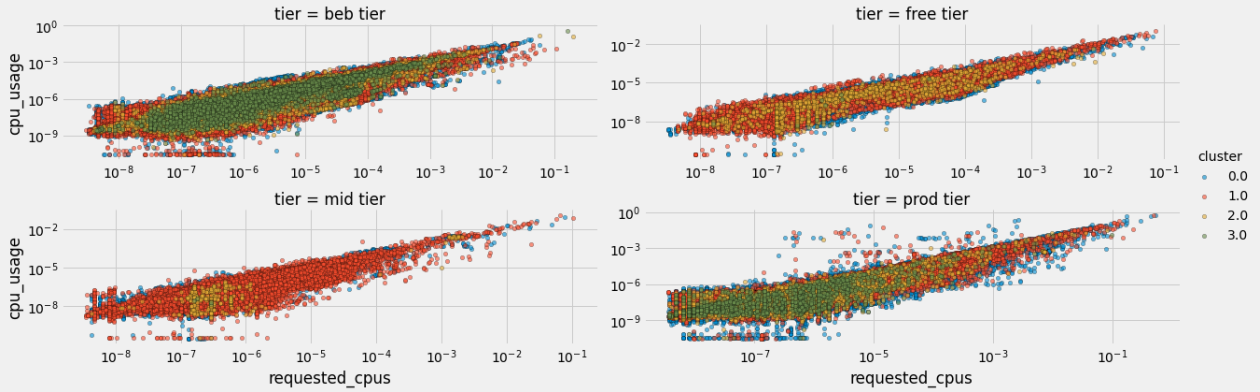

Figure 5.1: K-means results by tiers whose jobs are color-coded in different clustered groups. The red is the "normal" group, the yellow is the "marginal" group, the blue is the "extreme" group, and the green is the "high-performance" group.

Building upon K-means, the K-prototype model accounts for categorical features as well. However, the model does not provide any more insights or changes. The underlying pattern depicted by the numeric features in K-means occupies the leading aggregation in K-prototype. The K-prototype result shows that tiers (a categorical form of the priority level), scheduling class, vertical scaling service, and scheduler types (batch or not) do not contribute much to the natural aggregation in the Borg scheduling workload.

5.2 The Impact of Insulation on Resource Predictability

The high degree of heterogeneity and dynamism in the workloads cause difficulty in making accurate predictions for resource demands. The goal is to verify the statement in the literature[15]

positing that the insulation of mice and hog tasks is needed for making better resource usage predictions for future workloads. The given labels in the model symbolize the segregation of data points. A set of experiments were conducted, using the basic Lasso, as the main observation reference towards the impact of those labels. In addition to the mice and hogs labels, additional animal labels, defined by quantiles and the clustering groups created by the K-means, are added into the model to observe if different kinds of segregation may be intuitive and critical. The random forest regression is also applied to understand the main factors correlated with resource usage.

Table 5.1 summarizes the performance of five cases. The basic Lasso model describes that the priority level and total lifetime of jobs that exist in workloads are not significant predictors. When considering hogs and mice, there is a constant slight improvement on the Lasso models. Moreover, the Lasso model with animal labels has better improvement than hogs and mice in explaining the variation of data sets. However, we believed that the classification of animals may correlate with the impacts of outliers (the group of latency-sensitive and expensive services). The Lasso result of cluster labels is very similar to Lasso with hogs and mice and implies that the natural clustering may contain inconspicuous information of that phenomenon. Lastly, the random forest regression depicts the non-parametric pattern and indicates that animals serve as an important factor to improve resource usage prediction.

Table 5.1: 10-folds R^2 score on five experiment conditions.

Method	**Training Score**	**Testing Score**
Lasso	83.66	85.17
Lasso w. Hog & Mice	84.79	86.12
Lasso w. Animals	88.57	89.57
Lasso w. Clusters	84.75	86.05
Random Forest Regression	94.36	94.36

5.3 Reverse Engineering on Autopilot System

Previous research states that configuration tools such as the latest Autopilot, work more efficiently than user manually defined tasks in allocating resources[15]. We apply reverse engineering techniques to discover possible major contributors to Autopilot's prediction process success. The XGBoosting and random forest classifiers are fit to predict whether the Autopilot manages a job completely, with some special constraints or without. Both classification methods had great results in the 10-folds validation. On average, the XGBoosting classifier achieved 94% and 89% training and testing accuracy, while the random classifier forest had both 84% training and testing accuracy scores.

Both classification methods indicate that scheduling class, the amount of requested resources, and usage information are the deciding factors Autopilot considers in using historical or similar jobs to make resource predictions. In Figure 5.2, the random forest classifier decision considers scheduler types, tier types, and scheduling classes more important than those of the XGBoost classifier. In reverse thinking, the XGBoosting classifier makes better predictions than the random forest classifier and deduces that those factors may play more significant roles in Autopilot's decision.

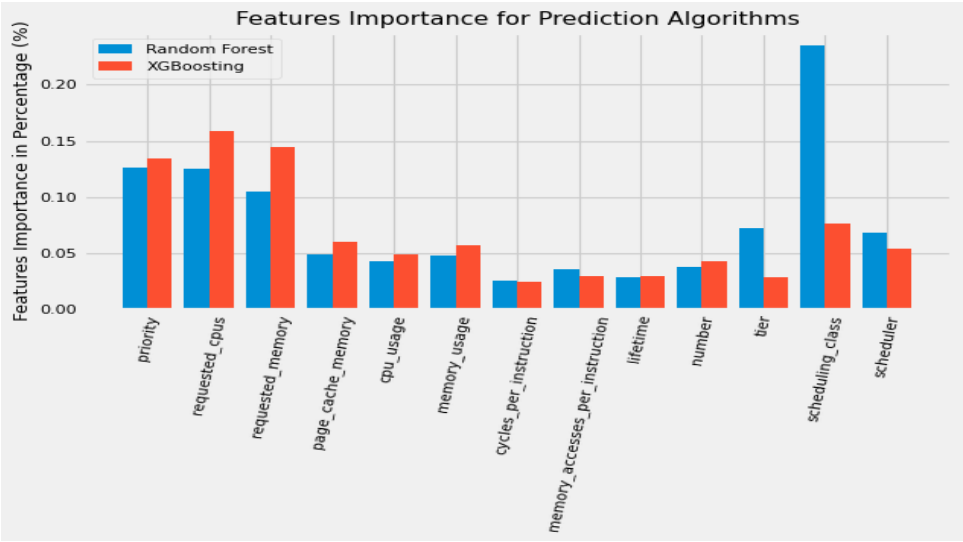

Figure 5.2: Variable importance on the XGBoosting and random forest classifier.

6. CONCLUSION

During the data exploration stage, we found that, even with sampling at a 1% ratio, big variations in the behaviors across cells are consistent with original data. The disparity between big jobs ("hogs") and small jobs ("mice") is still extreme.

The investigation using K-means and K-prototype clustering without priority levels underscores the findings from previous studies [15][25], the natural phenomenon of mice and hogs, even with 1% sample data. In contrast, Kmeans aggregation with priority levels describes a different set of classifications on jobs, namely: normal, marginal, high-performance, and extreme. Although the unusual distribution of priority levels is a concern in biasedly influencing the clustering result, those four groups imply correlation to the animal labels defined by jobs' quantiles and strong support to insulation.

In regards to the predictability of resource usage, measured by R^2, the results suggest a necessity in the insulation of hogs and mice, or perhaps, more exhaustive segregation such as animals which is intuitive to capture the stage-wise difference between high demand and regular jobs. Additional labeling can further improve the pursuit of optimizing resource allocation and predicting resource usage in dynamic workloads with a high degree of heterogeneity.

Last but not least, observing that current resource prediction algorithms (such as Autopilot) perform more efficiently than users manually defining requirements and procedures, we use reverse engineering with classification methods to identify the preferences of the Autopilot decision process. When Autopilot uses historical data as reference, the scheduler weighs and considers a set of attributes of jobs in rank, such as past resource requests and past actual usage information.

We believe these exploratory experiments on the V3 trace can motivate others to develop new ideas and methods to optimize the management and scheduling of cloud workloads.

REFERENCES

[1] "Borg cluster traces from Google." `https://github.com/google/cluster-data`, 2020. Last visited 3/3/2022.

[2] J. C. Mogul, P. Mahadevan, C. Diot, J. Wilkes, P. Gill, and A. Vahdat, "Data-driven networking research: models for academic collaboration with industry (a google point of view)," *ACM SIGCOMM Computer Communication Review*, vol. 51, no. 4, pp. 47–49, 2021.

[3] L. A. Barroso, U. Hölzle, P. Ranganathan, and M. Martonosi, *The Datacenter as a Computer: Designing Warehouse-Scale Machines, Third Edition*. 2018.

[4] K. Wang, Q. Zhou, S. Guo, and J. Luo, "Cluster frameworks for efficient scheduling and resource allocation in data center networks: A survey," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 3560–3580, 2018.

[5] A. Benoit, Ü. V. Çatalyürek, Y. Robert, and E. Saule, "A survey of pipelined workflow scheduling: Models and algorithms," *ACM Computing Surveys (CSUR)*, vol. 45, no. 4, pp. 1–36, 2013.

[6] S. Sherman, F. Baskett III, and J. C. Browne, "Trace-driven modeling and analysis of cpu scheduling in a multiprogramming system," *Communications of the ACM*, vol. 15, no. 12, pp. 1063–1069, 1972.

[7] S. R. Goldschmidt and J. L. Hennessy, "The accuracy of trace-driven simulations of multiprocessors," *ACM SIGMETRICS Performance Evaluation Review*, vol. 21, no. 1, pp. 146–157, 1993.

[8] A. Rico, A. Duran, F. Cabarcas, Y. Etsion, A. Ramirez, and M. Valero, "Trace-driven simulation of multithreaded applications," in *(IEEE ISPASS) IEEE International Symposium on Performance Analysis of Systems and Software*, pp. 87–96, IEEE, 2011.

[9] S. K. Setia, "Trace-driven analysis of migration-based gang scheduling policies for parallel computers," in *Proceedings of the 1997 International Conference on Parallel Processing (Cat. No. 97TB100162)*, pp. 489–492, IEEE, 1997.

[10] R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella, "Multi-resource packing for cluster schedulers," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 455–466, 2014.

[11] O. Hadary, L. Marshall, I. Menache, A. Pan, E. E. Greeff, D. Dion, S. Dorminey, S. Joshi, Y. Chen, M. Russinovich, *et al.*, "Protean:{VM} allocation service at scale," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pp. 845–861, 2020.

[12] I. Calciu, M. T. Imran, I. Puddu, S. Kashyap, H. A. Maruf, O. Mutlu, and A. Kolli, "Rethinking software runtimes for disaggregated memory," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 79–92, 2021.

[13] Y. Feng, Z. Liu, Y. Zhao, T. Jin, Y. Wu, Y. Zhang, J. Cheng, C. Li, and T. Guan, "Scaling large production clusters with partitioned synchronization," in *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pp. 81–97, 2021.

[14] Y. Chen, A. S. Ganapathi, R. Griffith, and R. H. Katz, "Analysis and lessons from a publicly available google cluster trace," *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2010-95*, vol. 94, 2010.

[15] M. Tirmazi, A. Barker, N. Deng, M. E. Haque, Z. G. Qin, S. Hand, M. Harchol-Balter, and J. Wilkes, "Borg: the next generation," in *EuroSys'20*, (Heraklion, Crete), 2020.

[16] A. Verma, L. Pedrosa, M. R. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at Google with Borg," in *Proceedings of the European Conference on Computer Systems (EuroSys)*, (Bordeaux, France), 2015.

[17] C. Reiss, J. Wilkes, and J. L. Hellerstein, "Google cluster-usage traces: format + schema," tech. rep., Google, Mountain View, CA, November 2011. Available at https://github.com/google/clusterdat.

[18] P. B. Menage, "Adding generic process containers to the linux kernel," in *Proceedings of the Linux symposium*, vol. 2, pp. 45–57, Citeseer, 2007.

[19] M. Alam, K. A. Shakil, and S. Sethi, "Analysis and clustering of workload in google cluster trace based on resource usage," 2015.

[20] M. Baker and J. Ousterhout, "Availability in the sprite distributed file system," *SIGOPS Oper. Syst. Rev.*, vol. 25, p. 95–98, apr 1991.

[21] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, p. 107–113, jan 2008.

[22] C. Chambers, A. Raniwala, F. Perry, S. Adams, R. Henry, R. Bradshaw, and Nathan, "Flumejava: Easy, efficient data-parallel pipelines," in *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, (2 Penn Plaza, Suite 701 New York, NY 10121-0701), pp. 363–375, 2010.

[23] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: a system for large-scale graph processing," in *Proceedings of the 2010 international conference on Management of data*, (New York, NY, USA), pp. 135–146, 2010.

[24] L. Lamport, "The part-time parliament," *ACM Trans. Comput. Syst.*, vol. 16, p. 133–169, may 1998.

[25] O. Abdul-Rahman and K. Aida, "Towards understanding the usage behavior of google cloud users: The mice and elephants phenomenon," *2014 IEEE 6th International Conference on Cloud Computing Technology and Science*, pp. 272–277, 2014.

[26] S. Di, D. Kondo, and F. Cappello, "Characterizing cloud applications on a google data center," *2013 42nd International Conference on Parallel Processing*, pp. 468–473, 2013.

[27] Google, "Bigquery documentation at google cloud platform." `https://cloud.google.com/bigquery/docs`. Last visited 3/3/22.

[28] O. Hadary, L. Marshall, I. Menache, A. Pan, E. E. Greeff, D. Dion, S. Dorminey, S. Joshi, Y. Chen, M. Russinovich, and T. Moscibroda, "Protean: VM allocation service at scale," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pp. 845–861, USENIX Association, Nov. 2020.

[29] W. Chen, K. Ye, Y. Wang, G. Xu, and C.-Z. Xu, "How does the workload look like in production cloud? analysis and clustering of workloads on alibaba cluster trace," in *2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)*, pp. 102–109, 2018.

[30] S. Luo, H. Xu, C. Lu, K. Ye, G. Xu, L. Zhang, Y. Ding, J. He, and C. Xu, "Characterizing microservice dependency and performance: Alibaba trace analysis," in *Proceedings of the ACM Symposium on Cloud Computing*, pp. 412–426, 2021.

[31] K. Rzadca, P. Findeisen, J. Swiderski, P. Zych, P. Broniek, J. Kusmierek, P. Nowak, B. Strack, P. Witusowski, S. Hand, *et al.*, "Autopilot: workload autoscaling at google," in *Proceedings of the Fifteenth European Conference on Computer Systems*, pp. 1–16, 2020.

[32] A. Likas, N. Vlassis, and J. J. Verbeek, "The global k-means clustering algorithm," *Pattern Recognition*, vol. 36, no. 2, pp. 451–461, 2003. Biometrics.

[33] J. Ji, T. Bai, C. Zhou, C. Ma, and Z. Wang, "An improved k-prototypes clustering algorithm for mixed numeric and categorical data," *Neurocomputing*, vol. 120, pp. 590–596, 2013.

[34] J. Ranstam and J. Cook, "Lasso regression," *Journal of British Surgery*, vol. 105, no. 10, pp. 1348–1348, 2018.

[35] M. R. Segal, "Machine learning benchmarks and random forest regression," 2004.

[36] T. Chen, T. He, M. Benesty, V. Khotilovich, Y. Tang, H. Cho, K. Chen, *et al.*, "Xgboost: extreme gradient boosting," *R package version 0.4-2*, vol. 1, no. 4, pp. 1–4, 2015.

[37] M. Pal, "Random forest classifier for remote sensing classification," *International journal of remote sensing*, vol. 26, no. 1, pp. 217–222, 2005.