Seagull: An Infrastructure for Load Prediction and Optimized Resource Allocation

Technical Report
June, 2020

Olga Poppe, Tayo Amuneke, Dalitso Banda, Aritra De, Ari Green, Manon Knoertzer, Ehi Nosakhare, Karthik Rajendran, Deepak Shankargouda, Meina Wang, Alan Au, Carlo Curino, Qun Guo, Alekh Jindal, Ajay Kalhan, Morgan Oslake, Sonia Parchani, Vijay Ramani, Raj Sellappan, Saikat Sen, Sheetal Shrotri, Soundararajan Srinivasan, Ping Xia, Shize Xu, Alicia Yang, and Yiwen Zhu

Microsoft Corporation, One Microsoft Way, Redmond, WA 98052
firstname.lastname@microsoft.com

Contents

Abstract

Microsoft Azure is dedicated to guarantee high quality of service to its customers, in particular, during periods of high customer activity, while controlling cost. We employ a Data Science (DS) driven solution to predict user load and leverage these predictions to optimize our services in various ways such as: 1) scheduling maintenance operations to minimize interference, and 2) auto-scaling resources based on predicted load for millions of databases.

To enable these optimizations, we built SEAGULL, an infrastructure that processes per-server telemetry, validates the data, trains and deploys ML models. The models are used to predict customer load per server (24h into the future), and optimize service operations. SEAGULL continually re-evaluates accuracy of model predictions, fallback to previously known good models and triggers alerts as appropriate. We deployed this infrastructure in production for PostgreSQL and MySQL servers across all Azure regions, and applied it to the problem of scheduling database backups during low-load time. This minimizes interference with user-induced load and improves customer experience.

1 Introduction

Microsoft Azure, Google Cloud Platform, Amazon Web Services, and Rackspace Cloud Servers are the leading cloud service providers that continuously expand cloud computing capabilities [30]. They aim to guarantee high quality of service to their customers, while controlling operating costs [41]. Achieving these conflicting goals manually is labor intensive, time consuming, error prone, and not scalable. Thus, cloud service providers shift towards automatically managed data services on the cloud. To this end, machine learning techniques are becoming increasingly popular to predict resource demand and leverage these predictions to automatically optimize resource allocation [16].

Motivating Example 1: ***Backup Scheduling***. Backups of databases are currently scheduled by an automated workflow that does not take typical customer activity patterns into account. Therefore, backups often collide with peaks of customer activity resulting in inevitable competition for resources and poor quality of service during backup windows. To solve this problem currently, an engineer plots the customer load per database per week and manually sets the backup window during low customer activity. However, this solution is time-consuming, error-prone, and it does not scale. Also, customer activity varies from week to week depending on her business rhythm. More recently, customers can select a backup window themselves. However, they may not know the best time to run a backup. Instead of these manual solutions, ML models could be deployed to predict customer load. These predictions could then be leveraged to schedule backups and other system maintenance operations during expected low customer activity.

Motivating Example 2: ***Preemptive Auto-Scale***. SQL databases are offered as provisioned [5] or serverless compute [6]. Provisioned compute means that the amount of allocated resources is fixed and does not change unless the customer manually updates it. It is inherently hard for customers to pick the right amount of resources upfront since workloads often change over time. To overcome the limitations of provisioned compute, serverless offering automatically scales compute based on workload demand. However, current resource scaling is reactive. For example, a paused database

is automatically resumed when the next login or other customer activity occurs. Such reactive auto-scale of resources inevitably introduces delays in compute warm-up after idle periods. Based on expected resource demand, we aim to extend current reactive auto-scale mechanisms with predictive policies to reduce this delay and make serverless compute more suitable for time-critical applications.

Challenges. While building SEAGULL infrastructure, we tackled the following open challenges.

• *Design of an end-to-end generic infrastructure* that predicts resource utilization and leverages these predictions to optimize resource allocation. This infrastructure must be designed in a modular way to be maintainable and applicable to different Azure system components in various projects aiming to, for example, optimize scheduling of system maintenance tasks, load balancing, tenant placement, and ultimately enable demand-driven auto-scale of resources. This infrastructure must build upon and be seamlessly integrated into the current system of Azure products and services.

• *Implementation and deployment of this infrastructure to production worldwide* to solve one concrete instance of the generic load prediction problem. Namely, we aim to predict customer CPU load and schedule full backups such that these backups do not interfere with peak customer activity. This infrastructure must be scalable to all PostgreSQL and MySQL servers across all Azure regions.

• *Accurate yet efficient customer low load prediction* for optimized backup scheduling. This challenge includes choice of an ML model that finds the middle ground between accuracy and scalability. In addition, prediction accuracy must be redefined to focus on predicting the lowest valley in customer CPU load that is long enough to fit a full backup of a server of its backup day. General load prediction per server per day is less critical for backup scheduling use case.

State-of-the-Art Approaches. While systems for ML were proposed in the past, most of them lack easy integration with Azure compute [10, 13, 17, 18, 25, 32]. We also considered leveraging the model-serving infrastructure Resource Central [16]. However, at that time Azure ML [4] provided support and integration for a broader set of modeling and tracking tools. Thus, we built the SEAGULL infrastructure using the functionality of Azure ML.

While time series forecast in general and load prediction in particular are well studied topics, none of the state-of-the-art approaches focused on predicting the lowest valley in customer CPU load for optimized backup scheduling. Instead, existing approaches focus on, for example, idle time detection for predictive resource provisioning [29, 41], VM workload prediction for dynamic VM allocation [15, 16], and demand-driven auto-scale of resources [21, 22, 23, 24, 38, 39, 43]. Thus, these approaches do not tackle the unique challenges of low load prediction for optimized backup scheduling described above. In particular, they neither define the accuracy of low load prediction, nor compare several ML models with respect to low load prediction.

Proposed Solution. We built the SEAGULL infrastructure that deploys ML techniques to predict resource utilization and leverages these predictions for optimized resource allocation. Azure ML pipeline is the core component of this infrastructure. This pipeline consumes the load per Azure region per week, validates this data, extracts features, trains an ML model, deploys this model to a REST endpoint, tracks the versions of all deployed models, predicts the load one day ahead, and evaluates the accuracy of these predictions. This core component can be re-used for load prediction and optimization of any Azure system components (databases, servers, VMs, nodes of a

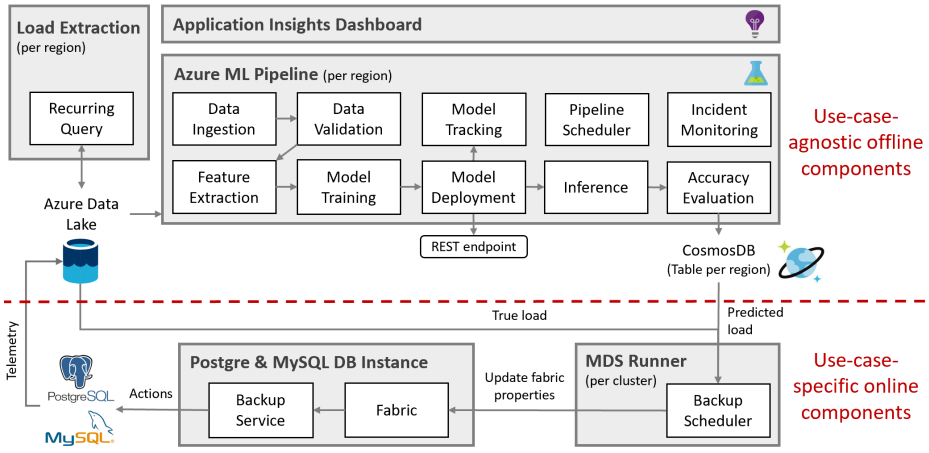


Figure 1: SEAGULL infrastructure

cluster, etc.). We implemented and deployed the SEAGULL infrastructure to production to schedule backups of PostgreSQL and MySQL servers during time intervals of expected low customer activity. We classified the servers based on their typical customer activity patterns and concluded that majority of servers either have stable load or follow a daily or a weekly pattern. Thus, the load per server on previous (equivalent) day can serve as strong predictor of the load per server. This heuristic is called persistent forecast. In our case, it correctly predicted the lowest load window per server on its backup day in 96% of cases. Our SEAGULL approach reduced the number of busy servers for which backups interfere with peak customer activity by half.

Contributions. Our SEAGULL approach features the following key innovations.

• We designed and implemented end-to-end SEAGULL infrastructure for load prediction (Figure 1). It consists of the use-case-agnostic offline and use-case-specific online components. Given load patterns evolve over time, the use-case-agnostic components automatically detect these data drifts, retrain the model, and notify about these changes. We describe one example of the use-case-specific components that leverage predicted load to optimize backup scheduling of PostgreSQL and MySQL servers in all Azure regions. Other use cases are described in Appendix A.

• We conducted comprehensive analysis to classify PostgreSQL and MySQL servers into homogeneous groups based on their lifespan and typical customer activity patterns. Based on this classification, we concluded that majority of servers are either stable or follow a daily or a weekly pattern. Therefore, their load can be accurately predicted using persistent forecast heuristic. Less then 5% of servers are neither stable nor follow a strong business pattern. Hence, their load is harder to predict with high accuracy.

• We defined the accuracy of low load window prediction per server on its backup day as the combination of two metrics. One, the lowest load window is chosen correctly if there is no other window that is long enough to fit a full backup and has significantly lower average user CPU load. Two, the load during a lowest load window is predicted accurately if majority of predicted data points are within a tight acceptable error bound of their respective true data points.

• We applied several ML models commonly used for time series prediction (GluonTS [9], Nim-

busML [11], Prophet [12], and ARIMA [2]) to predict low load of unstable servers that do not follow a pattern that can be recognized by persistent forecast. We compared these models with respect to accuracy and scalability on real production data during one month in four Azure regions. Surprisingly, the accuracy of ML models is not significantly higher than the accuracy of persistent forecast. Thus, we deployed persistent forecast based on previous day to predict low load for all servers.

Outline. This paper is organized as follows. We present the SEAGULL infrastructure in Section 2 and classify the servers in Section 3. Section 4 defines low load prediction accuracy, while Section 5 compares the ML models with respect to their accuracy and scalability. We summarize related work in Section 6 and conclude the paper in Section 7.

2 Seagull Approach in a Nutshell

Load Prediction Problem. Given prior load, our SEAGULL approach aims to predict the future load and utilize these predictions for optimized resource allocation. Backup scheduling during the time intervals of expected low customer activity is an instance of this generic load prediction problem. Exact problem statement is in Section 4.2.

SEAGULL Infrastructure consists of the use-case-agnostic offline and use-case-specific online components (Figure 1). To keep the discussion focused in this paper, we illustrate how this infrastructure is utilized for backup scheduling of PostgreSQL and MySQL servers in production (Example 1 in Section 1). Other use cases that are not deployed to production yet (including preemptive auto-scale of resources, Example 2 in Section 1) are described in Appendix A. Below, we provide an overview of the key components and refer the reader to in-depth analysis in the following sections.

Use-Case-Agnostic Offline Components consume the load per system component (e.g., database, server, VM) and apply ML models to predict future load of this component. While these components can be reused for several use cases, they often have to be adjusted to a particular use case as illustrated by the backup scheduling example below.

• ***Load Extraction Module*** is implemented as a recurring query that extracts the load and other relevant attributes from raw telemetry and stores this data in Azure Data Lake Store [3]. These files are input to the pipeline.

For the backup scheduling use case, we have selected the average customer CPU load percentage per five minutes as an indicator of customer activity. Customer CPU load does not include CPU load of system maintenance such as backups, updates, and statistics refresh. In addition to customer CPU load, other measures of customer activity (memory, I/O, disk, number of connections, etc.) can be added in the future to make a more accurate prediction of low customer activity. In the rest of this paper, customer CPU load percentage per server is referred to as load per server for readability. Servers are typically due for full backup at least once a week, so the load extraction query runs once a week per region. It extracts the load and default backup window per server and stores one file per week and region.

• ***Azure ML Pipeline*** is the core component of the SEAGULL infrastructure. It is built using the functionality of Azure Machine Learning [4] that facilitates end-to-end machine learning life cycle.

This pipeline consumes the load, validates it, extracts features, trains a model, deploys the model, and makes it accessible through a REST endpoint. The pipeline tracks the versions of deployed models, performs inference, and evaluates the accuracy of predictions. Results are stored in Cosmos DB [7], globally distributed and highly available database service. Based on predicted load, resource allocation can be optimized in various ways.

In our use case scenario, the ML predictions are input to the backup scheduling algorithm. A run of the Azure ML pipeline is also scheduled once a week per region because of the typical weekly backup schedule of the servers.

Due to space limitations in this paper, we only describe the most interesting modules of the pipeline. They are:

1. *Data Validation Module*. Data validation is a well-studied topic [14], so we implemented existing data validation rules. To make the pipeline applicable to any input data and account for possible changes of input data, we automatically deduce schema and other data properties (e.g., min and max values of numeric attribute values) from the input data. After the schema and data properties have been verified by a domain expert, they are used to detect schema and bounds anomalies in the input data. Alerts are triggered if the data does not conform to the expected format.

2. *Feature Extraction Module*. Lifespan and typical resource usage patterns are examples of the features that are useful for load prediction. In particular, we differentiate between short-lived and long-lived servers, stable and unstable servers, servers that follow a daily or a weekly pattern and servers that do not conform to such a pattern, predictable and unpredictable servers in Sections 3 and 4.2. However, other features are known to be highly predictive of future load. For example, servers that share the same subscriber identifier tend to have similar lifespan [36]. We plan to extend the feature extraction module by these additional features to make more reliable load predictions.

3. *Model Training and Inference Modules*. While many ML models can be plugged into the SEAGULL infrastructure for load prediction, we compared GluonTS [9], Prophet [12], and ARIMA [2] with respect to accuracy and scalability. We applied these models only to those servers that cannot be accurately predicted by the persistent forecast heuristic. Further details on choice of an ML model for optimized backup scheduling are in Section 5.

4. *Prediction Accuracy Evaluation Module*. In our scenario, the accuracy of load prediction for the whole day per server is less critical than correct prediction of lowest load window per day per server. So, we tailor prediction accuracy evaluation to our use case. In particular, we measure if the lowest load window is chosen correctly and if the load during this window is predicted accurately in Sections 3.1 and 4. We can also use these metrics to measure if backup windows manually selected by customers can be improved and suggest windows with expected lower load instead.

• ***Application Insights Dashboard*** [1] provides summarized view of the pipeline runs to facilitate real-time monitoring and incident management. Examples of incidents include missing or invalid input data, errors or exceptions in any step of the pipeline, and failed model deployment.

Use-Case-Specific Online Components utilize the predicted load for optimized resource allocation. Our backup scheduler runs within Master Data Services (MDS) runner per day and cluster.

The Runner Service provides the ability to deploy executables which probe their respective services resulting in measurement of availability and quality of service. The runner service is deployed to each Azure region.

For those servers that are due for full backups the next day, the backup scheduling algorithm verifies if these servers were predicted correctly for the last three weeks. This way, we verify that the servers were predictable for several weeks and we do not reschedule a backup at a worse time based on predictions we are not confident in. Three weeks of history is a compromise between prediction confidence and relevance of this rule to the majority of servers (58% of servers survive beyond three weeks, Figure 3). For such predictable servers, the algorithm extracts the predicted load for the next day and selects a time window during which customer activity is expected to be the lowest. The algorithm stores the start time of this window as a service fabric property of respective PostgreSQL and MySQL database instances. This property is used by the backup service to schedule backups. Servers that did not exist or were unpredictable for the last three weeks are scheduled for backup at default time.

3 PostgreSQL and MySQL Servers

In this section, we first define load prediction accuracy metric and then use this metric to measure if a server has stable load or follows a daily or a weekly pattern.

3.1 Load Prediction Accuracy Metric

While there are several established statistical measures of prediction error (e.g., mean absolute scaled error and mean normalized root mean squared error), we found them unintuitive and cumbersome to use in our case. They produce a number representing prediction error per server per day. They give no insights into whether the lowest load window was chosen correctly per server per day nor whether the load was predicted accurately during this window. Thus, Definitions 2 and 8 below define these two metrics.

Definition 1. *(**Acceptable Error Bound, Bucket Ratio Metric**) Given predicted and true load for a server s during a time interval t, we define the bucket ratio metric of the server s during the time interval t as the percentage of predicted data points that are within the acceptable error bound of +10/−5 of their respective true data points during the time interval t.*

Definition 1 specifies an asymmetric error bound that tolerates up to 10% over-predicted load but only at most 5% under-predicted load because a slight overestimation of low load periods is less critical for our use case than a slight underestimation that may result in interference with high customer load. In Definitions 1–9, we plug in constants that were empirically chosen by domain experts and are now used in production for the backup scheduling use case. Other constants can be plugged in for other scenarios.

Definition 2. *(**Accurate Load Prediction**) Prediction of the load of a server s during a time interval t is*

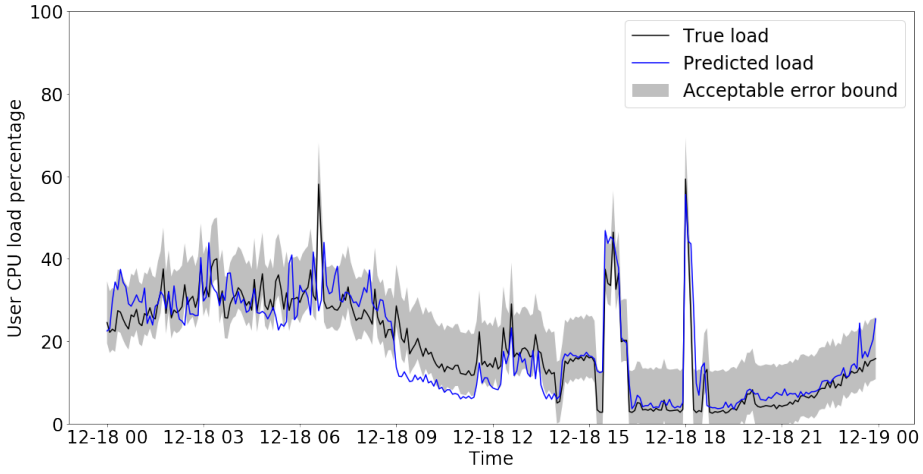


Figure 2: Acceptable error bound

accurate if the bucket ratio of the server s during the time interval t is at least 90%. Otherwise, a prediction is inaccurate.

In Figure 2, we depict predicted load as blue line, true load as back line, and acceptable error bound as gray shaded area. Intuitively speaking, a prediction is accurate if 90% of the blue line is in the shaded area. Even though for a human eye the prediction looks "close enough", the bucket ratio is only 75% and thus this prediction is inaccurate. This example illustrates that Definitions 1 and 2 impose quite strict constraints on prediction accuracy.

3.2 Server Classification

We classify the servers with respect to their lifetime and typical customer activity patterns in Figure 3. The classification provides us valuable insights about load predictability per class of servers. We will leverage these insights while choosing the ML model in Section 5.

Given a random sample of several tens of thousands of servers from four regions during one month in 2019, Figure 3 summarizes the percentage of servers that belong to each class. We define each class of servers below.

3.2.1 Server Lifespan

Servers are classified into short-lived and long-lived.

Definition 3. *(**Short-Lived Server**) A server is called long-lived if it existed more than three weeks. Otherwise, a server is called short-lived.*

As shown in Figure 3, 58% of servers "survive" for more than three weeks creating enough history to make a reliable conclusion whether they are predictable or not (Section 4.2). Remaining 42% of servers are short-lived (Figure 3). We exclude them from further consideration.

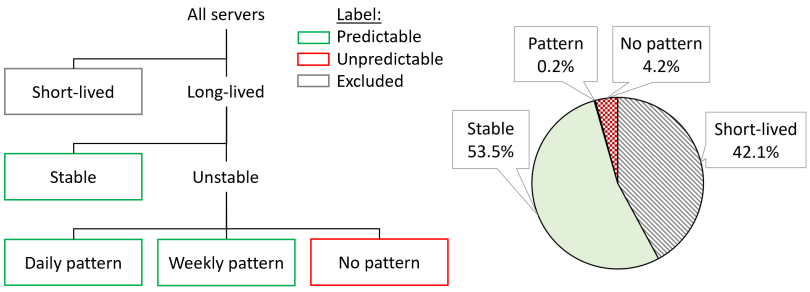


Figure 3: Classification of servers

3.2.2 Typical Customer Activity Patterns

We differentiate between stable and unstable servers.

Definition 4. *(**Stable Server**) A long-lived server is called stable during a time interval t if its load is accurately predicted by its average load during the time interval t (Definition 2). Otherwise, a server is called unstable.*

Figure 4 shows the true load of a server as a black line, the average load of this server during this week as a blue line, and the acceptable error bound as shaded gray area. The load of this server during this week is stable since the blue line is almost completely within the gray area. The bucket ratio is 99% for this server on this week (Definition 1).

53.5% of servers are long-lived and stable and thus easily predictable (Figure 3). 4.4% of long-lived unstable servers require a more detailed analysis. They are further classified into those that follow a daily or a weekly pattern and those that do not conform to such a pattern.

Definition 5. *(**Server with Daily Pattern**) Given the load of a server s on two consecutive days $d-1$ and d, the server s has a daily pattern on day d if its load on day d is accurately predicted by its load on the previous day $d-1$.*

A server has a daily pattern during a time interval t if its load conforms to this daily pattern on each day during the whole time period t.

Figure 5 shows an example of a server with a strong daily pattern. We plot the load on this day in black and on the previous day in blue. These lines overlap almost perfectly. The bucket ratio is 95%. Such a precise daily pattern could be the result of an automated recurring workload.

Definition 6. *(**Server with Weekly Pattern**) Given the load of a server s on two consecutive equivalent days of the week $d-7$ and d, the server s has a weekly pattern on day d if its load on day d is accurately predicted by its load on the previous equivalent day of the week $d-7$.*

A server has a weekly pattern during a time interval t if it does not have a daily pattern during the time period t and its load conforms to a weekly pattern on each day during the whole time interval t.

Figure 6 shows an example of a server that follows a weekly pattern. Similarly to previous Sunday (December 1), the load on this Sunday (December 8) is medium before noon and high after noon. The bucket ratio is over 90%. In contrast, the load on previous day (December 7) is low

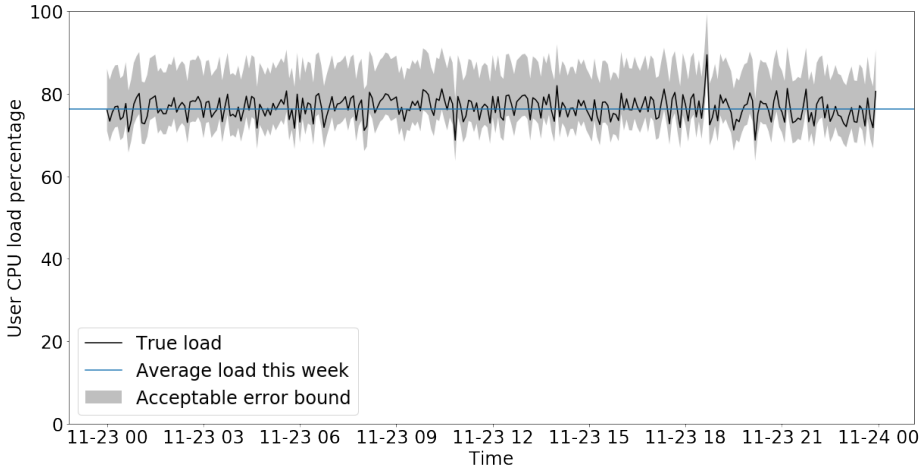


Figure 4: Stable server

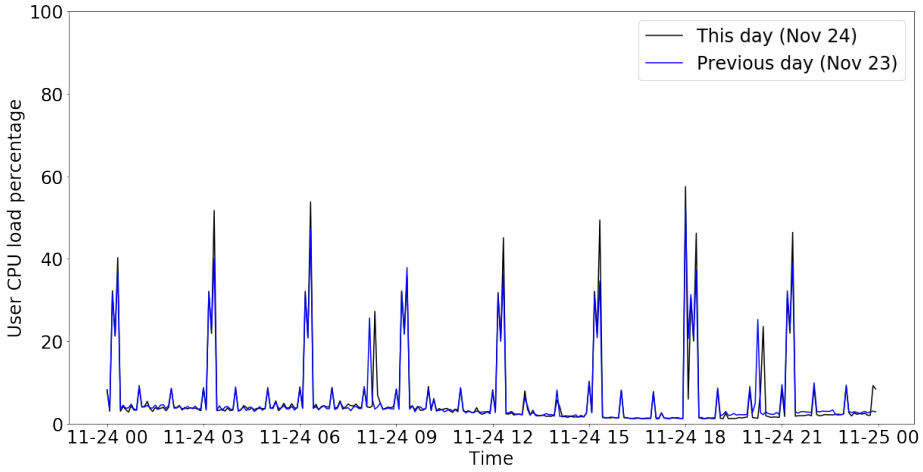


Figure 5: Server with daily pattern

before noon and medium after noon. The bucket ratio is only 1%. Thus, we conclude that this server follows a weekly pattern but does not conform to a daily pattern.

0.2% of servers conform to a daily or a weekly pattern and thus are easy to predict (Figure 3). Even though this percentage is relatively low, hundreds of top-revenue customers fall into this class of servers and cannot be disregarded.

Figure 7 illustrates the load of a server without any daily or weekly pattern. User idle time after 6AM was expected since the user was idle at the same time on the previous equivalent day (i.e., previous Sunday). However, high user activity before 6AM was not typical for this server neither the day before nor on the previous Sunday. The bucket ratio based on the previous day is 20% and based on the previous equivalent day is 72%. That is, this server follows neither daily, nor weekly pattern. 4.2% of servers do not have any pattern. They tend to be unpredictable (Section 4.2).

Summary. Figure 3 illustrates that 53.7% of servers is expected to be predictable because their

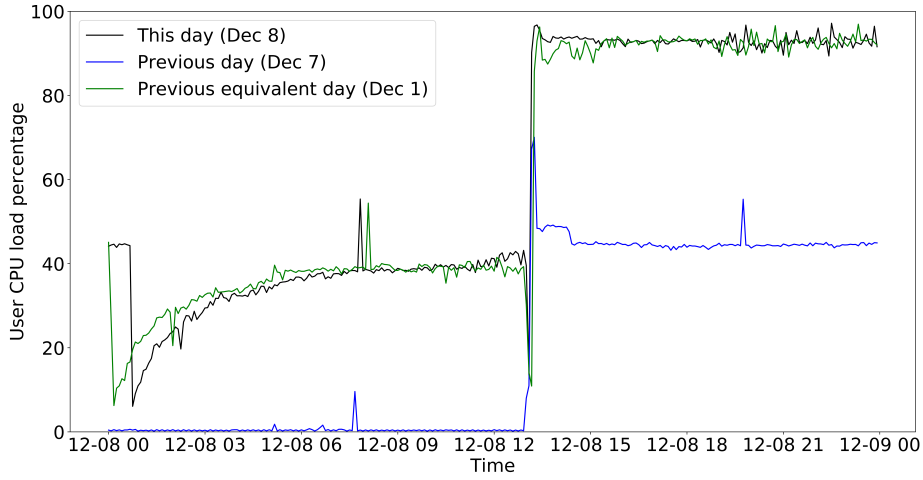


Figure 6: Server with weekly pattern

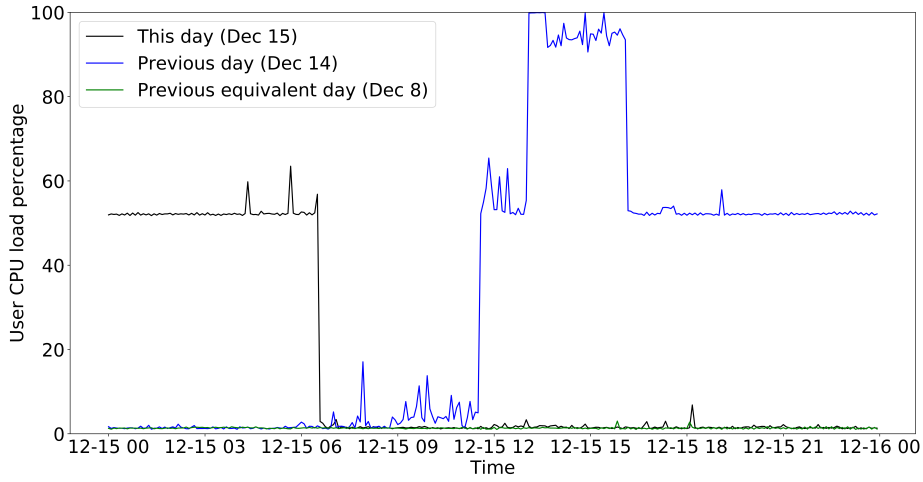


Figure 7: Server without daily or weekly pattern

load is either stable or conforms to a pattern. 4.2% of the servers are neither stable nor follow a pattern. They are likely to be unpredictable. 42.1% are short-lived and thus excluded from further consideration. These insights will be used while choosing the ML model to predict low load per server in Section 5.

4 Low Load Prediction Accuracy

In addition to the load prediction accuracy metric in Section 3.1, we now define the lowest load window metric. Based on these metrics, we then formulate the backup scheduling problem statement.

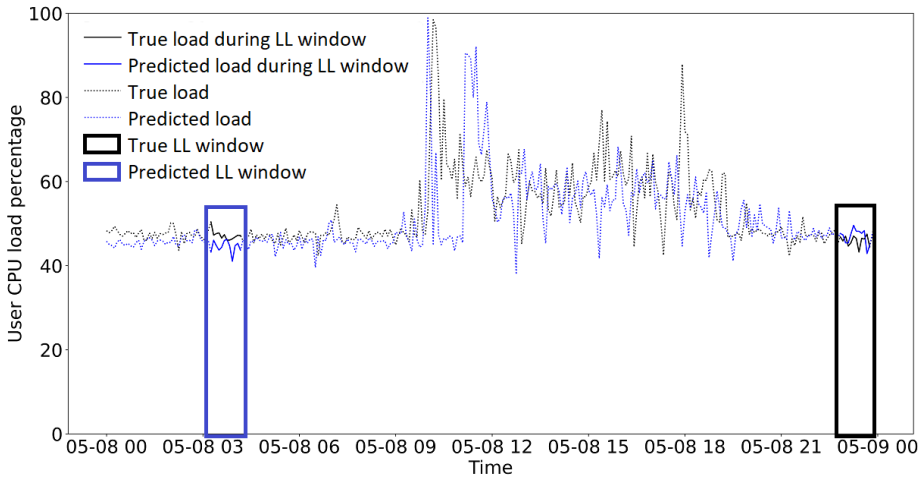


Figure 8: Correctly chosen LL window

4.1 Lowest Load Window Metric

For each server on its backup day, our goal is to predict the lowest valley in the user load that is long enough to fit a full backup of this server. The time interval of this valley is called the lowest load window. We measure if this window is chosen correctly and if the load during this window is predicted accurately. Accurate prediction of the load during the rest of the day is less critical for our purposes.

Definition 7. *(**Lowest Load (LL) Window**) Let s be a server which is due for full backup on day d. Let b be the expected duration of full backup of the server s. True LL window for the server s on the day d is the time interval of length b during which the average true load of the server s on the day d is minimal across all other time intervals of length b on the day d. Predicted LL window is defined analogously based on predicted load of the server s on day d.*

Definition 8. *(**Correctly Chosen LL Window**) Let w_t and w_p be the true and predicted LL windows for a server s on day d. If the average true load during the predicted LL window w_p is within an acceptable error bound of the average true load during the true LL window w_t, we say that the predicted LL window w_p is chosen correctly.*

In Figure 8, the true and predicted LL windows do not overlap. However, the average true load during true LL window is only slightly lower than the average true load during predicted LL window. Therefore, the true LL window would not be a significantly better time interval to run a backup than the predicted LL window. Hence, we conclude that the predicted LL window is chosen correctly.

4.2 Backup Scheduling Problem Statement

In Section 5, we focus on one instance of the load prediction problem (Section 2). Namely, for each server s that is due for full backup on day d, our SEAGULL approach aims to correctly choose the

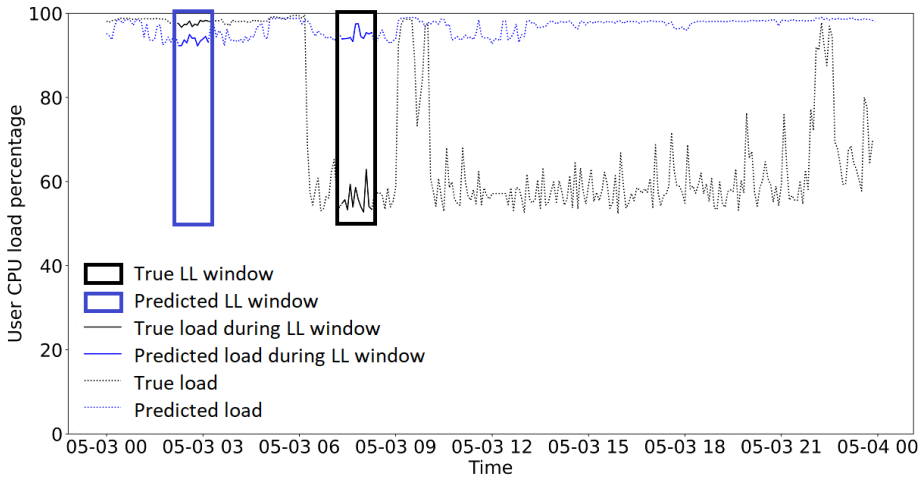


Figure 9: Incorrectly chosen LL window

LL window on day d and to accurately predict the load during this window (Definitions 2 and 8).

These two metrics are orthogonal. For example, the true and predicted LL windows coincide in Figure 10. Thus, the LL window is chosen correctly. However, the load prediction during this window is not accurate. Indeed, the true load is significantly higher than the predicted load and the bucket ratio is only 50% during this window.

The opposite case is also possible. Namely, the true load is predicted accurately during predicted LL window in Figure 9. The bucket ratio is 92%. However, the true load during the true LL window is much lower than during the predicted LL window. Thus, the LL window is not chosen correctly in this case. Based on these observations, we conclude that only both metrics combined give us reliable insights about low load prediction accuracy.

Definition 9. *(**Predictable Server**) A long-lived server is called predictable if for the last three weeks its LL windows were chosen correctly and the load during these windows was predicted accurately (Definitions 2 and 8).*

As explained in Section 2, we change backup window for predictable servers only. Servers that did not exist or were not predictable for three weeks, default to current backup time that is chosen independently from customer activity.

5 Low Load Prediction

In this section, we first describe the ML models that are commonly used for time series forecast and then choose a model per each class of servers and compare the models with respect to their accuracy and scalability.

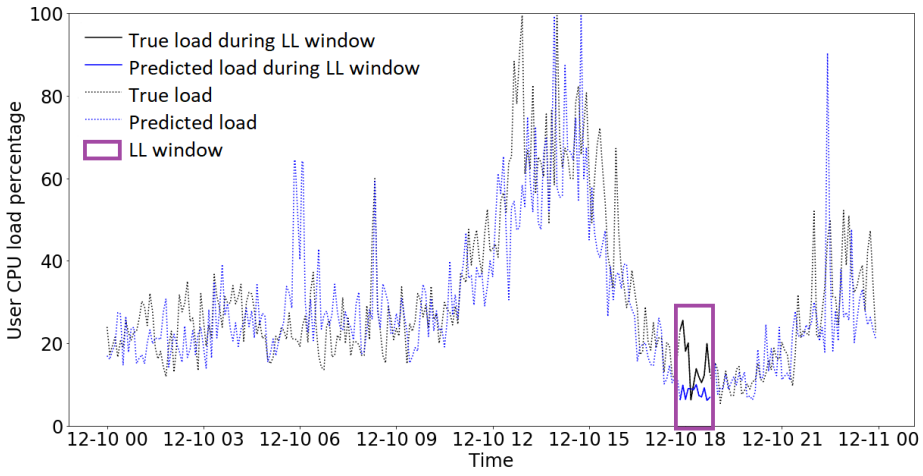
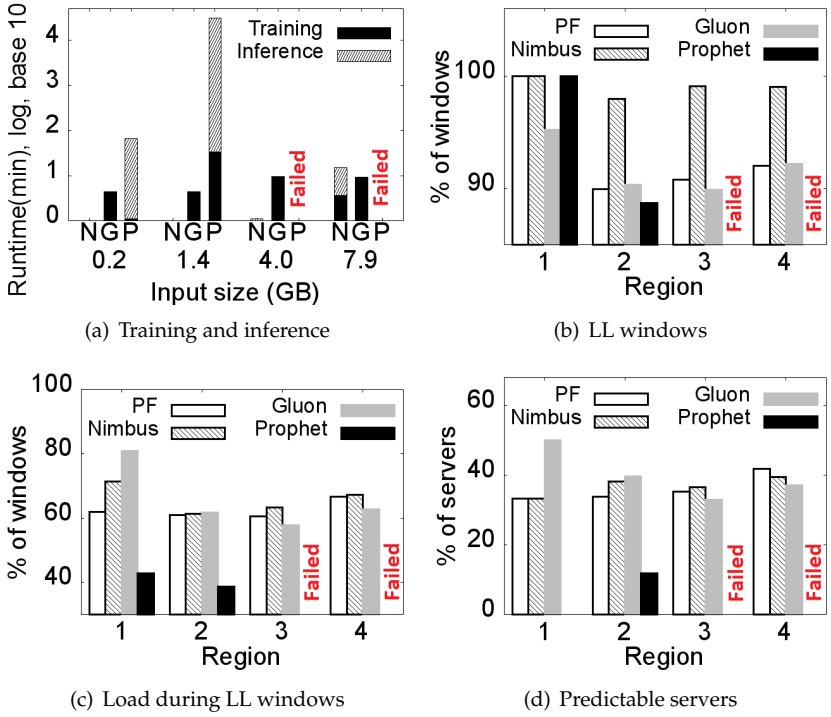


Figure 10: Low load prediction accuracy



(a) Training and inference

(b) LL windows

(c) Load during LL windows

(d) Predictable servers

Figure 11: Low load prediction using Persistent Forecast (PF), Nimbus (N), Gluon (G), and Prophet (P)

5.1 ML Models for Time Series Forecast

We now summarize the key ideas of the ML models that we considered to predict the low customer activity per server on its backup day.These models range from simple heuristics to complex neural-network-based ones.

Persistent Forecast refers to replicating previously seen load per server as the forecast of the

load for this server. We compared three variations of the persistent forecast model:

• *Previous week average* makes a prediction as the average load of a particular server during previous week.

• *Previous equivalent day* forecasts the load of a server by replicating its load on previous equivalent day of the week.

• *Previous day* takes the load a server on the previous day and utilizes it as predicted load on the next day.

NimbusML [11] is a Python module that provides Python bindings for ML.NET. NimbusML aims to enable data science teams that are more familiar with Python to take advantage of ML.NET's functionality and performance. It provides battle-tested, state-of-the-art ML algorithms, transforms, and components. Specifically, we use Singular Spectrum Analysis to transform forecasts.

GluonTS [9] is a toolkit for probabilistic time series modeling, focusing on deep learning-based models. We train a simple feed forward estimator. We tried several other estimators but this model achieved highest accuracy.

Prophet [12] is open source software released by Facebook. It forecasts a time series data based on an additive model where non-linear trends are fit with yearly, weekly, and daily seasonality, plus holiday effects. It works well for time series that have strong seasonal effects and several seasons of historical data. Prophet is robust to missing data and shifts in the trend, and typically handles outliers well.

ARIMA: Auto-Regressive Integrated Moving Average model [2] forecasts the future values of a series based on the different seasonal and temporal structures in the series. At inference, it predicts one signal at a time by fitting to this signal's prior values.

5.2 ML Model per Class of Servers

In this section, we discuss the applicability of each model to each class of servers we identified in Section 3.2. We differentiate between two cases:

• *Stable servers and servers that follow business patterns that can be recognized by persistent forecast*. Obviously, such servers can be accurately predicted by persistent forecast and no complex ML models are needed. Indeed, the previous week average can predict the load of stable servers (Definition 4); 53.5% of servers are stable (Figure 3). Previous equivalent day is more powerful than previous week average because it captures a weekly pattern (Definition 6), including stable load which covers 53.6% of servers. Previous day is also more powerful than previous week average, since it captures a daily pattern (Definition 5), including stable loads. 53.7% of servers can be predicted by the previous day's pattern. Since previous day is suitable for the largest subset of servers, we focus on this variant in the following.

• *Unstable servers that do not conform to a pattern that can be recognized by persistent forecast*. 4.2% of servers fall into this category. In Section 5.3, we apply ML models to such servers to find out if these models can detect a predictable load pattern for these servers.

5.3 Experimental Comparison of ML Models

5.3.1 Experimental Setup

Hardware. We conducted all experiments on a VM running Ubuntu 18.04. It has 16 CPUs and 64GB of memory.

Input Data. As described in Section 2, the pipeline runs per Azure region once a week. Thus, our input data is partitioned by region and week. Since the size of regions varies, the size of input files ranges from hundreds of kilobytes to a few gigabytes. Below, we randomly selected four input files with different sizes to demonstrate the scalability of ML models and find out if there are differences in accuracy of predictions between models and regions. The input files are in csv format. They contain server identifier, timestamp in minutes, average user CPU load percentage per five minutes, default backup start and end timestamps.

In order to identify predictable servers, we have to consider three weeks (Definition 9). To infer the load per server on its backup day, ML models are trained on one week of data prior to backup day per server. Thus, each input data set contains four weeks in one region, unless stated otherwise. We consider servers have at least three days of history prior to their backup days to train the ML models.

Methodology. We implemented the SEAGULL pipeline in Python. Our base-line implementation is *single-threaded*. Our *multi-threaded* Dask-based [8] implementation partitions the data per server and processes servers in parallel.

Metrics. For each ML model, we measure the percentage of correctly chosen LL windows, the percentage of LL windows with accurately predicted load, and the percentage of predictable servers among servers that existed at least three weeks (Definitions 2, 8, and 9). We measure the runtime of training, inference, and accuracy evaluation in minutes.

5.3.2 Stable Servers and Servers with Pattern

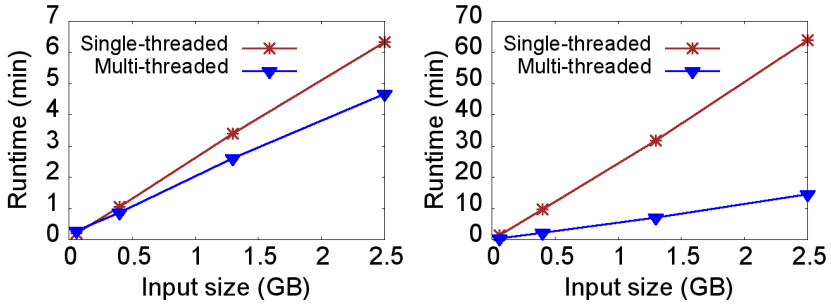
As explained in Section 5.2, majority of long-lived servers have stable load or follow daily or weekly patterns that can be recognized by persistent forecast. Therefore, we use persistent forecast to predict the load of such servers. For our sample data set, this heuristic correctly selected 99.83% of LL windows, accurately predicted the load during 99.06% of all windows, and classified 96.92% of servers as predictable.

5.3.3 Unstable Servers Without Pattern

We now apply ML models from the tools mentioned in Section 5.1 to unstable servers that do not follow business patterns that can be recognized by persistent forecast.

Training and Inference. *Persistent forecast* does not require training because it uses the load per server on the previous day as predicted load per server on the next day.

NimbusML scales well (Figure 11(a)). Runtime for training and inference increases linearly from 2.5 seconds to 4 minutes as the number of servers grows from 10 to 700. Some of these measurements are not visible due to log scale with base 10 in Figure 11(a).



(a) Backup day (b) Each day

Figure 12: Prediction accuracy evaluation

GluonTS also scales well. Training time ranges from 4 to 10 minutes, while inference time ranges from 0.2 to 16 seconds as the number of servers grows from 10 to 700.

Prophet does not scale as well. Its training time grows from 1 to 34 minutes, while inference takes from 1 to 15 hours as the number of servers increases from 10 to 100. Thus, we implemented Prophet on Dask and achieved up to 60X speedup compared to single-threaded execution. However, when the number of servers exceeds 200, Prophet runs out of memory on Dask independently from the number of workers. Single-threaded execution does not terminate.

ARIMA is computationally intensive since it searches the optimal values of six parameters per server in order to make an accurate load prediction per server. We had explored parameter sharing between servers but that resulted in a worsening of accuracy. While inference time is within a few seconds per server, fitting may take up to 3 hours per server. Hence, executing ARIMA in parallel for each server does not make runtime of ARIMA comparable to other models.

Low Load Prediction Accuracy. NimbusML correctly chooses the highest percentage of LL windows compared to other tools (Figure 11(b)). There is slight variance in accuracy of load prediction during LL windows and the percentage of predictable servers across regions and models (Figures 11(c) and 11(d)). Accuracy of persistent forecast, NimbusML, and GluonTS is comparable with respect to these two metrics. Prophet has similar or lower accuracy compared to the other two tools.

Choice of Model for Final Deployment. We deployed the persistent forecast based on previous day to production to predict low load per server due to the following practical considerations. One, given that persistent forecast is selected to predict stable servers and servers with pattern, it is easier to maintain a single model for the entire fleet of servers than a different model per each class of servers. Two, the accuracy of other models is not significantly higher than the accuracy of persistent forecast. Three, persistent forecast does not introduce any computational delay due to training and thus scales better than other models.

5.3.4 Prediction Accuracy Evaluation

In this section, we use the same settings as in production. Namely, each input data set contains the load of all servers during one week in one Azure region.

Backup Day. Figure 12(a) shows the runtime of single-threaded and multi-threaded implementation of accuracy evaluation of predicted load per server on its backup day while varying the size of input data. While for the smallest data set, Dask is 5 seconds slower than the single-threaded execution, for larger input data Dask consistently wins because our data is partitionalble per server and computations are parallelizable per server. For 2.5GB, Dask is 26% faster than single-threaded execution.

Each Day. In order to further improve on backup scheduling, we aim to move a backup of a server from its default backup day to other day of the week if the load is lower and/or prediction is more accurate on another day. In Figure 12(b), we measure the runtime of accuracy evaluation on each day one week ahead per server while varying the size of input data. For 2.5GB, the single-threaded implementation runs for over 1 hour. Dask consistently achieves 3-4.6X speedup compared to the single-threaded implementation for all input sizes. For 2.5GB, Dask terminates after 15 minutes which we consider to be an acceptable computational delay for a large Azure region.

6 Related Work

Systems for ML were proposed in the past. However, most of them lack easy integration with Azure compute [10, 13, 17, 18, 25, 32]. We also considered leveraging the model-serving infrastructure Resource Central [16]. However, at that time Azure ML [4] provided support and integration for a broader set of modeling and tracking tools.

Load Prediction for optimized resource allocation on a cluster has become a popular research direction in the recent years. Existing approaches focus on predicting survivability of databases for optimized resource provisioning [36], idle time detection for database quiescing and overbooking [29, 41], database workload prediction for database consolidation [20], VM workload prediction [27] for oversubscribing servers [16], dynamic VM provisioning [15], and reducing performance interference between VMs co-located on the same physical machine [34], workload classification for capacity planning and task scheduling [33], cost- and QoS-aware application placement in virtualized server clusters [40, 42], and preemptive auto-scale of resources [21, 22, 23, 24, 28, 38, 35, 37, 39, 43]. None of these approaches focused on predicting low load windows for optimized scheduling of system maintenance tasks. Thus, these approaches neither define low load prediction accuracy, nor compare ML models from the perspective of low load prediction.

Job Scheduling Algorithms were proposed in the literature [19, 26, 31]. Our backup scheduling algorithm (Section 2) is not the focus of this paper. It is just one example how the SEAGULL infrastructure can be used in production for optimized resource allocation on the cloud.

7 Summary of Results

We built the SEAGULL infrastructure for predicting the load of a system component (database, server, VM, node on a cluster, etc.) for projects aiming to optimize resource allocation. The SEAGULL infrastructure is deployed to production worldwide to schedule backups of PostgreSQL and MySQL servers such that these backups do not collide with high customer load. Backups of 33% of servers are now rescheduled to run during predicted low load windows. Backup collisions for busy servers during peak customer activity periods are now reduced by 50%.

To find the middle ground between accuracy of low load prediction and the overhead of model training and inference, we classified the servers with respect to their typical customer activity patterns and concluded that low load per server can be accurately predicted by persistent forecast. This heuristic correctly selected 99% of low load windows, accurately predicted the load during 96% of all windows, and classified 75% of long-lived servers as predictable.

Acknowledgements

The authors want to thank Markus Weimer, Matteo Interlandi, and Siqi Liu for their useful feedback on this paper. We are also grateful to Purnesh Dixit, Santhosh Pillai, Akshaya Annavajhala, Larry Franks, and Chris Lauren for multiple fruitful discussions about AML pipelines. We would further like to thank Hiren Patel for helping us with large-scale telemetry analysis.

References

[1] Application Insights. https://docs.microsoft.com/en-us/azure/azure-monitor/app/app-insights-overview.

[2] ARIMA. https://pypi.org/project/pmdarima/.

[3] Azure Data Lake Analytics. https://azure.microsoft.com/en-us/services/data-lake-analytics.

[4] Azure ML. https://azure.microsoft.com/en-us/services/machine-learning/.

[5] Azure SQL Database pricing. https://azure.microsoft.com/en-us/pricing/details/sql-database/single/.

[6] Azure SQL Database serverless. https://docs.microsoft.com/en-us/azure/sql-database/sql-database-serverless.

[7] Cosmos DB. https://docs.microsoft.com/en-us/azure/cosmos-db/introduction.

[8] Dask. https://dask.org/.

[9] GluonTS. https://gluon-ts.mxnet.io/.

[10] MLflow. https://mlflow.org/.

[11] NimbusML. https://docs.microsoft.com/en-us/python/api/nimbusml/nimbusml.timeseries.ssaforecaster.

[12] Prophet. https://facebook.github.io/prophet/.

[13] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: A system for large-scale machine learning. In *OSDI*, pages 265–283, 2016.

[14] E. Breck, N. Polyzotis, S. Roy, S. E. Whang, and M. Zinkevich. Data Validation for Machine Learning. In *SysML*, 2019.

[15] R. Calheiros, E. Masoumi, R. Ranjan, and R. Buyya. Workload Prediction Using ARIMA Model and Its Impact on Cloud Applications' QoS. *IEEE Transactions on Cloud Computing*, 3:449–458, 08 2014.

[16] E. Cortez, A. Bonde, A. Muzio, M. Russinovich, M. Fontoura, and R. Bianchini. Resource Central: Understanding and Predicting Workloads for Improved Resource Management in Large Cloud Platforms. In *SOSP*, 2017.

[17] D. Crankshaw, P. Bailis, J. E. Gonzalez, H. Li, Z. Zhang, M. J. Franklin, A. Ghodsi, and M. I. Jordan. The Missing Piece in Complex Analytics: Low Latency, Scalable Model Management and Serving with Velox. In *CIDR*, 2015.

[18] D. Crankshaw, X. Wang, G. Zhou, M. J. Franklin, J. E. Gonzalez, and I. Stoica. Clipper: A Low-Latency Online Prediction Serving System. In *NSDI*, pages 613–627, 2017.

[19] C. Curino, D. E. Difallah, C. Douglas, S. Krishnan, R. Ramakrishnan, and S. Rao. Reservation-Based Scheduling: If You're Late Don't Blame Us! In *SOCC*, page 1–14, 2014.

[20] C. Curino, E. P. Jones, S. Madden, and H. Balakrishnan. Workload-Aware Database Monitoring and Consolidation. In *SIGMOD*, page 313–324, 2011.

[21] S. Das, F. Li, V. R. Narasayya, and A. C. König. Automated Demand-driven Resource Scaling in Relational Database-as-a-Service. In *SIGMOD*, pages 1923–1924, 2016.

[22] C. Delimitrou and C. Kozyrakis. Quasar: Resource-Efficient and QoS-Aware Cluster Management. *SIGPLAN Not.*, 49(4):127–144, Feb. 2014.

[23] A. Floratou, A. Agrawal, B. Graham, S. Rao, and K. Ramasamy. Dhalion: Self-Regulating Stream Processing in Heron. In *Proc. VLDB Endow.*, pages 1825–1836, 2017.

[24] S. Islam, J. Keung, K. Lee, and A. Liu. Empirical Prediction Models for Adaptive Resource Provisioning in the Cloud. *Future Generation Comp. Syst.*, 28:155–162, 01 2012.

[25] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional Architecture for Fast Feature Embedding. In *MM*, page 675–678. Association for Computing Machinery, 2014.

[26] S. A. Jyothi, C. Curino, I. Menache, S. M. Narayanamurthy, A. Tumanov, J. Yaniv, R. Mavlyutov, I. n. Goiri, S. Krishnan, J. Kulkarni, and S. Rao. Morpheus: Towards Automated SLOs for Enterprise Clusters. In *OSDI*, page 117–134, 2016.

[27] A. Khan, X. Yan, S. Tao, and N. Anerousis. Workload Characterization and Prediction in the Cloud: A Multiple Time Series Approach. In *IEEE Network Operations and Management Symposium*, pages 1287–1294, 2012.

[28] C. Kilcioglu, J. M. Rao, A. Kannan, and R. P. McAfee. Usage Patterns and the Economics of the Public Cloud. In *WWW*, page 83–91, 2017.

[29] W. Lang, K. Ramachandra, D. J. DeWitt, S. Xu, Q. Guo, A. Kalhan, and P. Carlin. Not for the Timid: On the Impact of Aggressive over-Booking in the Cloud. *Proc. VLDB Endow.*, 9(13):1245–1256, 2016.

[30] A. Li, X. Yang, S. Kandula, and M. Zhang. CloudCmp: Comparing Public Cloud Providers. In *IMC*, page 1–14, 2010.

[31] D. Lo, L. Cheng, R. Govindaraju, P. Ranganathan, and C. Kozyrakis. Heracles: Improving Resource Efficiency at Scale. In *ISCA*, pages 450–462, 2015.

[32] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen, D. Xin, R. Xin, M. J. Franklin, R. Zadeh, M. Zaharia, and A. Talwalkar. MLlib: Machine Learning in Apache Spark. *Journal of Machine Learning Research*, 17(34):1–7, 2016.

[33] A. K. Mishra, J. L. Hellerstein, W. Cirne, and C. R. Das. Towards Characterizing Cloud Backend Workloads: Insights from Google Compute Clusters. *SIGMETRICS Perform. Eval. Rev.*, 37(4):34–41, Mar. 2010.

[34] D. Novaković, N. Vasić, S. Novaković, D. Kostić, and R. Bianchini. DeepDive: Transparently Identifying and Managing Performance Interference in Virtualized Environments. In *USENIX ATC*, pages 219–230, 2013.

[35] P. Padala, K.-Y. Hou, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, and A. Merchant. Automated Control of Multiple Virtualized Resources. In *EuroSys*, page 13–26, 2009.

[36] J. Picado, W. Lang, and E. C. Thayer. Survivability of Cloud Databases - Factors and Prediction. In *SIGMOD*, page 811–823, 2018.

[37] N. Roy, A. Dubey, and A. Gokhale. Efficient Autoscaling in the Cloud Using Predictive Models for Workload Forecasting. In *CLOUD*, pages 500–507, 2011.

[38] Z. Shen, S. Subbiah, X. Gu, and J. Wilkes. CloudScale: Elastic Resource Scaling for Multi-tenant Cloud Systems. In *SOCC*, 2011.

[39] R. Taft, N. El-Sayed, M. Serafini, Y. Lu, A. Aboulnaga, M. Stonebraker, R. Mayerhofer, and F. Andrade. P-Store: An Elastic Database System with Predictive Provisioning. In *SIGMOD*, page 205–219, 2018.

[40] A. Verma, P. Ahuja, e. V. Neogi, Anindya", and R. Schantz. pMapper: Power and Migration Cost Aware Application Placement in Virtualized Systems. In *Middleware*, pages 243–264, 2008.

[41] L. Viswanathan, B. Chandra, W. Lang, K. Ramachandra, J. M. Patel, A. Kalhan, D. J. DeWitt, and A. Halverson. Predictive Provisioning: Efficiently Anticipating Usage in Azure SQL Database. In *ICDE*, pages 1111–1116, 2017.

[42] H. Yang, A. Breslow, J. Mars, and L. Tang. Bubble-Flux: Precise Online QoS Management for Increased Utilization in Warehouse Scale Computers. In *ISCA*, page 607–618, 2013.

[43] Zhenhuan Gong, Xiaohui Gu, and J. Wilkes. PRESS: PRedictive Elastic ReSource Scaling for cloud systems. In *TNSM*, pages 9–16, 2010.

A Preemptive Auto-scale of SQL Databases

As a follow-up project, we will use SEAGULL infrastructure (Figure 1) for preemptive auto-scale of resources for Azure SQL databases (Example 2 in Section 1). Below, we briefly summarize our initial results in database classification and load prediction.

A.1 Classification of SQL Databases

SQL data contains database identifier, timestamp in minutes, and average CPU load per 15 minutes. We differentiate between stable and unstable databases.

Definition 10. *(**Stable Database**) A stable database is defined as a database whose variation does not exceed one standard deviation for the last three days in the period evaluated. Otherwise, a database is called unstable.*

We analyzed a random sample of several thousands of single standard and premium SQL databases during one month in 2019 and concluded that 19.36% of them are stable.

A.2 Prediction Error Metrics

For the preemptive auto-scale use case, we predict the CPU load per database 24 hours ahead. We define error as the difference between the forecast and the true load in Equation 1. We use the standard metrics, namely, Mean Normalized Root Mean Squared Error (Mean NRMSE) and Mean Absolute Scaled Error (MASE) to evaluate accuracy of models in Equations 2 and 3.

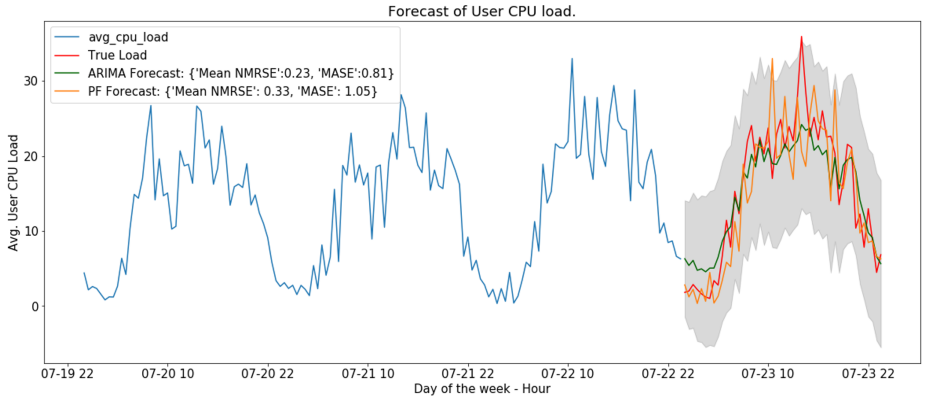


Figure 13: Accurate load prediction

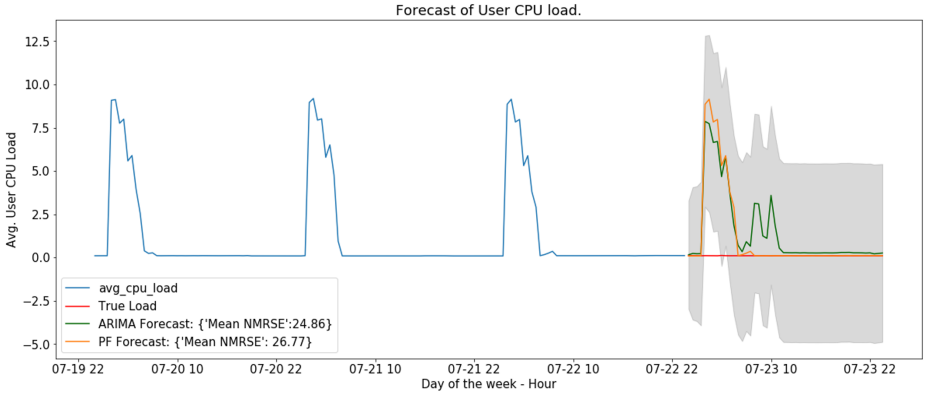


Figure 14: Inaccurate load prediction

$$error = forecast - true \tag{1}$$

$$Mean\ NRMSE = \frac{\sqrt{mean(error^2)}}{mean(true)} \tag{2}$$

$$MASE = mean\left(\frac{abs.(error)}{normalizing\ factor}\right) \tag{3}$$

A mean NRMSE of 1 is produced when the mean is predicted as the forecast, anything less than 1 would mean doing better than forecasting the mean. The normalizing factor in this case is the error produced by a one step ahead true forecast. MASE of less than 1 means doing better than a one step ahead true forecast. Figures 13 and 14 show examples of values of these metrics.

A.3 Load Prediction

Figures 15 and 16 summarize the accuracy and runtime of training and inference per ML model described in Section 5.1. Neural network refers to GluonTS [9] and persistent forecast is based

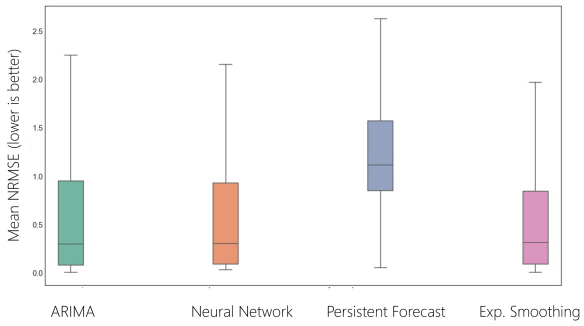


Figure 15: Model accuracy

Model	Training time	Inference time	Median MASE	Median Mean NRMSR
Persistent forecast	N/A	50 secs	1.24	0.44
Neural network	30 mins	15 secs	1.11	0.44
ARIMA	60 hours	3 mins	1.19	0.30

Figure 16: Training, inference, and accuracy

on previous day. GluonTS and ARIMA are trained on one week of historical load per database. ARIMA runs in parallel per database on HDI cluster with 2 head nodes with 4 cores and 28GB of memory per node and 2 worker nodes with 4 cores and 56GB per node. Given the coarse granularity of SQL data (per 15 minutes), ARIMA works better than for the fine-grained PostgreSQL and MySQL data (per 5 minutes). Nevertheless, the runtime for training of ARIMA is still not comparable with other models. Based on this preliminary evaluation, we concluded that for SQL databases persistent forecast also finds the middle ground between accuracy and computational overhead.