Seagull: An Infrastructure for Load Prediction and Optimized Resource Allocation

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

ABSTRACT

As the leading cloud service provider, Microsoft Azure is dedicated to guarantee high quality of service to its customers, in particular, during periods of high customer activity. We deploy machine learning (ML) to predict customer load and leverage these predictions to optimize resource allocation in various ways such as schedule maintenance operations such that they do not interfere with high customer load, balance load across the nodes on a cluster, and ultimately auto-scale resources depending on expected demand. To enable all these projects, we built an Azure ML pipeline that takes customer load per server, validates this data, trains an ML model, deploys this model, predicts customer load per server one day ahead, and evaluates the accuracy of these predictions. We deployed this infrastructure in production for tens of thousands of PostgreSQL and MySQL servers in all Azure regions for backup scheduling during time intervals of expected low customer load to avoid interference with peaks of customer activity and thus guarantee high quality of service to our customers at all times.

Raj: Example of a comment

Saikat: One more comment

PVLDB Reference Format:
Poppe, et al. Seagull: An Infrastructure for Load Prediction and Optimized Resource Allocation. *PVLDB*, 12(xxx): xxxx-yyyy, 2021.
DOI: https://doi.org/10.14778/xxxxxxx.xxxxxxx

1. INTRODUCTION

Microsoft Azure, Google Cloud Platform, and Amazon Web Services are the leading cloud service providers that continuously expand cloud computing capabilities. They aim to guarantee high quality of service to their customers, while controlling operating costs [21]. 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. It is critical to cloud service providers to come up with tools to ease the growing pain of manual resource management. To this end, machine learning techniques are becoming increasingly popular to predict resource demand and leverage these predictions to automatically optimize resource allocation on the cloud [12, 15, 16, 18, 20].

Motivating Example 1: *Preemptive Auto-Scale*. SQL databases have either 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. However, resource demand varies over time. Indeed, many databases follow daily or weekly patterns with extensive busy or idle periods. 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 certain delay in compute warm-up after idle periods. Based on expected resource demand, we aim to extend current reactive auto-scale rules by preemptive policies to reduce this delay and make serverless compute suitable for time-critical applications.

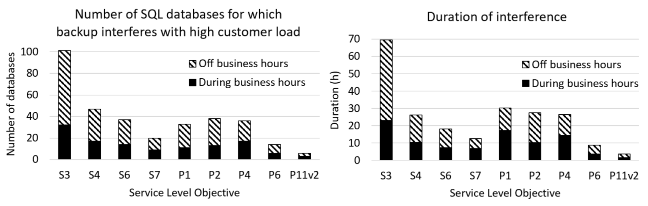


Figure 1: Current backup scheduling

Motivating Example 2: *Backup Scheduling*. Backups of SQL databases and PostgreSQL and MySQL servers 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 current backup windows. We analyzed single standard and premium SQL databases in one region during one week in August 2019. Figure 1 summarizes the number of SQL databases for which backups interfere with high customer load (i.e., over 65% of capacity) and the total duration of interference. As these charts illustrate, the problem will not be solved by simply moving backups off business hours. Instead, a customized backup schedule is required for each server.

This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-nd/4.0/. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.
Proceedings of the VLDB Endowment, Vol. 12, No. xxx
ISSN 2150-8097.
DOI: https://doi.org/10.14778/xxxxxxx.xxxxxxx

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. Such insights into future load of any system component can be utilized to optimize resource allocation in various ways.

Challenges. We faced 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 tens of thousands of 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. To the best of our knowledge, our Seagull approach is the first to design, implement, and deploy to production worldwide a generic infrastructure that predicts load and optimizes resource allocation of Azure products. Furthermore, while time series forecast in general and load prediction in particular are well studied topics, none of the existing approaches focused on predicting the lowest valley in customer CPU load for optimized backup scheduling. Instead, state-of-the-art approaches focus on idle time detection for predictive resource provisioning and overbooking [18, 21], VM workload prediction for oversubscribing servers [12], and demand-driven auto-scale of resources [15, 16, 20]. Therefore, these approaches do not tackle the unique challenges of low load prediction for optimized backup scheduling described above.

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 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 recurring pattern. Therefore, the load per server on previous day can serve as strong predictor of the load per server on the next day. This heuristic is called persistent forecast based on previous day. 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. This infrastructure is generic since it can be used to optimize resource allocation in various ways. For example, load balancing, tenant placement, and auto-scale would profit from predictions of future load. Seagull infrastructure is deployed in production for tens of thousands of PostgreSQL and MySQL servers in all Azure regions for optimized backup scheduling.

• We conducted comprehensive analysis to classify the servers into homogeneous groups based on their lifespan and typical customer activity pattern. Based on this classification, we concluded that majority of servers are either stable or follow a daily or a weekly pattern. Therefore, their load is predictable. Less then 5% of servers are unstable and do not follow a pattern. Hence, their load is unpredictable.

• We define the accuracy of low load window prediction per server on its backup day as the combination of two metrics. One, a low 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, user CPU load during a low 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 compared four ML models commonly used for time series prediction (persistent forecast, Neural Networks [9], Prophet [10], and Arima [2]) with respect to accuracy and scalability. We concluded that persistent forecast finds the middle ground between them in our case. More complex ML models are the tool of last resort in follow-up projects in which highly accurately predicted load per server per entire day is more critical than in our scenario.

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 INFRASTRUCTURE

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 cus-

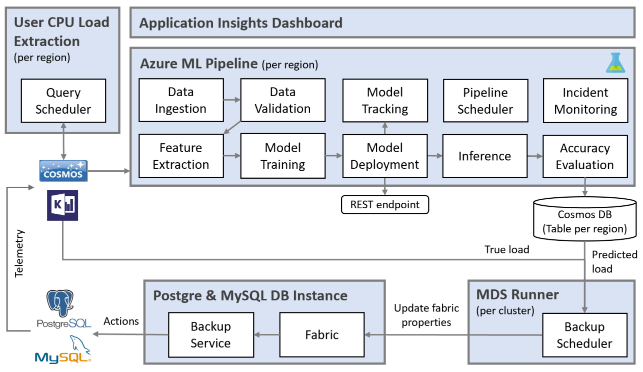


Figure 2: Seagull infrastructure

tomer activity is an instance of this generic load prediction problem. It is formulated in Section 4.2.

Seagull Infrastructure is shown in Figure 2. Below, we give an overview of its key components and refer the reader to in-depth analysis in the following sections.

• ***User CPU Load Extraction.*** In this project, we have selected 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 compactness and readability.

User CPU load extraction module is implemented as a recurring query that runs once a week and extracts user CPU load and default backup window per server, region, and week from raw telemetry. This query can be easily replaced and/or extended to extract other metrics per server. This query stores one file per region and week in Azure Data Lake Store [3]. These files are input to the Azure ML pipeline.

• ***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. A run of the pipeline is scheduled once a week per each region. This pipeline consumes user CPU load per server, region and week, validates this data, 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 per server and evaluates the accuracy of predictions per server. Results of this evaluation are stored in Cosmos DB [7] that maintains one table per region. These results are input to the backup scheduling algorithm.

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

1. *Data Validation Module.* Since data validation is a well-studied topic [11], we implemented existing data validation rules. We automatically deduct schema and data properties (e.g., min and max values) from the input data. After the schema and data properties were verified by a domain expert, they are used to detect schema and bounds anomalies in input data. Alerts are triggered if the input data does not conform to the expected format.

2. *Feature Extraction Module.* Server lifespan and typical customer activity 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 [19]. 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 any ML model can be plugged into the Seagull infrastructure, we compared persistent forecast, GluonTS [9], Prophet [10], and Arima [2] with respect to accuracy and scalability and concluded that persistent forecast finds the middle ground between them in our case. Further details on choice of an ML model for optimized backup scheduling are in Section 5.

4. *Prediction Accuracy Evaluation Module.* Since 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, we tailored 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.

• ***Backup Scheduling Algorithm*** runs within Master Data Services (MDS) runner once a day per 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.

The backup scheduling algorithm consumes the predicted user CPU load per server for the next day. For those servers that are due for full backups the next day and were predicted correctly in the last three weeks, the algorithm selects a time window during which customer activity is expected to be the lowest on the next day. 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 not predicted correctly for the last three weeks are scheduled for backup at default time.

• ***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, failed model deployment, etc.

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.

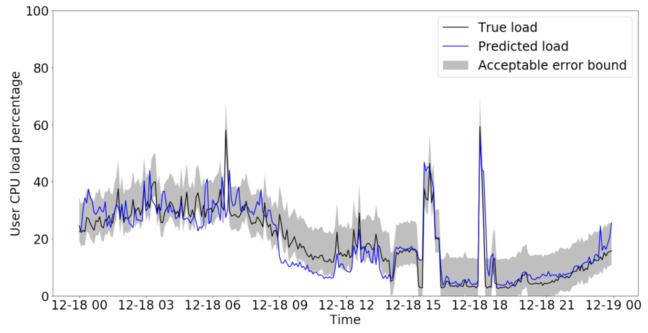


Figure 3: Acceptable error bound

They give no insights into whether a low 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 $+10/-5$[1] of their respective true data points during the time interval t.

Definition 2. (**Accurate Load Prediction**) Prediction of the load of a server s during a time interval t is *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 3, 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 PostgreSQL and MySQL servers with respect to their lifetime and typical user activity pattern in Figure 4. This classification provides us valuable insights about predictability of the load per server. We will leverage these insights while choosing the ML model in Section 5.

The number of servers per class can vary significantly over time and across regions. This observation strengthens the need for an infrastructure that detects these data drifts, retrains the model, and/or notifies about these changes. Given a random sample of 23K servers from four regions during four weeks in November and December 2019, Figure 4 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.

[1]In Definitions 1–9, we plug in empirically chosen constants that are used in production for the backup scheduling use case. Other constants can be plugged in for other scenarios. Tuning of these constants is out of the scope of this paper.

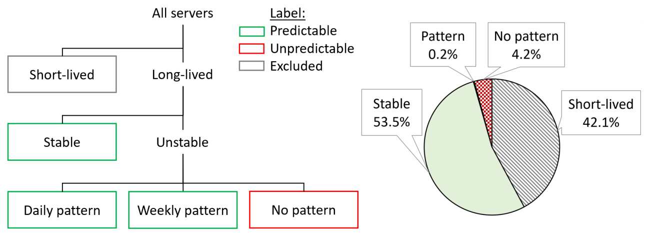


Figure 4: Classification of servers

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 4, 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 4). We exclude them from further consideration.

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 5 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 4). 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 6 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.

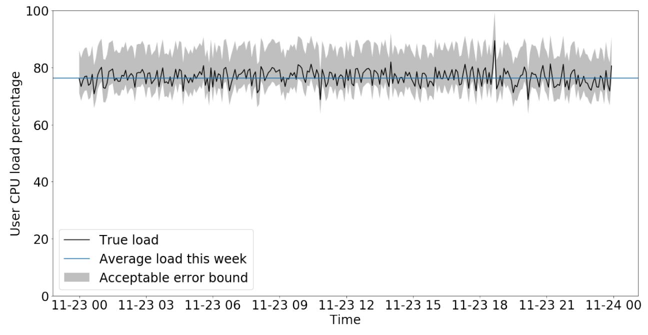


Figure 5: Stable server

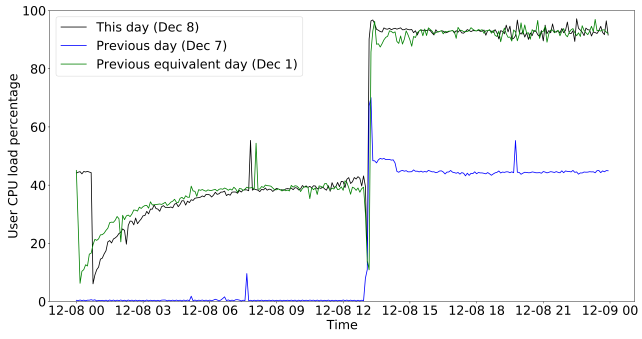


Figure 7: Server with weekly pattern

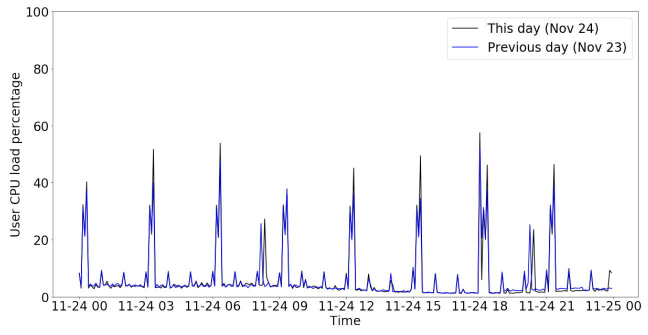


Figure 6: Server with daily pattern

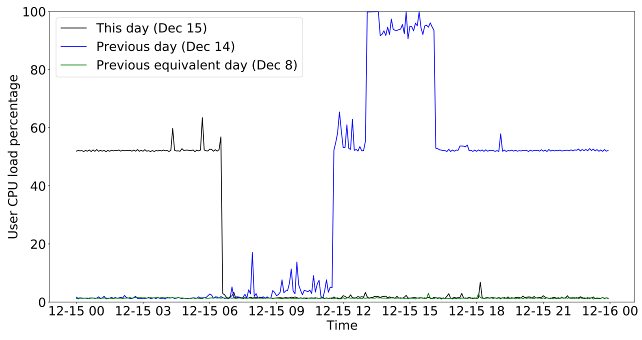


Figure 8: Server without daily or weekly pattern

Figure 7 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 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 4). Even though this percentage is relatively low, hundreds of top-revenue customers fall into this class of servers and cannot be disregarded.

Figure 8 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 4 illustrates that 53.7% of servers is expected to be predictable because their 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 identify predictable servers.

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 9, 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

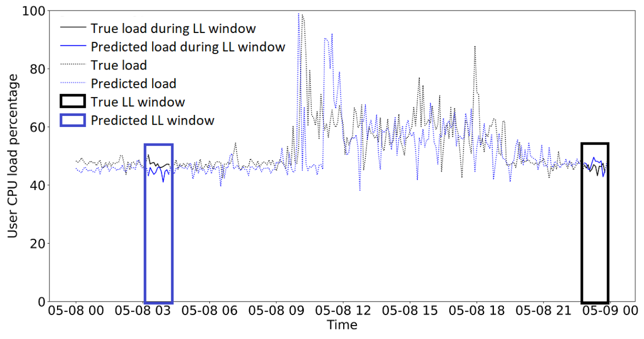


Figure 9: Correctly chosen LL window

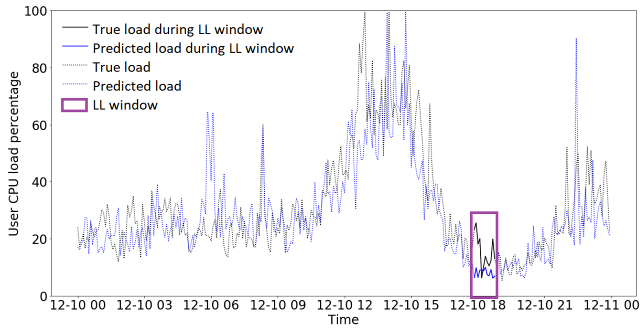


Figure 10: Low load prediction accuracy

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 Predictable Server

Backup Scheduling Problem. For each PostgreSQL or MySQL server s that is due for full backup on day d, our SEAGULL approach aims to correctly choose the LL window on day d and to accurately predict the load during this window (Definitions 2 and 8). These two metrics are orthogonal as illustrated by examples below.

In Figure 10, the true and predicted LL windows coincide. 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. Therefore, 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 described 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 compare these models with respect to their accuracy and scalability.

5.1 ML Models for Time Series Forecast

We now summarize the key ideas of the ML models that we considered to predict low customer activity per server on its backup day. These models range from the simple persistent forecast heuristic to the complex neural-network-based analytics. Below, we also discuss the applicability of each model to our data set that we analyzed in Section 3.

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 based on the average load per server during previous week. This variant can predict the load of stable servers (Definition 4). 53.5% of servers are stable (Figure 4). Previous week average is applicable to servers that existed longer than one week, which corresponds to 68% of servers.
- ***Previous equivalent day*** forecasts the load per server based on its load on previous equivalent day of the week. 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. Similarly to previous week average, previous equivalent day requires one week of history per server to make a prediction.
- ***Previous day*** variant is based on the load per server on previous day. Similarly to previous equivalent day, previous day is more powerful than previous week average, since it captures a daily pattern (Definition 5), including stable load. Thus, it is applicable to 53.7% of servers. In contrast to the other two variants of the persistent forecast model, previous day requires only one day of history per server to make a prediction, which covers 82% of servers. Given that previous day is suitable for the largest subset of servers, we focus on this variant of persistent forecast in our further analysis.

GluonTS: Gluon Time Series [9] is a highly optimized toolkit for probabilistic time series modeling, focusing on deep learning-based models. We train simple feed forward estimator on one week of data per server.

Prophet [10] is open source software released by Facebook. It is a procedure for forecasting 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. We enable daily and weekly seasonality and fit this model using one week of data per server.

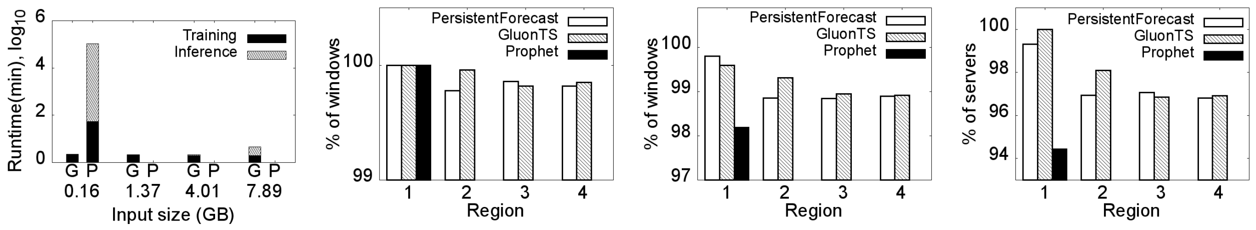
ARIMA: Auto-Regressive Integrated Moving Average model forecasts the future values of a series based on the different seasonal and temporal structures in the series. At inference, it predicts one signal at a time by fitting to this signal's prior values. We use a pmdarima [2] to automatically search for optimal values of six parameters in order to get an accurate forecast per server.

5.2 Experimental Comparison of ML Models

5.2.1 *Experimental Setup*

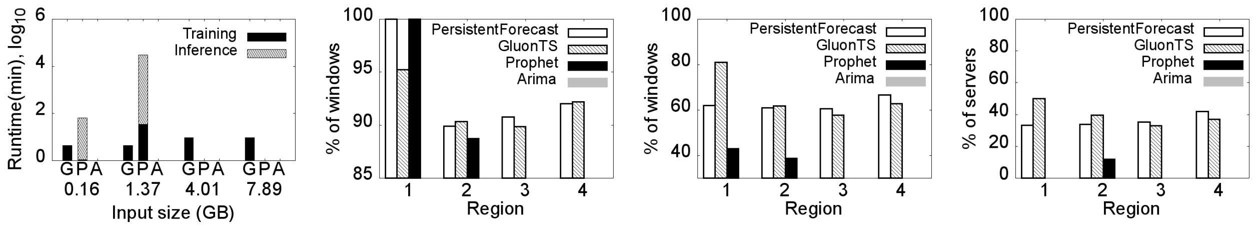
Hardware. We conducted all experiments on a VM running Ubuntu 18.04. It has 16 CPUs and 64GB of memory.

Input Data. Since the pipeline runs per Azure region once a week (Section 2), our input data is partitioned by



(a) Training and inference (b) LL windows (c) Load during LL windows (d) Predictable servers

Figure 11: Low load prediction for stable servers using Persistent Forecast, GluonTS (G), and Prophet (P)



(a) Training and inference (b) LL windows (c) Load during LL windows (d) Predictable servers

Figure 12: Low load prediction for unstable servers using Persistent Forecast, GluonTS (G), Prophet (P), and Arima (A)

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. These 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, GluonTS, Prophet, and Arima 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. There are total 23K of servers in four regions during four weeks. Servers with at least three days of history prior to their backup days, are considered. There are total 14K of such servers in four regions during four weeks.

Methodology. We implemented our Seagull pipeline in Python. Below, we compare two implementation variants: (1) Our base-line implementation is ***single-threaded***. Persistent forecast and GluonTS are efficient. Thus, they are implemented single-threaded. (2) Our ***multi-threaded*** Dask-based [8] implementation partitions the data per server and executes all computations per server concurrently. Since our VM has 16 cores, we used 16 workers with 4 threads per worker unless stated otherwise. Prophet and Arima are computation-intensive. They are implemented on Dask.

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. For Prophet and Arima that are implemented on Dask, we sum the time for training and inference for all threads.

5.2.2 *Training and Inference*

Persistent Forecast does not require training because it simply uses the load per server on the previous day as predicted load per server on the next day.

GluonTS scales well. Even though its implementation is single-threaded, it predicts load within a few minutes in all cases in Figures 11–12. Runtime for training and inference increases linearly with the growing number of servers (Figures 11(a) and 12(a)). For stable servers, training and inference terminate within 3 minutes. For unstable servers, training time ranges from 4 to 10 minutes, while inference time ranges from 0.2 to 16 seconds as the number of considered servers grows from 8 to 700.

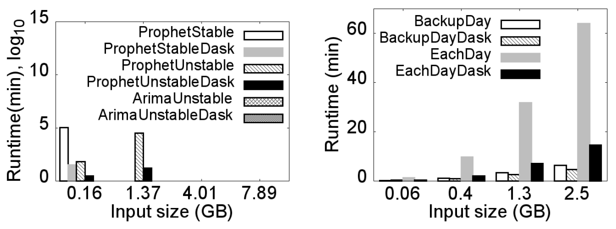
Prophet does not scale well. Its training time grows linearly from 1 to 55 minutes as the number of considered servers increases from 8 to 152 (Figures 11(a) and 12(a)). Inference is even more time-consuming. It takes from 1 to 32 hours. Thus, we implemented Prophet on Dask and achieved up to 60X speedup compared to single-threaded execution (Figure 13(a)). 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 within several hours. These measurements are not shown in Figures 11–13.

Arima TODO

5.2.3 *Low Load Prediction Accuracy*

For each ML model, we measure three low load prediction metrics for stable and unstable servers separately in four regions. The accuracy of low load predictions for stable servers ranges from 93 to 100% (Figures 11(b)–11(d)). As expected, it is much higher than for unstable servers that ranges from 0 to 100% (Figures 12(b)–12(d)).

While GluonTS is more accurate than persistent forecast heuristic in many cases, persistent forecast based on previ-



(a) Training and inference (b) Accuracy evaluation

Figure 13: Single-threaded versus multi-threaded Dask-based implementation

ous day has two practical advantages. One, it is applicable to the largest subset of servers (15K servers, i.e., 65% of all servers) since this heuristic does not require one week of training data per server. Two, it does not introduce any computational delay due to training and thus scales even better than GluonTS. For our sample data set (Section 5.2.1), persistent forecast correctly selected 99% of LL windows, accurately predicted the load during 96% of all windows, and classified 75% of long-lived servers as predictable. Based on these results, we deployed persistent forecast to production to predict low load per server.

5.2.4 *Prediction Accuracy Evaluation*

In this section, we use the same settings as in production. Namely, the load per server was predicted by persistent forecast based on previous day. Each input data set contains the load of all servers during one week in one Azure region.

Backup Day. Figure 13(b) shows the runtime of single-threaded (BackupDay) and multi-threaded (BackupDayDask) 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 partitionalbe 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 (Section 7). In Figure 13(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 (EachDay) runs for over 1 hour. The multi-threaded implementation (EachDayDask) 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

Load Prediction on a cluster for optimized resource allocation has become a popular research direction in the recent years. Existing approaches focus on predicting survivability of cloud databases for optimized resource provisioning [19], idle time detection for database quiescing and overbooking [18, 21], database workload prediction for database consolidation [14], VM workload prediction for oversubscribing servers [12], and demand-driven auto-scale of resources [15, 16, 20]. None of these approaches focused on predicting low load windows for optimized scheduling of system maintenance operations. Thus, these approaches neither analyze the large-scale production telemetry, nor define low load prediction accuracy, nor compare several ML models from the perspective of low load prediction. Moreover, these approaches did not focus on design, implementation, and deployment of a generic infrastructure for load prediction that is scalable to tens of thousands of servers in production.

Job Scheduling Algorithms. Several job scheduling algorithms were proposed to, for example, enable reservation-based scheduling [13], enforce Service Level Objective and mitigate performance variance [17]. 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.

7. CONCLUSIONS AND FUTURE WORK

We built the SEAGULL infrastructure for predicting the utilization of resources (I/O, memory, disk) of any system component (database, server, VM, node on a cluster) for several projects aiming to optimize resource allocation. The SEAGULL infrastructure is deployed to production worldwide to schedule full backups of tens of thousands of PostgreSQL and MySQL servers such that these backups do not collide with high customer CPU load. Backups of 33% of servers are now rescheduled to run during predicted low load windows. The number of busy servers for which backups collide with peak customer activity is now reduced by 50%.

Going forward, we will generalize our SEAGULL approach in several ways. One, we will include other metrics and apply other ML models to improve load prediction accuracy. Two, we will apply these techniques to other Azure databases to optimize scheduling of system maintenance operations. Lastly and most importantly, we will use the SEAGULL infrastructure to optimize resource allocation in several follow-up projects including preemptive demand-driven load balancing and auto-scale of resources on Azure clusters.

Acknowledgements

We thank Siqi Liu, Purnesh Dixit, Santhosh Pillai, Akshaya Annavajhala, Larry Franks, and Chris Lauren for multiple fruitful discussions about ML models and AML pipelines. We are also grateful to Hiren Patel for helping us with large-scale telemetry analysis.

8. 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] Prophet. https://facebook.github.io/prophet/.

[11] E. Breck, N. Polyzotis, S. Roy, S. E. Whang, and M. Zinkevich. Data validation for machine learning. In *SysML*, 2019.

[12] 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.

[13] 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.

[14] C. Curino, E. P. Jones, S. Madden, and H. Balakrishnan. Workload-aware database monitoring and consolidation. In *SIGMOD*, page 313–324, 2011.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] J. Picado, W. Lang, and E. C. Thayer. Survivability of cloud databases - factors and prediction. In *SIGMOD*, page 811–823, 2018.

[20] 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.

[21] 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.