Proactive Resource Allocation Policy for Microsoft Azure Cognitive Search

Olga Poppe, Pablo Castro, Willis Lang, and Jyoti Leeka
olpoppe|pablocas|wilang|jyleeka@microsoft.com

ABSTRACT

Modern cloud services aim to find the middle ground between quality of service and operational cost efficiency by allocating resources if and only if these resources are needed by the customers. Unfortunately, most industrial demand-driven resource allocation approaches are reactive. Given that scaling mechanisms are not instantaneous, the reactive policy may introduce delays to latency-sensitive customer workloads and waste operational costs for cloud service providers. To solve this catch-22, we define the proactive resource allocation policy for Microsoft Azure Cognitive Search. In addition to the current resource demand, the proactive policy takes the typical resource usage patterns into account. We gained the following valuable insights from these patterns over several months of production workloads. One, 87% of the workload is stable due to continuous resource demand. Two, 90% of varying demand is predictable based on a few weeks of historical traces. Three, resources can be reclaimed 52% of the time due to extensive idle intervals of varying workload. Given the size and scope of our analysis, we believe that our approach applies to any latency-sensitive cloud service.

1. INTRODUCTION

We are currently witnessing the growing popularity of demand-driven resource allocation among all cloud service providers, including Microsoft Azure [3], Amazon Web Servoces [7], and Google Cloud Platform [4]. As a result, the customers do not have to provision a fixed amount of resources up front. Instead, they send workloads to the systems that dynamically and transparently manage resources to handle the changing demand over time. These systems deploy low-latency resource allocation mechanisms. While the workload is running, resources are allocated. When the workload stops, resources are reclaimed and possibly reused to serve current workloads. In this way, demand-driven resource allocation reduces the amount of maintained resources and thus saves the operational costs.

The natural progression for demand-driven resource allocation is proactive decision making. In addition to the current demand, proactive decisions take the predicted future demand into account to allocate resources ahead of demand. The proactive policy improves the quality of service by reducing delays due to the reaction time between demand signal and effective change in resource availability [23, 25, 27, 30]. Furthermore, the proactive policy allows to re-target resources during predicted long idle intervals to serve the current workloads.

We define the proactive policy for Microsoft Azure Cognitive Search, which is a PaaS solution that allows to build sophisticated search capabilities within customer applications on customer data [1, 2]. It is a rapidly growing cloud service which currently runs tens of billions of queries per month. It offers all the functionality needed to create rich search scenarios such as automatic content ingestion, fast full-text search, auto-complete, customizable scoring, and a natural language understanding stack that ensures high relevance of search results.

Challenges. Defining the proactive policy for any latency-sensitive cloud service is not trivial.

(1) *High latency sensitivity.* Search is highly sensitive to execution latency. Currently, low latency is achieved through a combination of index data structure design, a favorable index size-memory ratio, and by maintaining warm compute and cache to execute search queries as soon as they arrive. However, current solution results in low efficiency of compute utilization when search requests are not continuous. We aim to move the efficiency needle without sacrificing the low latency requirement.

(2) *Benefit versus overhead of proactive policy.* Proactive resource allocation is not always applicable. For example, if the demand is stable or idle time is too fragmented for effective reuse of resources, then the resources must be provisioned to guarantee high quality of service. A proactive policy will

do more harm than good in such cases due to its overhead of load prediction. We aim to identify the prerequisites of the proactive policy to exclude part of the workload from further consideration.

(3) *Wide range of applicable techniques.* There are many approaches to load prediction for proactive resource allocation. They range from simple statistics to complex machine learning models. Each of them has multiple tunable parameters. Each of them has advantages with respect to prediction accuracy, execution latency, and supportability long term in production worldwide. The search space is too large to be explored exhaustively. We aim to compare several techniques and propose an effective approach to proactive resource alloction.

State-of-the-Art. Several existing approaches implement reactive demand-driven resource allocation [10, 11, 12]. They might cause delays in resource availability after long idle intervals during which resources are reclaimed. Thus, the reactive policy is not suitable for latency-sensitive cloud services. To optimize quality of service, we apply proactive resource allocation policy. While academic approaches leverage complex and computationally expensive machine learning models to predict the load [8, 13, 14, 15, 20, 26, 29], industrial approaches deploy light-weight yet accurate forecast techniques to production [17, 19, 21, 22, 23, 25, 27, 30]. Unfortunately, the existing approaches fail to identify cases when the proactive policy is not applicable. Hence, they introduce significant computational overhead of load prediction to latency-sensitive cloud services. To avoid this overhead, our approach identifies the prerequisites of the proactive policy and focuses on cases when the proactive policy saves operational costs without sacrificing high quality of service.

Proactive Resource Allocation Policy. We analyzed several months of production telemetry for Microsoft Azure Cognitive Search and concluded that no single resource allocation policy is effective for all workloads. Instead, the policy must be tailored for each type of workload. Therefore, we first characterize the search requests along various dimensions, including stability, predictability, and fragmentation of idle time. Afterwards, we identify the prerequisites of proactive resource allocation. In particular, resources are proactively allocated for varying predictable workloads to guarantee high quality of service. In addition, resources are reclaimed and reused during extensive idle intervals to save operational costs. Furthermore, resources are provisioned for stable workloads to avoid the overhead of load prediction. Lastly, resources are allocated reactively for unpredictable workloads.

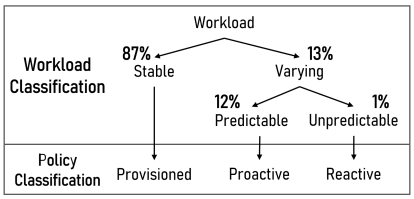


Figure 1: Workload and policy classification

Contributions. Our key contributions include:

(1) We classify the workload of search requests into stable and varying, predictable and unpredictable (Figure 1). 87% of the workload is stable. 12% is varying and predictable. Only 1% is varying and unpredictable. We study the fragmentation of idle time. 63% of the total idle time is composed of idle intervals that exceed two hours.

(2) We define the spectrum of resource allocation policies, including provisioned, reactive, proactive, and optimal. We review the range of demand prediction techniques from simple probabilistic algorithms to advanced time series forecast models.

(3) We identify five prerequisites of the proactive policy and apply it to predictable varying workload with extensive idle intervals. In this way, we exclude 87% of the workload and thus reduce the execution latency of the proactive policy by 2X compared to the proactive policy applied to the entire workload.

(4) We define the KPI metrics that measure the effectiveness of the resource allocation policies in addressing the business needs for various workloads. In particular, we measure demand predictability, quality of service, correctness of resource allocation, operational cost efficiency, and execution latency.

(5) We experimentally compare the resource allocation policies. In contrast to the reactive policy, the proactive policy correctly predicts 90% of varying search requests to guarantee high quality of service. In contrast to the provisioned policy, the proactive policy reclaims resources 52% of the time for varying workload to save operational costs.

Outline. We define the optimization objective in Section 2. We identify the prerequisites of the proactive policy in Section 3. We define the resource allocation policies in Section 4 and the accuracy metrics in Section 5. We present the experiments in Section 6. We review the related work in Section 7 and conclude the paper in Section 8.

2. OPTIMIZATION OBJECTIVE

Index. To speed up search requests, the search engine builds indexes and allocates resources per index. Let $\mathbb{I}$ be the set of indexes.

Time is represented by a linearly ordered set of

time points $(\mathbb{T}, \leq)$ where $\mathbb{T} \subseteq \mathbb{R}^+$ are the non-negative real numbers.

Resource Demand $D : \mathbb{I} \times \mathbb{T} \rightarrow \{0, 1\}$ is a function that maps an index $i \in \mathbb{I}$ and a time point $t \in \mathbb{T}$ to a binary value indicating whether the resources of i are needed at t. $\forall i \in \mathbb{I}\ \forall t \in \mathbb{T}$ if the resources of i are needed at t then $D(i,t) = 1$, else $D(i,t) = 0$.

Predicted resource demand, denoted as $P(i,t)$, is defined analogously to the actual demand $D(i,t)$.

Resource Allocation $A : \mathbb{I} \times \mathbb{T} \rightarrow \{0, 1\}$ is a function that maps i and t to a binary value indicating whether the resources are allocated for i at t.

Quality of Service (QoS) is measured as the ratio of the time when the resources are needed and allocated $T_{na}(i) = \{t \in \mathbb{T} \mid D(i,t) > 0 \text{ and } A(i,t) > 0\}$ to the time when the resources are needed $T_n(i) = \{t \in \mathbb{T} \mid D(i,t) > 0\}$ per index.

$$QoS = \sum_{i \in \mathbb{I}} T_{na}(i) \Big/ \sum_{i \in \mathbb{I}} T_n(i) \tag{1}$$

If the resources are always available when they are needed, then QoS equals to 1. If the resources become available after a delay, then QoS is lower.

Operational Cost Efficiency is measured as the ratio of the time when the resources are needed and allocated to the time when the resources are allocated $T_a(i) = \{t \in \mathbb{T} \mid A(i,t) > 0\}$ per index.

$$Eff = \sum_{i \in \mathbb{I}} T_{na}(i) \Big/ \sum_{i \in \mathbb{I}} T_a(i) \tag{2}$$

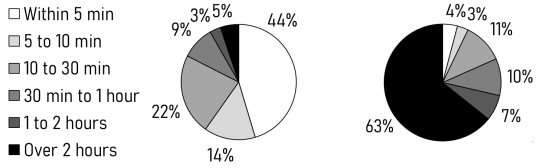
If the resources are only allocated when they are needed, then efficiency equals to 1. If the resources stay idle, then efficiency is lower.

Optimization Objective. An optimal policy allocates resources if and only if they are needed. Figure 3(d) illustrates this policy. Then, QoS and efficiency are equal to 1. An optimal policy requires a perfect demand prediction, which is hard to achieve in practice due to varying workloads. Nevertheless, the ultimate goal of a resource allocation policy is to maximize both QoS and efficiency, while keeping the execution latency low to guarantee real-time response of a latency-sensitive cloud service.

3. PREREQUISITES OF PROACTIVE RESOURCE ALLOCATION

3.1 Fine-Grained Production Telemetry

To guarantee accurate load prediction, the production telemetry must be fine-grained, cover several months, and contain all features that can be useful for prediction. We analyze two months of production telemetry in one Azure region where tens of thousands of indexes are currently deployed. Each event carries a timestamp in milliseconds, an index identifier, and a subscriber identifier.



(a) Number of idle intervals (b) Duration of idle time

Figure 2: Fragmentation of idle time

3.2 Varying Resource Demand

The resource demand of an index $i \in \mathbb{I}$ is stable if $\forall t \in \mathbb{T}\ D(i,t) = 1$. Otherwise, the demand of i is varying. Resources are provisioned to stable indexes to guarantee high QoS and efficiency, while avoiding the latency of load prediction. 32% of indexes receive requests every few minutes. 87% of requests belong to them (Figure 1).

Resources can be shared among indexes with varying demand. We focus on optimizing resource allocation for these indexes below. 68% of indexes have varying demand. 13% of requests belong to them.

3.3 Extensive Idle Intervals

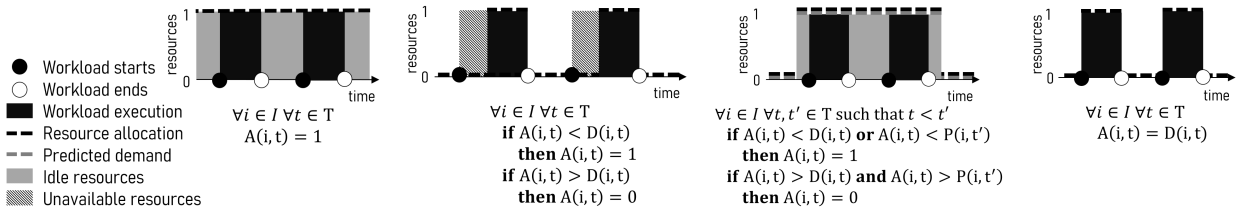
While the number of requests per index and hour can reach several hundreds, indexes receive requests only 18% of the time and stay idle 82% of the time on average. While 44% of idle intervals are within 5 minutes (Figure 2(a)), their total duration contributes only 4% to the total idle time (Figure 2(b)). Resources are not reclaimed during such short idle intervals to relieve the backend from the overhead of frequent scaling operations [23]. Even though only 5% of idle intervals exceed two hours, the total duration of these intervals contributes 63% to the total idle time. Resources can be effectively reused during such extensive idle intervals.

3.4 Accurate Demand Prediction

The proactive resource allocation policy relies on highly accurate workload prediction. Predicted long duration of idle intervals enables resource reclamation. Predicted start of customer workload enables resource allocation ahead of demand.

3.5 Low-Latency Prediction and Scaling

Low-latency workload prediction and scaling mechanisms are indispensable for the effectiveness of proactive resource allocation. Computationally expensive predictions and slow mechanisms reduce the time intervals during which resources can be reused. Worst yet, they introduce delays and jeopardize high



(a) Provisioned policy (b) Reactive policy (c) Proactive policy (d) Optimal policy

Figure 3: Resource allocation policies

quality of service for unpredictable workloads for which resources have to be allocated reactively.

4. RESOURCE ALLOCATION POLICIES

4.1 Base-Line Policies

Provisioned Policy always allocates resources, independently from demand. Figure 3(a) illustrates this policy. Analysis of historical load traces reveals that resources stay idle (i.e., $D(i,t) < A(i,t)$), unless the customer manually de-allocates resources during idle intervals [9, 10, 17, 19, 22, 30]. Manual resource allocation is labor-intensive, time-consuming, error-prone, neither scalable, nor durable.

Reactive Policy allocates resources based on the current demand [10, 11, 12]. Figure 3(b) illustrates this policy and defines its algorithm.

Unfortunately, resource scaling mechanisms are not instantaneous. Therefore, resources may be unavailable (i.e., $D(i,t) > A(i,t)$) when the workload starts in Figure 3(b). These delays make the reactive policy less suitable for latency-sensitive applications than the provisioned policy.

To reduce operational costs, the reactive policy reclaims resources when the workload stops. However, if idle intervals are short, then resource availability time is too fragmented for effective reuse. Worst yet, frequent scaling operations may introduce a significant backend overhead.

4.2 Proactive Policy

The proactive policy analyzes the historical resource usage patterns, predicts future demand, and allocates resources based on both current and predicted demand [23, 25, 27, 30]. Figure 3(c) illustrates this policy and defines its algorithm.

On the up side, the proactive policy reduces or even avoids delays in resource availability when the workload starts compared to the reactive policy (compare Figures 3(b) and 3(c)). Furthermore, the proactive policy relieves the backend from frequent scaling operations due to short idle intervals.

On the down side, idle time might increase compared to the reactive policy since resources are allocated ahead of demand and thus not immediately used by the customers. Also, in contrast to the base-line policies, the proactive policy introduces the computational overhead of demand prediction.

4.3 Demand Prediction Techniques

Any demand prediction algorithm can be plugged in the proactive policy. We now briefly summarize the range of commonly used techniques in industry.

Persistent forecast algorithm looks up the demand on previous day (or weekday), assumes that it stays the same on the following day (or weekday), and makes proactive decisions per index [22, 23].

Probabilistic algorithm analyzes historical traces and computes probability of requests per index and time window. Resources are proactively allocated during a window if the probability exceeds a threshold during the window. Resources are reclaimed once the workload stops and the probability falls below the threshold. Proactive resource allocation decisions can also leverage any other statistics, for example, count of requests [19, 23, 25, 27, 30].

Predictive algorithm trains a machine learning model on historical traces, predicts the demand, and makes proactive decisions based on both current and predicted resource demand per index. Time series forecast models, linear regression, exponential smoothing, classification models, and Neural Networks are commonly used [21, 22, 23, 27].

5. ACCURACY METRICS

While the standard metrics (such as hinge loss) allow to measure the overall accuracy of resource demand prediction, they provide little insight into the effectiveness of a policy in addressing the business needs. Table 1 summarizes the accuracy metrics for in-depth evaluation of the policies [23, 30].

Demand Predictability. To measure QoS, we differentiate between predictable and unpredictable demand. Resource demand for an index i at a time point t is predictable if $D(i,t) = 1$ and $P(i,t) = 1$. It is unpredictable if $D(i,t) = 1$ and $P(i,t) = 0$.

Correctness of Resource Allocation. To measure

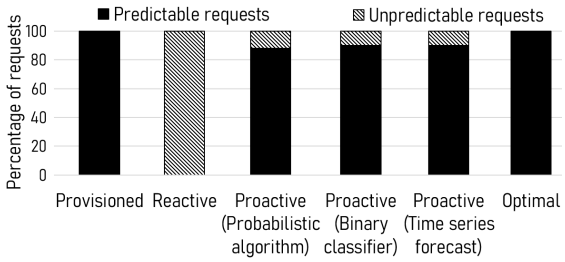


Figure 4: Demand predictability

Resource allocation	Conditions
Correctly allocated due to predictable demand	$P(i,t) = A(i,t) = D(i,t) = 1$
Correctly de-allocated (reclaimed)	$P(i,t) = A(i,t) = D(i,t) = 0$
Wrongly allocated (idle)	$P(i,t) = A(i,t) = 1$ but $D(i,t) = 0$
Wrongly de-allocated due to unpredictable demand	$P(i,t) = A(i,t) = 0$ but $D(i,t) = 1$

Table 1: Accuracy metrics

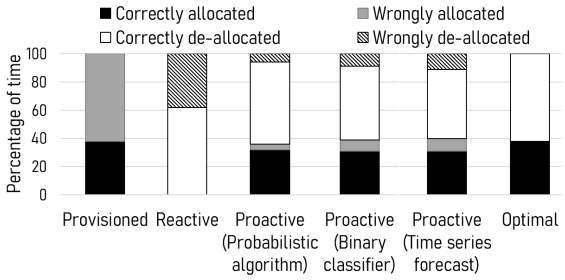


Figure 5: Correctness of resource allocation

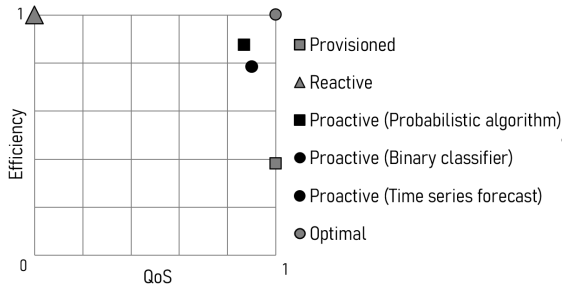


Figure 6: Quality of service versus efficiency

efficiency, we differentiate between the time intervals when the resources are correctly or wrongly allocated or de-allocated. Resource allocation for an index i at a time point t is correct if $D(i,t) = 1$. Analogously, de-allocation of resources of an index i at a time point t is correct if $D(i,t) = 0$.

Impact of Resource Allocation Decisions. If demand is predictable, then the resources are correctly allocated and high QoS is guaranteed. If demand is unpredictable, then the resources are wrongly de-allocated. In this case, resources must be allocated reactively. Thus, low-latency scaling mechanisms are indispensable even for the proactive policy.

While the resources are correctly de-allocated, they can be reclaimed to improve efficiency without sacrificing high QoS. While the resources are wrongly allocated, they stay idle and efficiency suffers. This waste of operational costs can be mitigated by the high accuracy of load prediction.

6. EXPERIMENTAL EVALUATION

6.1 Experimental Setup

Input Data contains two month of search requests for several thousands of indexes in one Azure region.

Hardware. We run all experiments on a Windows 11 machine with 10 Intel 2.80GHz CPUs, 128GB RAM, and 3TB SSD.

Methodology. We implemented the workload classification, the accuracy evaluation, and the probabilistic algorithm in Python 3.7. We use the existing libraries for ML.NET binary classifier [5] and the time series forecast model NimbusML [6]. We train these models based on three weeks of history and predict search requests one day ahead per index.

Metrics. We measure accuracy per Section 5 and execution latency. We run each latency experiment ten times and report the average result.

6.2 Quality of Service versus Efficiency

We consider indexes with varying workload in Figures 4–6. We exclude indexes with stable workload since the provisioned policy is the most effective for them (Figure 1). We omit the request processing time since it is the same for all policies (shown as black area in Figure 3).

Provisioned Policy is one extreme of the spectrum. Its QoS is optimal since resources are always allocated. However, resources stay idle 62% of the time in Figure 5 and efficiency is 0.38 in Figure 6.

Reactive Policy is the opposite extreme of the spectrum. On the up side, resources are de-allocated once the workload stops and efficiency is optimal. On the down side, resources are always wrongly de-allocated when the workload starts and QoS suffers.

Optimal Policy is the third extreme of the spectrum. It makes no mistakes in resource allocation. Thus, both QoS and efficiency are equal to 1.

Proactive Policy can leverage any demand prediction technique. We compare the probabilistic algorithm, ML.NET binary classifier, and the time series forecast model NimbusML.

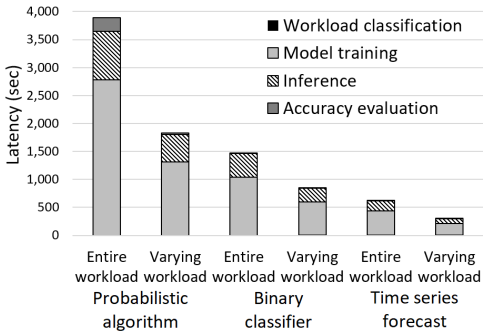


Figure 7: Execution latency

The accuracy of the binary classifier and the time series forecast model is quite similar. They correctly predict 90% of requests (Figure 4) and correctly allocate resources 31% of the time (Figure 5). Due to 10% of unpredictable requests, 10% of the time resources are wrongly de-allocated. 52% of the time resources are correctly de-allocated. Only 8% of the time resources are wrongly allocated.

As Figure 6 illustrates, these ML models achieve high QoS score of 0.9 at the cost of slightly lower efficiency score of 0.78 than the probabilistic algorithm. However, the probabilistic algorithm is also quite accurate. It correctly predicts 88% of requests and achieves QoS and efficiency scores of 0.87. Based on the results in Figures 4–6, we conclude that the proactive policy finds a reasonable balance between QoS and operational cost efficiency.

6.3 Execution Latency

In Figure 7, we compare the execution latency of the probabilistic algorithm, ML.NET binary classifier, and the time series forecast model NimbusML for the entire workload and the varying workload. Each bar is broken down into the following four sections: (1) Workload classification into stable and varying, (2) Model training based on three weeks of historical traces, (3) Inference one day ahead per index, and (4) Accuracy evaluation per Figures 4–6. The latency of workload classification is within one second and thus not visible at the scale of Figure 7.

Per our workload analysis in Section 3.2, 32% of indexes have stable resource demand and receive 87% of all search requests (Figure 1). Excluding such large portion of the workload reduces the latency by 2X for all models in Figure 7.

The latency of time series prediction using NimbusML is the lowest compared to other models in Figure 7. Its training is 3 minutes for all indexes with varying workload, while accuracy evaluation is 30 milliseconds for all indexes with varying workload. The average inference latency per index with varying workload is 110 milliseconds. Based on the results in Figures 6 and 7, we conclude that NimbusML is an accurate and light-weight model.

7. RELATED WORK

Demand-driven allocation of resources in the cloud has recently become a popular research direction [10, 11, 12, 13, 14, 16, 20, 26, 28, 29]. Some of these approaches are reactive [10, 11, 12]. A reactive policy reclaims resources once the customer workload stops, to save operational costs. Thus, quality of service may be jeopardized due to delays in resource availability when customer workload starts. In contrast, our approach makes proactive decisions based on recently observed resource usage patterns.

There are approaches to load analysis [9, 16, 18, 24] and load prediction using machine learning models [8, 13, 14, 15, 20, 26, 29]. Some of these models are computationally expensive and unintuitive for non-experts [19, 22, 27]. Thus, industrial approaches tend to deploy simple and light-weight forecast techniques to production [17, 19, 21, 22, 23, 25, 27, 30]. However, even light-weight techniques introduce the overhead of detailed continuous workload analysis which can be avoided in many cases. This is the approach we followed in this paper.

8. CONCLUSIONS AND FUTURE WORK

We define and evaluate the proactive resource allocation policy for Microsoft Azure Cognitive Search. We classify the workload of search requests and tailor the policy for each type of the workload. Such tailored approach significantly improves the quality of service, operational cost efficiency, and execution latency compared to other policies.

In this paper, we focus on the binary problem of proactive allocation and de-allocation of resources. Our solution caters to the application at hand and is a stepping stone towards a more general problem of proactive auto-scale of resources to any percentage of capacity. Solving this general problem is a subject for future research.

This paper presents the pre-deployment evaluation of the proactive policy. The post-deployment evaluation is subject for future work. It will measure how much operational costs (i.e., physical machines) are indeed saved by the proactive policy. It will depend on the effectiveness of placement policies which aim to place indexes that receive requests during mutually exclusive time intervals on the same physical machine to facilitate resource sharing among them. In-depth analysis of effective index placement is a subject for a follow-up publication.

REFERENCES

[1] Azure Cognitive Search. https://azure.microsoft.com/en-us/services/search/, 2023.

[2] Azure Cognitive Search Documentation. https://docs.microsoft.com/en-us/azure/search/search-what-is-azure-search, 2023.

[3] Azure SQL Database Serverless. https://docs.microsoft.com/en-us/azure/azure-sql/database/serverless-tier-overview, 2023.

[4] Google Serverless Computing. https://cloud.google.com/serverless, 2023.

[5] ML.NET Binary Trainer. https://docs.microsoft.com/en-us/dotnet/api/microsoft.ml.trainers.fasttree.fastforestbinarytrainer, 2023.

[6] NimbusML. https://docs.microsoft.com/en-us/python/api/nimbusml/nimbusml.timeseries.ssaforecaster, 2023.

[7] Serverless on AWS. https://aws.amazon.com/serverless/, 2023.

[8] 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.

[9] 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*, page 153–167, 2017.

[10] 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.

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

[12] 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.

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

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] J. Moeller, Z. Ye, K. Lin, and W. Lang. Toto - Benchmarking the Efficiency of a Cloud Service. In *SIGMOD*, pages 2543–2556, 2021.

[20] 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.

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

[22] O. Poppe, T. Amuneke, D. Banda, A. De, A. Green, M. Knoertzer, E. Nosakhare, K. Rajendran, D. Shankargouda, M. Wang, A. Au, C. Curino, Q. Guo, A. Jindal, A. Kalhan, M. Oslake, S. Parchani, V. Ramani, R. Sellappan, S. Sen, S. Shrotri, S. Srinivasan, P. Xia, S. Xu, A. Yang, and Y. Zhu. Seagull: An Infrastructure for Load Prediction and Optimized Resource Allocation. *Proc. VLDB Endow.*, 14(2):154–162, 2020.

[23] O. Poppe, Q. Guo, W. Lang, P. Arora, M. Oslake, S. Xu, and A. Kalhan. Moneyball: Proactive Auto-Scaling in Microsoft Azure SQL Database Serverless. *Proc. VLDB Endow.*, 15(6):1279–1287, 2022.

[24] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch. Heterogeneity and Dynamicity of Clouds at Scale: Google Trace Analysis. In *SOCC*, pages 1–13, 2012.

[25] F. Romero, G. I. Chaudhry, I. Goiri, P. Gopa, P. Batum, N. J. Yadwadkar, R. Fonseca,

C. Kozyrakis, and R. Bianchini. Faa$T: A Transparent Auto-Scaling Cache for Serverless Applications. In *SoCC*, pages 122–137. ACM, 2021.

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

[27] M. Shahrad, R. Fonseca, I. Goiri, G. Chaudhry, P. Batum, J. Cooke, E. Laureano, C. Tresness, M. Russinovich, and R. Bianchini. Serverless in the Wild: Characterizing and Optimizing the Serverless Workload at a Large Cloud Provider. In A. Gavrilovska and E. Zadok, editors, *USENIX*, pages 205–218. USENIX Association, 2020.

[28] Z. Shen, S. Subbiah, X. Gu, and J. Wilkes. CloudScale: Elastic Resource Scaling for Multi-tenant Cloud Systems. In *SOCC*, pages 1–14, 2011.

[29] 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.

[30] 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.