Predictive Provisioning: Efficiently Anticipating Usage in Azure SQL Database

Lalitha Viswanathan
University of Wisconsin-Madison
lviswanathan@wisc.edu

Bikash Chandra*
IIT Bombay
bikash@cse.iitb.ac.in
*while an intern at Microsoft

Willis Lang
Microsoft Gray Systems Lab
wilang@microsoft.com

Karthik Ramachandra
Microsoft Gray Systems Lab
karam@microsoft.com

Jignesh M. Patel
University of Wisconsin-Madison
jignesh@cs.wisc.edu

Ajay Kalhan
Microsoft
ajayk@microsoft.com

David J. DeWitt+
MIT
david.dewitt@outlook.com
+work done while at Microsoft

Alan Halverson
Microsoft Gray Systems Lab
alanhal@microsoft.com

***Abstract*—Over-booking cloud resources is an effective way to increase the cost efficiency of a cluster, and is being studied within Microsoft for the Azure SQL Database service. A key challenge is to strike the right balance between the potentially conflicting goals of optimizing for resource allocation efficiency and positive user experience. Understanding when cloud database customers use their database instances and when they are idle can allow one to successfully balance these two metrics. In our work, we formulate and evaluate production-feasible methods to develop idleness profiles for customer databases. Using one of the largest data center telemetry datasets, namely Azure SQL Database telemetry across multiple data centers, we show that our schemes are effective in predicting future patterns of database usage. Our methods are practical and improve the efficiency of clusters while managing customer expectations.**

I. Introduction

The cloud services enterprise is a tight-margin business where single digit percentage swings in cost and/or revenue mean the difference between an operating profit and an operating loss. Cloud service providers pay hundreds of millions of dollars per data center (this is just the initial investment), and the key to succeeding in the cloud business is to efficiently manage cluster resources in data centers while maintaining a positive customer experience. For instance, Microsoft operates 31 data centers[1] around the world, which requires investing billions of dollars in CAPEX costs that must be efficiently leveraged to generate revenue. Consequently, there is immense industry and academic interest in studying how to efficiently deploy and assign resources to cloud customers.

Most prior studies (see Section VI) are based on the premise that over-booking a cluster's resources is a simple and effective way to increasing cluster efficiency. Over-booking improves cluster productivity by matching the available/provisioned computing resources (e.g., CPU) to the actual use by customer workloads, thereby allowing a cloud provider to service the same workload with fewer resources. However, when over-booking, there is a danger of under-provisioning, which can hurt the quality of service. For example, if two workloads are over-provisioned on a single node that can support only one active workload, then the service can suffer if both workloads are simultaneously active.

[1]At the time we prepared this manuscript.

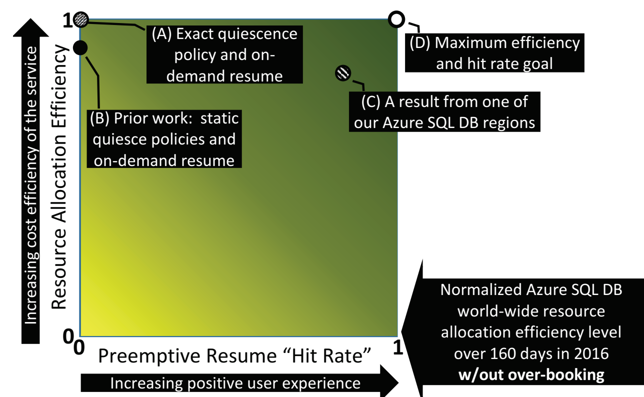

Fig. 1. Our optimization space for over-booking strategies. Our methods balance the cost efficiency of the service and user satisfaction/availability. A *quiescing mechanism* is used to reclaiming resources from a database, and a *resume mechanism* is used to assign back resources to a reclaimed database.

One approach to address this *multi-tenant provisioning* problem is to use telemetry data from production operations, and to develop models that identify patterns in service usage. Then, these patterns can be used to *predict* future accesses to determine when and how to over-book. This paper develops a variety of such prediction models, and then uses an actual trace collected from all of Microsoft Azure SQL Database data centers to evaluate the proposed models. To the best of our knowledge, this is the first work on this problem of *predictive provisioning* that uses such a large and diverse collection of actual production telemetry data.

In this paper, we focus on a specific and previously-proposed coarse-grained over-booking approach for Database-as-a-Service (DBaaS) clusters [8]. This strategy takes advantage of SQL Database's shared-disk architecture that is employed for certain database tiers. With this strategy, when a database is idle for a prolonged period, the database is "detached" from the database engine process (which is a process running an instance of SQL Server), thereby freeing up resources for other databases. This operation essentially *quiesces* the database. In this paper, we define a *resource allocation efficiency* metric for this previously proposed over-booking mechanism as follows: if a database is quiesced every time that it is idle, then the resource allocation efficiency for this database is 100%. If

2375-026X/17 $31.00 © 2017 IEEE
DOI 10.1109/ICDE.2017.155

IEEE computer society

resources are continuously allocated to the database while it is idle, then the efficiency for this database is 0%. (This metric is discussed in more detail in Section III.)

In Figure 1, we show a plot of two optimization metrics. The first metric, plotted on the y-axis is the *resource allocation efficiency*. We will discuss the other metric shortly. On this plot, the y-intercept represents the normalized resource allocation efficiency, *without over-booking*, aggregated over all databases across all of the worldwide regions that operate the SQL Database service in a 160 day period in 2016. The absolute allocation efficiency across the entire service (not shown) is low, motivating the need for an efficient over-booking method.

In prior work [8], the authors describe a static quiescing policy that waits for a fixed length of idle time before quiescing the (database) service. If we consider the extreme case when the idle time policy parameter asymptotically approaches zero, then this represents a perfect scheme as no resources are allocated to any database instance that is idle. We illustrate this scenario in Figure 1 with the point at the upper left of the plot (point A). In this previous work, the idle time policy parameter is set to a non-zero value, such as three hours, and corresponds to the point B shown in Figure 1. Unfortunately, this previous approach may cause poor user experience, as when the user connects to a quiesced database, the user will likely suffer a brief but noticeable period of unavailability while that database instance is being resumed. We call this scenario "Resume On-Demand," and it can cause negative user experience and lower the quality of service. We quantify this quality of service metric on the x-axis, and we describe this metric next.

In this paper, we discuss how to improve the quality of service in this scenario by preemptively "re-attaching" the database files, and thus *resuming* the database instance before any actual user activity occurs. (Thus, the user will not experience a degradation in service as the database instance will be up and running when the user queries are fired.) We use a measure called the *preemptive resume hit rate* that is the ratio of the number of actual preemptive resume events over the optimal number that is required so that users never encounter a preempted queisced database when a query is actually fired. This resume hit rate metric is plotted on the x-axis in Figure 1.

Putting everything together, our problem is that of *predictive provisioning*, and can be stated as follows: For a given database instance, we wish to leverage an idleness model to determine (a) when we should quiesce a database instance to maximize the resource allocation efficiency, and (b) when we should preemptively resume a database instance to maximize our resume hit rate. In Figure 1, point D indicates the hypothetical perfect solution that achieves the maximum resource allocation efficiency and the ideal hit rate.

Solving this problem is a challenging task, especially at scale. Microsoft has millions of databases hosted on its SQL Database platform. Fortunately, cloud providers such as Microsoft have un-ending streams of cloud telemetry that captures the activity of the databases. With this telemetry data, we can try to identify "patterns of idleness" in *each* database instance to build a predictive resource allocation solution. However, this vast amount of telemetry data presents many practical data analysis problems that must be solved to achieve our desired goal. Thus, we must look for computationally

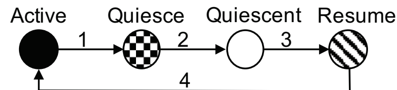

Fig. 2. Database state transition diagram for the quiesce/resume mechanism.

inexpensive analysis methods that can take advantage of the telemetry data, and can be run continuously on the telemetry data streams.

In our work, we focus on efficient methods to predict when we should quiesce an idle database instance, and when to preemptively resume the database instance before a user connects to it. In Figure 1, the point C highlights a result from our methods for a region running our SQL Database service that hosts hundreds of thousands of database instances. As illustrated by that summary result, the methods that we propose in this paper are able to dramatically increase the hit rate while trading off some efficiency.

To evaluate our methods, we use actual telemetry traces that cover every single deployed customer database. We show a number of simple, but highly effective methods that provide good allocation efficiency along with high resume hit-rates. In fact, point C in the figure came from a constant time update and look-up approach with a minimal memory footprint.

The contributions of our work are as follows:

- We formulate an important and practical optimization problem for over-booking cloud resources based on two metrics: resource allocation efficiency and preemptive resume hit rate.
- We describe and identify a number of operationally scalable and effective methods that leverage past database activity to tackle our optimization problem.
- We evaluate our methods over a 160 day production trace (to the best of our knowledge, the largest cloud production trace studied in the literature) over all of Microsoft Azure SQL Database regions, and find that our light-weight approaches are promising, and they provide a strong baseline for even more sophisticated approaches in the future.

II. Background

Microsoft Azure SQL Database: Microsoft Azure SQL Database [1], [2] is a public database service that is deployed using a Platform-as-a-Service (PaaS) model. The current version of this service is built on a split storage architecture with different database tiers: Premium or Standard/Basic. The premium tier databases store their data on SSD storage that is attached to the compute nodes, and target high-performance workloads. On the other hand, the Standard/Basic tier databases store their data files in a remote Azure Storage service. Since Standard/Basic tier databases are substantially cheaper than Premium tier databases, they constitute the vast majority of Azure SQL Databases. As such, efficient management of these databases is of critical interest to Microsoft.

Quiescing and over-booking: Our work focuses on the over-booking mechanism described in [8]. In that prior work, there are four basic states for an Azure SQL Database instance: active, quiescent, quiesce, and resume (see Figure 2). The latter

two states (quiesce and resume) represent the mechanism's transitioning states corresponding to the database being "detached" and "attached", respectively. During the quiescent state, the database is unavailable to the user, and the cloud provider is able to recoup/repurpose the allocated resources. On the other hand, in the active state, the database is available to the user, and resources are allocated regardless of whether the database is actually serving any active queries.

Prior work [8] used a simple policy that is based on the length of the idle period to make transition 1 in Figure 2. For example, a database may be queisced if it is observed to be idle for 12 hours. In this previous work, transition 4 was only made "on-demand", when a connection was attempted to a quiescent database. Thus, every transition 4 event could lead to a negative user experience. We improve upon this previous work by using Azure SQL Database telemetry to determine the transition triggers for each step in Figure 2 more intelligently.

Azure SQL Database telemetry: The analysis and evaluation that we perform in this paper uses Azure SQL Database telemetry that is emitted from each unique database (instance) from the time that the database is created to the time that it is dropped. For each database, telemetry streams capture events such as the attempts to establish connections, and database utilization levels (e.g., CPU, I/O, log, etc.) These utilization levels are captured at a subminute granularity. Prior work has described this telemetry dataset in more detail [7], [15].

Within Microsoft, we use this telemetry data to simulate different mechanisms and policies to evaluate their effectiveness on actual workload patterns at production scale. In this work, we follow the same approach and leverage this production telemetry to evaluate models for predictive provisioning.

III. Problem Formulation

Previous work [8] has shown that with an idleness-based quiescence policy, it is possible to reclaim substantial capacity even after accounting for the associated costs of enforcing the policy. The authors concluded that the costs associated with unavailability (due to resuming the database on-demand) start to dominate as we decrease the idleness policy parameter. Unavailability leads to SLA violations and subscription refunds that may nullify any efficiency gains from over-booking.

Furthermore, the static idleness policy (e.g., watching 12 idle hours go by to make a decision) itself is wasteful, but employed because of its simplicity. Reasoning about these costs, it is straightforward to observe that the cause of both these costs is the uncertainty in predicting the future activity (or idleness) of individual database instances.

Based on these observations, a natural question to ask is whether we can use telemetry data to identify patterns of idleness that can then be leveraged to preemptively resume databases before actual activity occurs. In other words, we pose the following ambitious problem: *Given the fine-grained telemetry data for each database, can we anticipate its activity patterns with sufficiently high accuracy so that we could dynamically quiesce and resume individual databases based on their activity patterns?*

As an approach, in this paper, we deliberately resist the urge to follow the fashionable trend of throwing an ensemble of machine learning methods at the problem. Rather, in this initial approach, we search for simple (but effective) models that are cheap to compute and intuitive to understand (and debug) when put into actual operation.

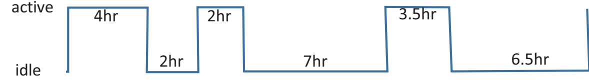

Fig. 3. Binary activity pattern for a database

Overall, our goal is to dynamically quiesce and resume databases using predictive strategies based on an activity-idleness model that is backed by telemetry data. We now define two metrics to make this formulation more concrete.

Resource allocation efficiency (E): Let T_i be the total time that a given database (instance) is idle during the entire time period under consideration. Let T_q be the total time it spent in the quiescent state, due to some strategy. Then the resource allocation efficiency E of a strategy with respect to this database is defined as the ratio of T_q over T_i. Aggregating over an entire Azure SQL Database region R, we get,

$$E = \frac{\sum_R T_q}{\sum_R T_i} \tag{1}$$

A solution that gives 100% efficiency is one that quiesces the database perfectly, so that there is no idle time. This operating point is not attainable in practice. However, we aim to get as close to the optimal allocation as possible.

Preemptive resume hit rate (H): Sometimes the strategy in question may mis-predict the resume time; i.e., a user connection might appear prior to the anticipated resume time (when the database is queisced). The preemptive resume hit rate captures how often a strategy can successfully preemptively resume a quiesced database before actual user activity is encountered. It is simply the ratio of the number of successful preemptive resumptions to the number of quiescing events.

A solution that has a 100% preemptive resume hit rate never mis-predicts (which would force an on-demand resume.) This characteristic is hard to attain while maintaining our efficiency metric (**E**) above; again, in practice, we aim to maximize the resume hit rate.

Figure 3 shows a sample of the discretized binary activity pattern for a database with alternating active and idle periods. The metrics for the given period of time using a 3-hour idleness based quiescence policy with on demand resume can be computed as follows:

$$\begin{aligned} T_i &= \quad 2hr + 7hr + 6.5hr = 15.5hr \\ T_q &= \quad 4hr + 3.5hr = 7.5hr \\ E &= \quad 7.5/15.5 = 0.48 \end{aligned}$$

The resume hit rate would be 0 since we use on-demand resume.

These two metrics quantify the tradeoffs involved in dynamic quiescence and preemptive resumption and hence are both necessary to evaluate and compare different methods. For instance, observe that preemptively resuming a database in a conservative manner will clearly increase the resume hit

rate (H). If allocation efficiency is not considered, a strategy that always resumes *immediately* after quiescence would win. Similarly, if we do not attempt any preemptive resumptions at all, the allocation efficiency would be very high since we remain quiescent until user activity occurs. The resume hit rate prevents such a strategy from winning. Therefore, any method that we consider, must attempt to maximize both metrics.

IV. METHODS

With the metrics as described above, we now describe three methods that address the optimization problem. At the outset, we re-emphasize that operational efficiency is a key factor in identifying these methods. In other words, our goal is to be able to deploy these methods on millions of databases and make decisions based on the identified patterns. The methods that we present here are constant time update and look-up methods, with minimal memory requirement.

To begin, we first describe techniques for preemptive resumption, while fixing a static quiescence policy as described in [8]. Then, we propose a dynamic approach that preemptively resumes databases *and* also aggressively quiesce databases based on past database activity patterns. The idea is that static approaches leave significant low-hanging fruit on the table by always waiting a fixed amount of time to make a decision.

As a strawman baseline approach, we have also considered a *randomized decision maker* that picks times to resume databases based on a coin flip (our version still uses a static quiesce policy). We define a parameter **wait_time** as the time interval at which we make decisions on when to resume. For every database in the quiescent state, the algorithm chooses a (weighted) random number at every *wait_time* interval to decide if it should be resumed. Note that this approach is used only as a baseline to evaluate other approaches.

A. Idle Time Averaging

This simple approach is based on the hypothesis that for a given database, the duration of its prior idle periods is a strong predictor for future idle periods. Based on this hypothesis, we consider the mean duration of prior idle periods in the history of each database. However, the duration of past idle periods can vary significantly, and so we apply a conservative correction by subtracting the standard deviation of the prior idle periods from the mean.

We continue to use an idleness-based policy for quiescing databases and the approach proposed above is used only for preemptive resumption. The predicted resume time for a given database is given by $\mu - \sigma$, where μ is the mean of prior idle period durations and σ is their standard deviation. The number of prior idle periods that is used is decided by a parameter N_p.

Based on our evaluation of this approach, we observed that while the results were better than the random strawman approach, there was still considerable room for improvement. We refer the reader to Section V for more details regarding our evaluation. A deeper analysis of the data revealed two main observations. The first observation is that some databases do not have a repeating pattern of idle periods (these databases may be used for ad hoc applications). The second, and more interesting, observation is the existence of bimodal and trimodal (and

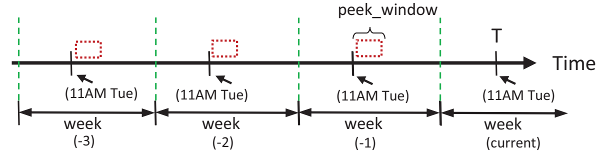

Fig. 4. Example illustrating N-Week Lookback

beyond) patterns in some databases. Below are two actual Azure SQL Database examples of idle time patterns in minutes (colors added to help identify the bimodal and trimodal patterns):

- 2170, 127, 125, 126, 125, 3005, 124, 127, 126, 123, 3008, 128, 123, 128, 126, 3003, 124, 124, 125, 124, 3004, 127,
- 3121,420, 295, 419, 299, 420, 300, 415, 298, 420, 3180, 416, 298, 420, 296, 418, 301, 432, 296, 400, 3180, 422, 299, 413, 300, 420, 296, 419, 299, 416, 3178,

With these types of patterns, the naive averaging-based approach will never be able to make accurate predictions.

B. N-Week Lookback (Business Rhythm)

The above observation led us to a different approach that does not attempt to aggregate consecutive idle periods, but instead relied on the regular work-week business rhythm in order to make predictive decisions. For example, we hypothesize that if a database was active from 11am-12pm local time on Tuesday last week and the week before, then it is likely that it will be active again at around the same time this week. Figure 4 illustrates how the decision to resume is made at 11am on Tuesday. In that example, we show a key parameter called **peek_window**, which is defined as the length of a constructed time window that we examine for activity when we look back at activity profiles across past weeks.

This strategy works as follows. Just as before, we use an idleness-based policy for quiescing databases. At a regular interval, for every database in quiescent state, we do the following: Let the current time of day be T, as illustrated in Figure 4. We look back to the last N_w weeks of data for this database (N_w = 3 in Figure 4). We examine a span of time (parameterized as the *peek_window*) starting from T for activity in each of the N_w weeks. If m out of the N_w prior weeks had activity during the *peek_window*, then we compute m/N_w as a score. (In our work, we also considered a number of linear and non-linear decay schemes to calculate the score. In this paper, we only present the uniform scoring model.)

Then, with a threshold parameter (which we call *resume_threshold*, denoted as H_r), we examine if the score is greater than the threshold parameter value, and if so, we preemptively resume the database at T. For the rest of our discussions and evaluations, we use $N_w = 5$.

C. N-Week Lookback with Dynamic Quiescence

In the two methods described above, we have used telemetry data to resume databases preemptively instead of doing it on-demand. We now go a step further to identify patterns that allow us to aggressively quiesce databases as well.

The intuition behind this method is similar to the N-Week Lookback approach, but applied to quiescence as well. In other words, we rely on the business rhythm property of the data.

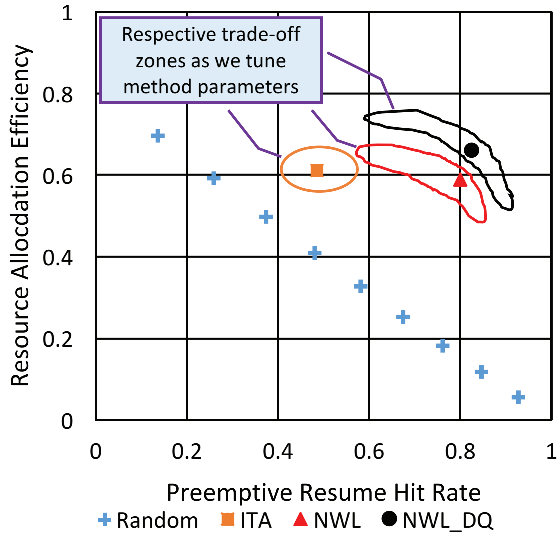

Fig. 5. Four methods evaluated over the telemetry trace. ITA - Idle Time Averaging; NWL - N-Week Lookback; NWL_DQ - N-Week Lookback with Dynamic Quiescence

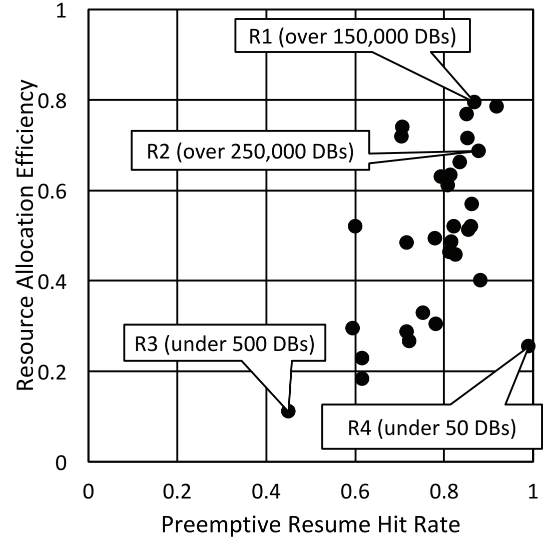

Fig. 6. The N-Week Lookback with Dynamic Quiescence results broken down per region (e.g., R1, R2, etc.) of Azure SQL Database. Regions that are lightly populated may be new, and do not have regular workloads yet.

For example, we hypothesize that if a database was idle from 9pm-10pm local time on Thursday last week and the week before, then it is likely that it will be idle again at around the same time this week. In essence, the quiescence policy that we use here is a dual of the resumption policy with some changes as we describe below.

This strategy works as follows. Whenever a database exhibits idleness for a relatively short duration (say 30 minutes or one hour), our quiescence mechanism is kicked off. We define a parameter **initial_wait** (denoted by ϵ) as the short duration of idleness signalling that quiescence can be considered. Let the current time of the day be T, and suppose the idle duration of ϵ has been observed for a database. We then take the following steps for such a database:

- Examine the last N_w weeks of data for this database.
- Examine a span of time (the *peek_window*) starting from T in each of the N_w weeks.
- If a significant portion of this time has been idle in the N_w prior weeks, compute the fraction of idle time as a score.

Then, with a threshold parameter for quiescence (denoted as H_q), we decide to quiesce the database at time T if the score is greater than the threshold. Once quiesced, we fall back to the N-Week Lookback resume method described in Section IV-B. As we can see, this approach employs a more aggressive quiescence policy while retaining the benefits of preemptive resume as before.

V. Evaluation and Discussion

Our evaluations are on data from a 160-day telemetry trace (late spring to early fall 2016) covering all worldwide clusters that run the Microsoft Azure SQL Database service. There are over 1.5M databases included in this telemetry trace, and the raw telemetry data size is many hundreds of terabytes.

A. Evaluation space and Parameters

In this short paper, we present four computationally scalable methods that progressively get higher allocation efficiencies and resume hit rates. Of course, other machine learning algorithms exist that balance simplicity with effectiveness, but they incur a higher computational cost (exploring this issue further is part of future work). While we have examined a number of different combination of parameter settings and algorithms, in the interest of space, we focus on the following four key methods.

[Random]: This baseline method uses a 3 hour idleness-based policy, a 2 hour *wait_time*, and a binary random variable weighted between 0.1–0.9.

[ITA]: This method uses the approach of Idle Time Averaging with static quiescence, using a 3 hour idleness-based policy and $N_p = 5$.

[NWL]: This method is the N-Week Lookback (static quiescence) approach, with a 3 hour idleness-based policy, a *peek_window* of 60 minutes, a threshold of $H_r = 0.4$, and $N_w = 5$.

[NWL_DQ]: This method uses the N-Week Lookback with Dynamic Quiescence approach. The parameters settings are: $\epsilon = 30min$, the *peek_window* parameter is set to 60 minutes, $H_r = 0.4$, $H_q = 0.2$, and $N_w = 5$.

B. Results and Discussion

In Figure 5, shows the results using the four methods over the entire 160-day period. The x-axis shows the preemptive resume hit rate metric and the y-axis shows the resource allocation efficiency. Remember from Figure 1 that our goal is to maximize both metrics, and hence we aim to move closer to the top-right corner of the space.

The randomized predictor (*Random*) results in an almost linear pattern as we vary the random binary variable's weighting. As expected, this linear pattern forms a baseline, and we are only interested in methods that can do better than this method. For the other three methods, we show trade-off regions in Figure 5 that indicate the space achievable by varying the parameters of that individual method. (To make the figure easier to understand, we omit the other data points.)

The Idle Time Averaging method (*ITA*) is able to perform better than the Random method on both metrics, as it can capture cases where the idle duration is a strong predictor of future idleness. The N-Week Lookback with a static quiescence policy (*NWL*) is able to preemptively resume databases in many more cases, which indicates that the weekly business rhythm is indeed a stronger predictor of future activity patterns. Therefore, compared to *ITA*, *NWL* is able to achieve up to 80% hit rate while not compromising on the allocation efficiency metric. The N-Week Lookback with Dynamic Quiescence (*NWL_DQ*) method improves upon *NWL* on both metrics as shown by the black region in Figure 5. An aggressive dynamic quiescence policy coupled with a robust preemptive resume strategy results in overall better performance on both metrics.

Next, we dig deeper into the *NWL_DQ* method, and present the results for each region of the Azure SQL Database service. Figure 6 shows this drilled down result, where each point represents a region. We observe that *NWL_DQ* is able to perform very well in certain regions (see region R1), and there are regions where it is unable to achieve acceptable efficiencies and hit rates (i.e. below the baseline Random performance). By analyzing these results further, we discovered that regions that are either new, or sparsely populated are often the ones where we are unable to predict activity with sufficient accuracy. Densely populated regions generally have many more databases with regular predictable workloads than the sparsely populated ones. Comparing the performance of the *NWL_DQ* method across the global (Figure 5) and region-level (Figure 6) data sets shows that it is able to achieve good values for both of our metrics globally, despite the new and sparse regions.

We are actively considering a number of computationally efficient statistical and machine-learning approaches that go beyond or complement our existing methods. As we have noted however, operationally, our main goals are two-fold: maximize resume hit rate (quality of service is the priority) and develop the least intrusive methods to run in production. We have shown that our methods, which have constant time update and look-up properties, and require only a modest amount of memory to run, can provide almost 90% hit rates with 80% allocation efficiency in certain regions.

VI. Related Work

There has been significant interest in studying cloud database service efficiency through multi-tenancy and over-booking [3], [4], [6], [9]–[11], [13]–[15]. While the prior works tackle important problems related to scheduling, resource management, and data placement, none of them have validated cloud user behavioral models with actual traces of production telemetry. Prior work has also developed machine learning [5], [16] or statistical [12] models to predict cluster and user behavior changes, but they do not discuss our over-booking context and the need for computationally scalable methods.

VII. Conclusions

In this paper, we have considered a practical challenge that is faced by cloud service providers, which is to balance the conflicting objectives of resource allocation efficiency and maintaining a high quality of service. We formulate this problem, and define metrics that enable us to quantify these objectives and evaluate solutions. We also propose efficient, computationally scalable methods that are able to predict user activity by leveraging past database activity patterns. Our evaluation is done over a large trace of real production telemetry data from Microsoft Azure SQL Database, and to the best of our knowledge represents the first study that uses data at this scale. Our present work is only an initial step towards a more intelligent cloud database service, and there is a huge potential for future work in this area that uses even more sophisticated techniques to move us closer to the optimal operating point (point D in Figure 1).

References

[1] Microsoft corporation. http://azure.microsoft.com, 2016.

[2] P. A. Bernstein, I. Cseri, N. Dani, N. Ellis, A. Kalhan, G. Kakivaya, D. B. Lomet, R. Manner, L. Novik, and T. Talius. Adapting Microsoft SQL Server for Cloud Computing. In *ICDE*, pages 1255–1263, 2011.

[3] C. Curino, E. P. Jones, S. Madden, and H. Balakrishnan. Workload-aware Database Monitoring and Consolidation. In *SIGMOD*, 2011.

[4] C. Curino, Y. Zhang, E. P. C. Jones, and S. Madden. Schism: a Workload-Driven Approach to Database Replication and Partitioning. *PVLDB*, pages 48–57, 2010.

[5] J. Duggan, U. Cetintemel, O. Papaemmanouil, and E. Upfal. Performance Prediction for Concurrent Database Workloads. In *SIGMOD*, 2011.

[6] A. J. Elmore, S. Das, A. Pucher, D. Agrawal, A. El Abbadi, and X. Yan. Characterizing Tenant Behavior for Placement and Crisis Mitigation in Multitenant DBMSs. In *SIGMOD*, pages 517–528, 2013.

[7] W. Lang, F. Bertsch, D. J. DeWitt, and N. Ellis. Microsoft Azure SQL Database Telemetry. SoCC, pages 189–194, 2015.

[8] 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. *PVLDB*, 2016.

[9] W. Lang, S. Shankar, J. Patel, and A. Kalhan. Towards Multi-Tenant Performance SLOs. In *ICDE*, pages 702–713, 2012.

[10] Z. Liu, H. Hacigümüş, H. J. Moon, Y. Chi, and W.-P. Hsiung. PMAX: Tenant Placement in Multitenant Databases for Profit Maximization. EDBT, pages 442–453, 2013.

[11] H. J. Moon, H. Hacigümüş, Y. Chi, and W.-P. Hsiung. SWAT: A Lightweight Load Balancing Method for Multitenant Databases. In *EDBT*, pages 65–76, 2013.

[12] B. Mozafari, C. Curino, A. Jindal, and S. Madden. Performance and Resource Modeling in Highly-concurrent OLTP Workloads. In *SIGMOD*, pages 301–312, 2013.

[13] B. Mozafari, C. Curino, and S. Madden. DBSeer: Resource and Performance Prediction for Building a Next Generation Database Cloud. In *CIDR*, 2013.

[14] J. Schaffner, T. Januschowski, M. Kercher, T. Kraska, H. Plattner, M. J. Franklin, and D. Jacobs. RTP: Robust Tenant Placement for Elastic In-memory Database Clusters. SIGMOD, pages 773–784, 2013.

[15] R. Taft, W. Lang, J. Duggan, A. J. Elmore, M. Stonebraker, and D. J. DeWitt. STeP: Scalable Tenant Placement for Managing Database-as-a-Service Deployments. SoCC, 2016.

[16] P. Xiong, Y. Chi, S. Zhu, J. Tatemura, C. Pu, and H. Hacigümüş. ActiveSLA: A Profit-oriented Admission Control Framework for Database-as-a-service Providers. SoCC, 2011.