Not for the Timid: On the Impact of Aggressive Over-booking in the Cloud

Willis Lang*, Karthik Ramachandra*, David J. DeWitt*,
Shize Xu, Qun Guo, Ajay Kalhan, Peter Carlin
Microsoft Gray Systems Lab*, Microsoft
{wilang, karam, dewitt, shizexu, qunguo, ajayk, peterca}@microsoft.com

ABSTRACT

To lower hosting costs and service prices, database-as-a-service (DBaaS) providers strive to maximize cluster utilization without negatively affecting their users' service experience. Some of the most effective approaches for increasing service efficiency result in the over-booking of the cluster with user databases. For instance, one approach is to reclaim cluster capacity from a database when it is idle, temporarily re-using the capacity for some other purpose, and over-booking the cluster's resources. Such approaches are largely driven by policies that determine when it is prudent to temporarily reclaim capacity from an idle database. In this paper, we examine policies that inherently tune the system's idle sensitivity. Increased sensitivity to idleness leads to aggressive over-booking while the converse leads to conservative reclamation and lower utilization levels. Aggressive over-booking also incurs a "reserve" capacity cost (for when we suddenly "owe" capacity to previously idle databases.) We answer these key questions in this paper: (1) how to find a "good" resource reclamation policy for a given DBaaS cluster of users; and (2) how to forecast the needed near-term reserve capacity. To help us answer these questions, we used production user activity traces from Azure SQL DB and built models of an over-booking mechanism. We show that choosing the right policy can substantially boost the efficiency of the service, facilitating lower service prices via lower amortized infrastructure costs.

1. INTRODUCTION

One of the main challenges of a database-as-a-service (DBaaS) provider such as Microsoft is to control costs (and lower prices) while providing excellent service. With the DBaaS adoption rate skyrocketing along with the increasing focus on big-data analytics, providers are seeing consistent year-over-year growth in subscribers and revenue. Microsoft's own infrastructure footprint to support this growth includes data centers spanning 22 regions with five more regions on the way (at time of preparing this paper [2],) where each data center investment costs hundreds of millions of dollars. At billion dollar scale, in a low-margin business, achieving additional percentage points of efficiency results in $10Ms-$100Ms of yearly savings.

Proceedings of the VLDB Endowment, Vol. 9, No. 13

The efficiency challenge is to maintain high user density (and utilization levels) on these clusters without noticeable disruption to the users' workloads. To solve, or combat, this so-called "multi-tenancy problem", many different approaches have been presented and studied in the recent literature (see Section 2.1.) These approaches all focus on optimizing for compatible tenant co-location or co-scheduling – essentially to try to *over-book the cluster with databases*. It is not the goal of this paper to re-examine these issues, but to focus on an ignored, real-world problem that providers must deal with when striving for high service efficiency – managing the capacity-related *side-effects* of employing these over-booking policies. Specifically, what happens when, after aggressive over-booking, due to a change in cluster-wide usage, we suddenly find that we owe more capacity than we actually have?

To start, we shall consider a straight-forward multi-tenant mechanism that can be used – reclaiming capacity when a user is idle for a prolonged time. Certain DBaaS architectures, such as the current architecture used in Azure SQL DB (ASD), trade-off higher performance isolation and manageability at the expense of lower user density and efficiency by focusing on modularity. In the ASD architecture, we can consider a SQL Server instance process as acting as a quasi-virtual machine, that executes queries for its attached databases. For certain tiers of ASD subscribers, namely Basic and Standard, these attached databases are backed by files stored via a "shared disk" architecture. While these databases are attached, the instance process is up and consuming cluster capacity. However, if we "detach" a database from the instance[1], (when the database is idle,) then this SQL Server instance may be shut down and its resources can be *reclaimed.* When a query is issued, an instance is brought back online in the cluster wherever the necessary capacity is available and the database is re-attached.[2] This suggests an immediate opportunity: if databases are idle for considerable periods of time, then we can detach them from the instance to reclaim capacity for other databases – effectively *quiesce* them. This act can immensely boost cluster utilization.

Obviously, there are certain problems that arise from employing this mechanism and over-booking a cluster. To illustrate, we can consider how a parking lot that sells reserved spots to customers, might operate. On any given day, at any given time, some of the customers may not be present to occupy their parking space. When this happens, given the under-utilization of the lot, additional customers may be accommodated in the reserved spots. At a holistic view, as long as there is a *net-positive* (or net-zero) increase of free parking spots, this is sustainable. Unfortunately, sometimes, more

[1]The detach mechanism is only used here as a simplified example, and not a real or complete mechanism in use.

[2]Building an efficient mechanism is an important aspect of this problem, but is neither a sufficient solution, nor is it within the scope of this paper.

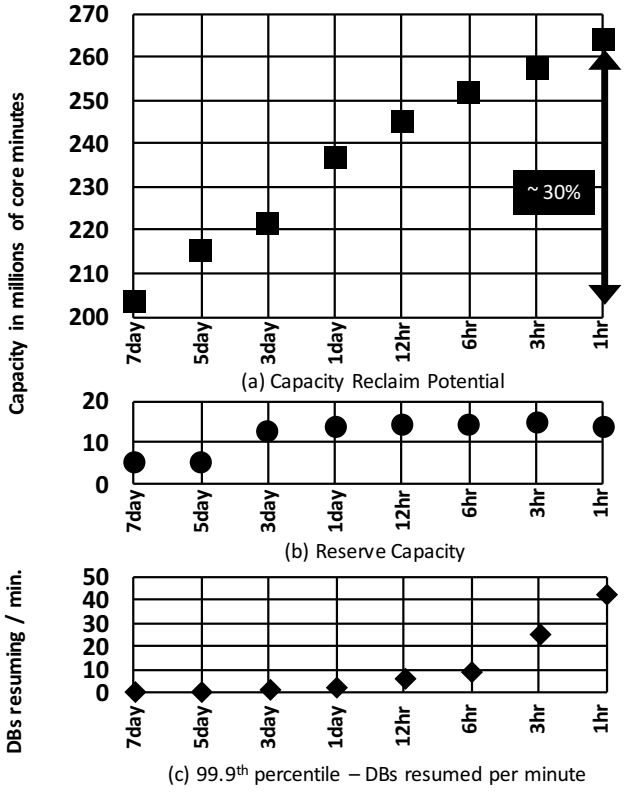


Figure 1: (a) Capacity (in core minutes) reclaimed and (b) reserved, along with (c) the observed DBs resumed per minute based on different quiescence policy lengths over a three month period from a production ASD cluster.

parking spots are being claimed than freed, a *net-negative* trend. As a result, the parking lot may need to dedicate some reserved capacity to handle net-negative capacity changes. We address this oft-ignored problem that arises in the context of DBaaS when we employ quiescence techniques to over-book the cluster – *a set reserve capacity must be kept on hand for when quiesced DBs require resources forcing them to be resumed.*

In a cloud service like Azure SQL DB, there are generally periods of ebb and flow in user activity (think weekends and holidays), where there are sustained periods of net-positive and net-negative quiescence. The problem that we (and the parking lots) have is that we don't know how long we should wait before we quiesce an idle DB and provide the capacity to some other DB. One straightforward approach is to hypothesize that if the DB has been idle for a static amount of time, then it is likely to continue in this way and hence it is safe to quiesce. We can think of this duration of idle time as being defined by the idleness based *quiescence policy*.

As the astute reader may already surmise, the quiescence policy has a direct implication on the amount of reserve capacity that must be kept on hand, as well as the amount of capacity that we can reclaim. Note however, that while the efficiency of the quiesce mechanism *may* help mitigate this issue to some extent, the act of over-booking itself creates the possibility of zero available capacity that cannot be solved fully by any degree of mechanism improvement. Our first goal is therefore, given a production cluster (and users), to find its "ideal" quiescence policy.

In Figure 1 we provide an example result from our analysis. These results are from a single cluster located in the US over a three month period ending 01/31/16 (results for additional clusters are presented in Section 6). Here we used the production telemetry data to determine user activity and idle periods as an input into our quiescence model. For these results, we have employed eight different quiescence policy lengths (the idle time required before we initiate the mechanism) from 1 hour to 7 days. The methods by which we model the required reserve capacity are described in Sections 4 and 5, but in essence, for each policy, we calculate the amount of net churn of DBs quiesced and resumed over the *entire three month period.* As we model quiescence, we can determine how much potential capacity we will reclaim with this mechanism.

Figures 1(a) and (b) contrast the potentially reclaimable capacity (in millions of CPU core minutes – a simplified metric for this paper), versus the reserve capacity required, respectively. As we shorten the policy length from 7 days to 1 hour, we see in Figure 1(a) that the amount of capacity reclaimed increases – by almost 30% – from about 200 to 260 million CPU core minutes. Correspondingly, we also see an increase in the required reserve capacity in Figure 1(b). Figure 1(c) shows the 99^{th} percentile of all DBs resumed per minute of the three month period given the real user activity. We see that the number of DBs resumed per minute climbs dramatically as we shorten the policy length. This would increase the operational complexity and may sometimes decrease user satisfaction.

To determine the "ideal" policy, we must normalize DBs resumed per minute so that we can come up with a single measure per policy. We did this by conservatively assuming that, if a DB is resumed more than five times in a month, (assuming a 99.99% Azure SQL DB SLA [2],) then a capacity compensation (of one month) will be provided. Consequently, we find the measure: $net\_reclaim\_potential = (reclaim\_potential - reserve\_capacity - compensated\_capacity)$. In relation to Figure 1, we found that for this US cluster, the 1 day policy was the best out of these 8 options and provided an 18% improvement over the worst performing policy, which was the 1 hour policy due to high compensation. At scale, in the context of billions of dollars of global infrastructure costs, *choosing the right policy* can swing service operating costs by tens or even hundreds of millions of dollars a year, and these are savings which can be directly passed to customers.

As we will show, different clusters exhibit different activity patterns due in part to the population makeup. Therefore, none of this would do us any good if we aren't able to continuously forecast, monitor, and adjust our reserve capacities in a production setting. In this paper, we will also discuss our second goal to develop a predictive model for the amount of reserve capacity and show the results of evaluating our model. Our evaluations of historical production data provides penalty measures for various quiescence policies and forecasting models which show us how well we would have done. We employ penalty measures such as a capacity outage when databases are resumed but no reserve capacity is available (over-aggressive forecasting) and unused reserve capacity (over-conservative forecasting). Our three month evaluation can be found in Section 6.

To summarize, we make the following contributions in this paper.

- We introduce and formulate the oft-ignored problem due to over-booking DBaaS clusters: setting aside the right amount of reserve capacity and directly costing potential user impact.
- Using a quiescence mechanism, we define a tuning variable, a policy analysis model, and various metrics that provide insights towards answering the above problems.
- We present our analysis-backed solutions to two real-world problems: how to find the ideal quiescence policy for our mechanism and how to periodically forecast the reserve capacity.
- We evaluate our solutions over extensive three month datasets from two production clusters and present our results and findings.

2. PRELIMINARIES

To begin, we discuss related work on the topic of database-as-a-service followed by some background on the Azure SQL DB service as well as the telemetry data that we used.

2.1 Related Work

Along with the sudden surge in adoption in data processing in the cloud, we have also seen a swing in research towards the relevant topics. The different approaches to cloud data processing can be broadly divided by the different architectures [10]: *shared hardware* [4], *shared table* [1, 24], and finally the middle ground model *shared process* [3, 22]. These different architectures trade isolation and manageability against properties such as density and efficiency. Azure SQL DB employs a different model – a process-centric model that focuses on performance and security isolation [3, 7, 21, 22].

On the topic of efficient multi-tenant scheduling and placement, there have been many methods and schemes discussed in the literature [5, 6, 12, 13, 15, 16, 18, 20, 23]. While some of the placement schemes do not consider "over-booking" (as a consequence of a specific cloud architecture), most of the tenant placement research strives to take advantage of underutilization or temporal contra-patterns, which inevitably leads to the problem that *we have sold more capacity than we actually have.* Similarly, most of the scheduling research leans toward providing resources when they're needed but taking them away when they're not. However, as we mentioned in Section 1, none of them consider the actual problem associated with over-booking, that is, what to do if we have used up all of our capacity and a user that was idle now demands resources?

The other aspect that has garnered significant attention is that of efficient database migration mechanisms to allow placement reconfigurations and failovers [8, 11, 17]. Mechanisms such as these are extremely important as to reduce latency of any capacity rearragement. Ultimately effective full solutions must incorporate well-designed mechanisms as well as the analysis framework that we focus on to determine the right policies to employ.

Other efforts more similar to ours deal with developing models on the workloads to aid in the system's decision-making. These include developing machine learning models [9,25] or statistical models [19] to predict changes that the system must adapt to. However, none of these prior studies focus on our over-booked capacity problem, nor do they build and evaluate their models on the quality and quantity of production Azure SQL DB data that we have here.

2.2 Azure SQL DB Service

We now provide the basics of Microsoft's Azure SQL DB service necessary for this paper; additional information can be found at [2]. Azure SQL DB operates on a process-oriented model where multiple customer databases may be co-located and served together via a single (Azure) SQL Server process. Given that Azure SQL DB is running on SQL Server, many of the basic database management concepts of on-prem SQL Server remain available to the service.

Most notably, we could consider the SQL Server detach/attach mechanism (*as an example mechanism for our purposes*) that allows a database to be "disconnected" from the SQL Server instance process. After the database is detached, its physical files can be moved or reattached by any other SQL Server instance and the database will then consume resources provided by the new instance.

Note that this mechanism is *not* free. For example, detaching a database can take minutes or longer depending on the volume of dirty memory pages in the buffer pool and I/O latency. Other factors can influence the latency of a reattach as well and we account for these (see Section 3.1.)

DB	a unique identifier for the customer's database
slo	the subscription t-shirt size
timestamp	the timespan from $(timestamp - 15s)$ to $timestamp$
node	the node that the database resides on
avg_cpu	the average cpu utilization (%)
avg_io	the average I/O utilization (%)
avg_memory	the average memory utilization (%)

Table 1: Telemetry Schema

Azure SQL Database currently offers a tiered subscription model that allows customers to choose from databases with different subscription levels, also known as '*t-shirt*' *sizes*. T-shirt sizes correspond not only to various levels of performance, but also availability, and reliability objectives. Azure SQL Database's current subscription model consists of three tiers: Basic, Standard, and Premium (Standard and Premium are further subdivided into four and three sub-level *t-shirt sizes*, respectively.)

The main difference that sets Premium databases apart is the fact that the physical data files are stored on the same node that hosts the SQL Server instance and not as files stored in Azure Storage volumes. This distinction provides immense benefits in performance, but the Azure DB service must now manage physical data file replication on other Azure SQL DB nodes for availability. On the other hand, Basic and Standard tier databases are backed by physical files stored on the Azure Storage layer, which performs replication, thereby providing availability. For these two tiers, we can attach and detach databases at will, changing the location of the SQL engine within the cluster. Databases subscribing to these two lower-cost tiers also make up the vast majority of all databases in the service.

Finally, all of the t-shirt sizes come with performance SLOs that essentially define the capacity requirements of a particular database. These are defined using a proprietary metric 'database transaction unit' (DTU), which is an internal transactional benchmark metric in the spirit of TPC-C [2]. Internally, these DTUs map to traditional CPU core, memory, and I/O bandwidth metrics. [3] *In this paper, we will frequently use CPU Cores as a normalizing capacity metric.*

2.3 Telemetry Data

In this work, we relied upon the performance telemetry data to tell us about an Azure SQL DB user's activity. Instead of telling us what queries were run, or how often they were issued, it tells us the amount of CPU, memory, and I/O that was consumed by this database (in the process of executing queries.) The rough schema of the data is described by the column descriptions in Table 1.

This telemetry was collected from two Azure SQL DB clusters, one from the US and the other from Europe. The data is very similar to the telemetry described in [14], but it is of the *current* Azure architecture instead of the earlier version (v1) of Azure DB released in that paper. The main difference is that we now have data of finer granularity (records are emitted every 15 seconds instead of every 5 minutes) and generally, fewer anomalies are present. Every 15 seconds, every database could emit a row containing the resource utilization of that database during that timespan. Similar to the prior data release, to reduce data volume, records are not emitted if the database was completely idle (all three resource metrics were 0). This implies, for instance, that if we see tuple $(db_3, basic, 01/01/16\ 12{:}00{:}15, n38, 0.85, 0.20, 0.14)$ followed by tuple $(db_3, basic, 01/01/16\ 12{:}05{:}15, n38, 0.32, 0.19, 0.12)$, then db_3 was idle for five minutes.

[3] We will not disclose the exact mappings.

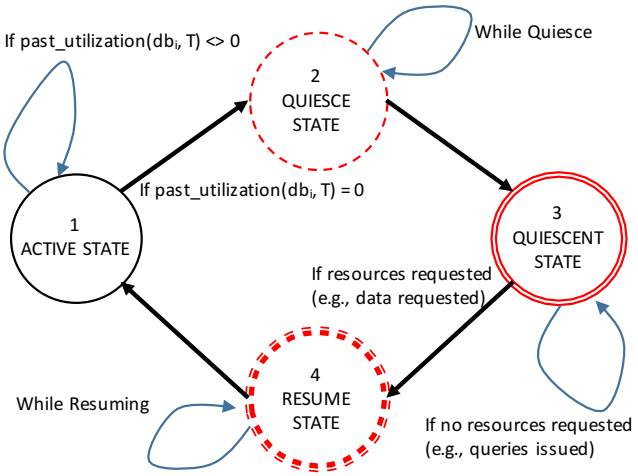


Figure 2: Quiescence State Diagram for some database db_i and policy length p. $past\_utilization$ is a function that returns the utilization level of the db_i in the past T time.

3. PROBLEM FORMULATION

We now formulate two capacity management problems involved in the over-booking of database clusters. First, we will discuss a mechanism that can be employed to over-book clusters, and then we will describe the problems in the context of Azure SQL DB service.

3.1 Mechanism Background

The quiesce mechanism that is relatively straight-forward to implement (and think about) in SQL Server is the act of detaching a database.[4] When the database is detached, it can no longer service queries as the database engine no longer has access to the data. When a database is detached, certain memory and log operations may need to be performed and/or completed. This includes memory resident data checkpointing which can take a non-trivial amount of time to complete (minutes or longer).

Conversely, if the user requests a data page from a quiesced DB, then it must be "*resumed*", which would include invoking SQL Server's database attach function. Similarly, this is not an instantaneous action as the database's physical files stored in the "shared disk" layer must be found, sufficient free capacity for the DB's subscription tier must be found, and the attach itself performed. Therefore, both of the transitions – quiesce and resume – must be accounted for.

Figure 2 illustrates the intuitive four-state state diagram for our quiesce/resume workflow.

DEFINITION 3.1. *A database can be modeled as being in one of four states: (1) Active; (2) Quiesce; (3) Quiescent; and (4) Resume.*

In our model, databases remain in state 1 if there is any level of utilization in the past T length timespan. If sufficient idleness is detected, the db is transitioned to state 2 where it stays until the quiesce process is completed and it moves onto state 3. While there are no requests of this database, it remains in the quiescent state. Once a database receives a request, it transitions to state 4 and stays there until resume completes, at which point it returns to state 1.

[4] Note that this detach/attach mechanism is only used here as a simplified example, and is not a real or complete mechanism in use.

3.2 Formulation

Now that we have described our mechanism, we can focus on the problems associated with over-booking. Using the above mechanism, we may attempt to increase service efficiency by reclaiming cluster resources from databases that are idle (e.g., by detaching such databases), and using the reclaimed capacity to host more databases. A question that immediately arises here is: How do we decide which databases should be detached? In other words, we need a reasonable approach to identify the right quiesce candidates, hereafter referred to as the *quiescence policy*.

While there could be several quiescence policies possible, in this paper we restrict ourselves to a set of policies based on the duration of idleness exhibited by databases. The idleness-based quiescence policy is defined as follows: *A database is deemed to be a candidate for quiesce if it exhibits continuous idleness for a specified duration T, which we call the* ***quiescence policy length***. The idleness-based quiescence policy P is parameterized by the policy length T, and is based on the hypothesis that if a database has remained idle for time T, it is likely to remain idle for a longer duration and hence, is a suitable quiesce candidate.

Enforcing a quiescence policy $P(T)$ involves quiescing the databases that are identified by the policy, thus freeing up the corresponding cluster resources. The amount of resources that would free up as a result of applying policy $P(T)$ is referred to as the ***reclaim potential*** of $P(T)$. However, enforcing a quiescence policy also implies that a fraction of the quiescent databases may have to be resumed. There are continually databases being quiesced and resumed, so we care about the *net churn* in these databases. In the case of negative net churn (where we are resuming more databases than we quiesce), we need to accommodate these resumed DBs by reserving certain cluster resources. The reserve capacity required in order to enforce a policy is essentially the capacity necessary for the continuous swings in the net churn. The cost of maintaining this reserve is the ***reserve capacity cost*** of $P(T)$.

Another important factor that needs to be considered is the ***resume cost***. Resume incurs costs because they involve operations that include bringing back an instance online in the cluster wherever the necessary capacity is available, and re-attaching the database to it. Too many DBs being resumed can increase operational complexity and may even lead to dissatisfied customers, and hence is not desirable. Therefore, we account for the resume cost by taking a very conservative stance. We assume that if a database is resumed more than 5 times in a month, (4.5min. with a simplified 1 minute resume time,) it fails the 99.99% Azure SQL DB SLA [2], and has to be compensated in capacity. The capacity compensation incurred due to enforcing a policy is the *resume cost* of a policy $P(T)$.

Therefore, the ***total policy cost*** of a quiescence policy is the sum of its resume cost and the cost of the reserve capacity. Assuming that this cost is fulfilled from the *reclaim potential*, we can arrive at the ***net reclaim potential*** of $P(T)$ by subtracting the total policy cost from its *reclaim potential*.

$$\begin{aligned} net\_reclaim\_potential = & reclaim\_potential \\ & - reserve\_capacity\_cost \\ & - resume\_cost \end{aligned} \quad (1)$$

With the above notions, we are now ready to formulate the first problem we address in this paper.

PROBLEM 1. *Determine the quiescence policy $P(T)$ that leads to the maximum net reclaim potential.*

Observe that there is a trade-off here, between the *reclaim potential* and the *total policy cost*. Analyzing and understanding the

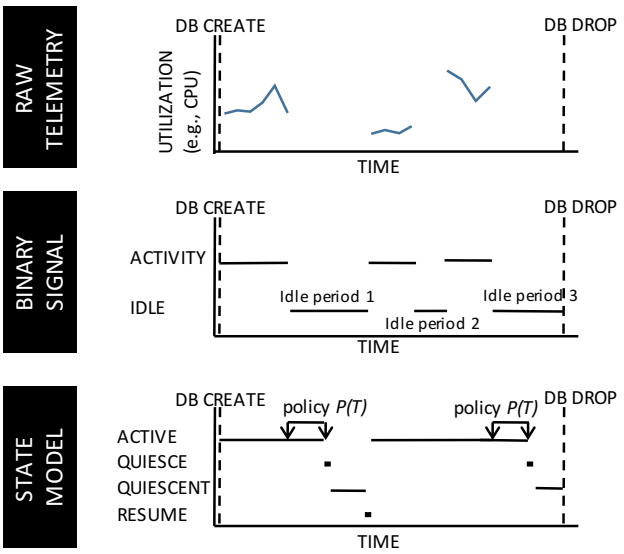


Figure 3: Example of processing the raw telemetry of a single database into our state model.

nature of this trade off is one of the important contributions we make in this paper.

The above problem aims to identify a suitable quiescence policy. Once a policy is chosen, a key aspect of enforcing the policy is to reserve a certain amount of cluster resources for databases that may need to be resumed. Note that this reserve capacity is not only a function of the chosen quiescence policy, but it also varies with time. Several factors such as weekends, holidays, special events etc. can impact the usage patterns of databases on the cloud, and implicitly impact both quiese and resume rates. Perhaps, more importantly, as the service takes on (and loses) customers *the aggregate behavior of the database population may change.* Therefore, we need to be able to continuously forecast the reserve capacity required in the future. This forecasting, based on observed patterns of resource utilization, needs to be robust enough to withstand variations in resource utilization. This leads us onto the other problem that we consider in this paper.

PROBLEM 2. *Given the resource utilization telemetry data and a quiescence policy $P(T)$, evolve robust prediction and evaluation schemes for forecasting the reserve capacity.*

Predicting reserve capacity in a conservative manner may be safe in terms of reducing the resume cost. However, the downside of conservative predictions is that it may result in reserve cluster capacity that may remain unused. This defeats the stated goal of our work, which is to increase the density of databases on the cluster. On the other hand, aggressive predictions may increase the number of databases by over-booking to a large extent, but may end up in a situation where there may be insufficient cluster capacity for databases that need to be resumed. Therefore robust prediction schemes that can balance these constraints, are critical to achieving our goal. Evaluating these predictions is a key aspect of ensuring that our predictions are production ready. The evaluations should allow us to reason about our predictions (and the resulting cluster behavior) in sufficient detail. Therefore prediction and evaluation go hand in hand in addressing the problem.

In Sections 4 & 5, we will describe how we use the Azure SQL DB telemetry described in Section 2.3, to answer the above problems.

4. QUIESCENCE MODELING

We now discuss how we start with the raw, analog telemetry data, discretize it to a basic binary signal, and then apply different quiescence models by varying the policy $P(T)$. In Figure 3, we present an illustrative example of how we process and analyze the data. First, as we described in Section 2.3, the raw data that serves as the input to all of this contains a row per database per timepoint *if the database was not idle (i.e., at least one performance metric's value was non-zero.)* This is why, in Figure 3, the raw telemetry data is *not* a continuous curve per database, but a series of curve fragments. Given this property of the raw data, it is straightforward to convert the "analog signal" into a binary one of "activity" or "idle". Finally, we must fit the binary data into the state model (Figure 2) by applying some chosen policy length T.

It is important to notice three key points from this transformation in Figure 3: (1) the first long idle period is significantly reduced when we transform it into "quiescent" because the policy $P(T)$ causes T amount of idle time to be spent in the "active" state. (2) we notice that the second idle period is completely eliminated because it is shorter than length T. Finally, (3) we must account for all quiesce and resume time in the model processing. (In this example, we also make a simplifying assumption that a database can be dropped from the quiescent state. Removing this assumption is straightforward.)

At this point, we need to introduce a bit of formalism to help describe the cluster-level capacity bookkeeping and also to set up our discussion in Section 5 and beyond. Recall, we have our state model given in Definition 3.1. Given these states, at any point in time that a database exists, it must be found in one of these four states:

DEFINITION 4.1. *While a database db_i exists in timespan $[j, k)$, $\mathbf{state(db_i, t, s)} \in \{1, 0\}$ indicates whether or not db_i is in state s at time t, where $j \leq t < k$; $k = \infty$ if db_i is not dropped.*

Finally, we can determine the minimum capacity requirement for any database in terms of a minimum CPU reservation in unit "cores" (a simplification for this paper):

DEFINITION 4.2. $\mathbf{capacity\_map(db_i)} \in \mathbb{R}^+$ *provides the mapping between a database db_i and its CPU core capacity reservation.*

For simplicity, we assume that a database never changes its core capacity reservation although, we can easily extend Definition 4.2 by adding a time component. With the above, for any state of interest, we can compute the total number of databases (and correspondingly, the capacity in CPU cores,) at any point in time. For instance, the equation below defines how we can compute the total capacity units (in cores) spent in the *resume* state 4 between minutes 580 and 600.

$$\text{for state 4: } \sum_{t=580}^{600} \sum_{\forall db_i} \mathbf{state}(db_i, t, 4) \times \mathbf{capacity\_map}(db_i) \tag{2}$$

In Figure 4, we show the result of applying Equation 2 over three months of data for all four states. The data is from a production cluster in the U.S. The quiescence policy we use in this modeling is $P(12hr)$. Each point in the figure represents a single minute. In this figure, with a 12hr quiescence policy, we can readily see daily and weekly patterns in both the Active and Quiescent state models.

If we focus in on the quiescent state results and model a variety of policies, we get Figure 5. In this figure, we zoom in on December for the same U.S. cluster as in Figure 4. We can see that as we increase the idle sensitivity, we begin to see much finer detail corresponding to weekend and daily patterns. Also, we can see that the level of

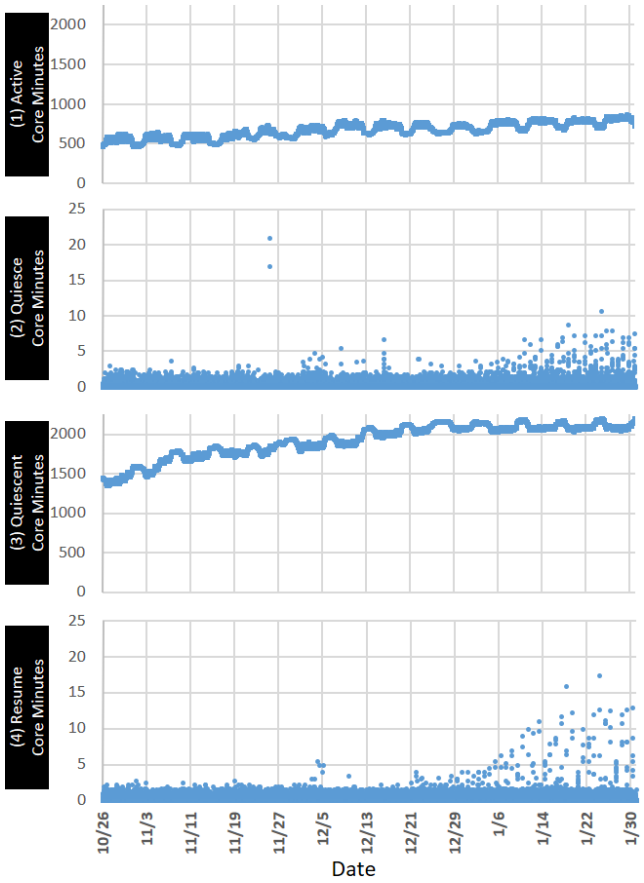


Figure 4: Application of Equation 2 for all four states over three months of production data with a policy $P(12hr)$.

quiescent cores increased (and was prolonged) during the Christmas holiday, eliminating the five day pattern for that work week. With the ability to model the impact of various quiescence policies, we can now attack the two problems.

5. ANALYSIS AND PREDICTION

With the method described in Section 4, we can model quiescence policies with different *quiescence policy lengths*. A careful and systematic analysis of these models is required in order to determine solutions to problems that we formulated in Section 3. In order to compare quiescence policies (Problem 1) and in order to forecast and evaluate the reserve capacity (Problem 2), certain metrics were defined in Section 3. We now make those definitions more concrete, and show how they are computed. Then, in Sections 5.2 and 5.3 we show how they can be used in answering the questions we are interested in.

5.1 Computing the Metrics

Consider the output of modeling a quiescence policy $P(12hr)$ on a cluster , shown in Figure 4. In particular, the quiescence patterns over a 3 week period in December is shown by the curve in Figure 6. The daily and weekly quiesce (and resume) patterns are easy to observe in Figure 6. A churn of quiesce and resume can be seen for each week day, and a large number of quiesces can be seen during weekends. For a given quiescence policy P with length T, we consider the following metrics.

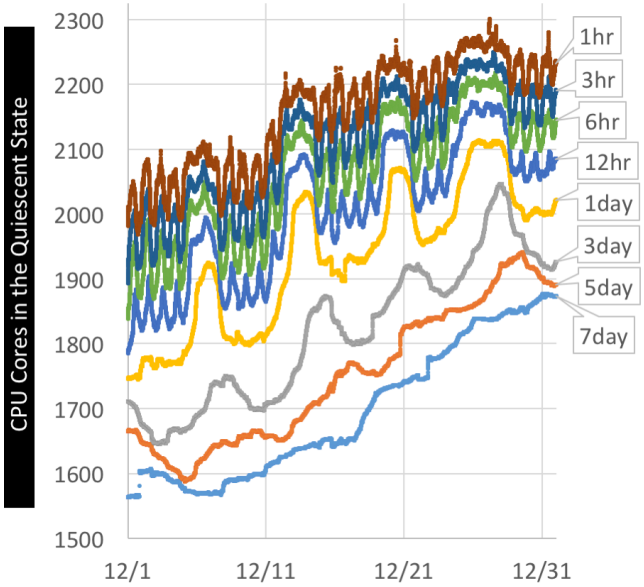


Figure 5: Quiescent State while varying the policy $P(T)$ for the month of December on the U.S. cluster.

5.1.1 Reclaim Potential

The *reclaim potential* is the amount of resources that would be freed up due to enforcing policy P for a specified duration. This is measured in terms of core minutes, denoted by *reclaim_potential(P)*. In Figure 6, the entire *area under the polygons* is a measure of the cores that are always in a quiescent state, which is, in other words, the capacity that is now reclaimed due to quiescence. Therefore, for N weeks, the reclaim potential of policy P is given by the equation below, where $free\_height_{min}(w)$ and $free\_height_{max}(w)$ are respectively the minimum and the maximum heights *under* the polygon for week w (shown for week 3 in Figure 6), and *minutes(w)* is the number of minutes in the week.

$$\sum_{w=1}^{N} 1/2(free\_height_{min}(w)+free\_height_{max}(w)) \times minutes(w) \quad (3)$$

5.1.2 Cost of Reserve Capacity

The *cost of reserve capacity* is the amount of reserve capacity required if policy P has to be enforced for a specified duration. This is essentially the net number of DBs resumed that was observed, and have to be accommodated. It is measured in terms of core minutes and denoted by *cost_of_reserve(P)*. In Figure 6, a net positive resume value is indicated by a downward trend in the number of quiescent cores. For a single day, the number of DBs resumed that have to be accommodated is equal to the net decrease in the number of quiescent cores observed within that day. Accumulating this over a one-week duration, we get the required *cost_of_reserve(P)*. The dashed polygons that bound the quiescence pattern in Figure 6 encapsulate the "churn" of quiesce and resume for every week. Therefore the *area of this polygon* is equal to the *cost_of_reserve(P)* for a given week. In general, for N weeks, the cost of reserve capacity is given by the following equation:

$$\sum_{w=1}^{N} 1/2(rsv\_height_{min}(w) + rsv\_height_{max}(w)) \times minutes(w) \quad (4)$$

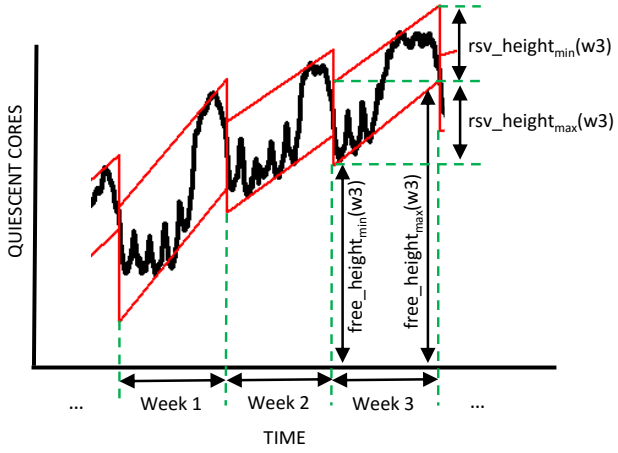


Figure 6: Quiescence patterns with bounding polygons over a 3 week period using the quiescence policy $P(12hr)$.

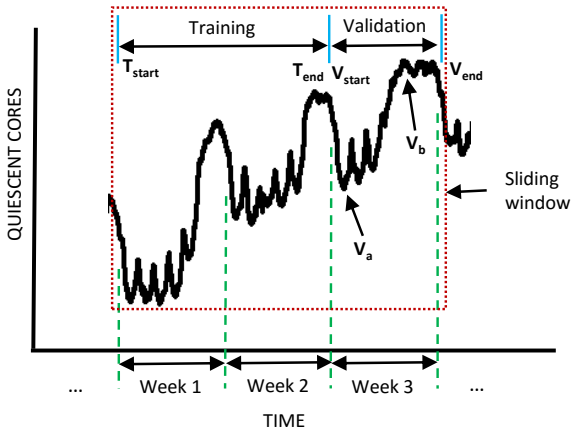


Figure 7: Forecasting reserve capacity: Training and validation phases.

where $rsv\_height_{min}(w)$ and $rsv\_height_{max}(w)$ are respectively the minimum and the maximum heights of the polygon for week w (shown for week 3 in Figure 6), and *minutes(w)* is the number of minutes in the week.

5.1.3 Resume Cost

As described in Section 3, it is not desirable to incur too many DB resumes. Therefore, we account for the resume cost by making a pessimistic assumption that if a database is resumed more than 5 times in a month, it has to be compensated in capacity (since it fails the 99.99% Azure SQL DB SLA [2]). The capacity compensation incurred due to enforcing a policy for a specified duration is the resume cost of a policy $P(T)$ and is denoted as *resume_cost(P)*.

The capacity compensation policy that we use is as follows. Consider a database belonging to subscription tier Y that is resumed more than 5 times in a month. Then, we compensate such a database with a 30 day reservation of tier Y subscription. This essentially translates to the CPU core capacity reservation given in Definition 4.2 for that database for 30 days.

5.1.4 Net Reclaim Potential

The amount of cluster resources that would become available due to enforcing policy P for a specified duration, after incorporating all the costs involved. The total cost of enforcing policy P includes the cost of the reserve capacity (*cost_of_reserve(P)*) and the cost due to resume compensation that violate the SLA *resume_cost(P)*. This is computed as described in Equation 1.

5.2 Comparing Quiescence Policies

As stated earlier, the goal of the first problem is to determine the quiescence policy $P(T)$ that leads to the maximum net reclaim potential. This can be done by comparing the set of policies under consideration, based on the metrics defined above. These comparisons can be performed as totals in order to get an overall idea of the reclaim potential over the entire duration under consideration. Another alternative that can help make this decision is to observe the behavior of the costs on a week-by-week basis, and study their variance over time. In Section 6, we show the results of these comparisons and describe these comparisons in more detail.

Clusters exhibit activity patterns that undergo changes over time due to changes in the population.Therefore the choice of the quiescence policy may have to be revisited periodically. In our analysis however, we observed that the ideal quiescence policy remained quite stable over three months. In a production setting, we could re-determine the ideal quiescence policy using the above metrics when necessary.

5.3 Forecasting Reserve Capacity

The focus of the second problem is to come up with robust schemes for prediction and evaluation of reserve cluster capacity. Intuitively, reserve capacity is the height of the "churn" of quiesce and resume observed. In these terms, forecasting reserve capacity essentially translates to predicting the height of this churn, for a specific duration in the future.

For evaluating our model, we use the standard cross validation approach, in order to estimate how accurately our model works in practice. We first train our model on the training data set, which yields a prediction. Subsequently, this prediction is validated against a validation data set. The goal of cross validation is to define a dataset to "test" the model in the validation phase, in order to limit problems like overfitting and give an insight on how the model will generalize to an unknown (future) dataset.

The immediate question that arises here is regarding the ratio of durations for training and prediction. Considering a large duration for training might improve the accuracy and robustness of our predictions. However, we also do not want our predictions to be unnecessarily biased by past anomalies either. Considering this trade off, we have currently used a (2 week:1 week) training:validation ratio in our analysis and experiments. Another factor that influences this decision is the granularity of prediction that is required. In production, we envision this forecasting to be performed on a weekly basis, therefore we predict for a duration of one week.

In general, given a training-validation ratio of $t : v$ (where t and v are measured in weeks), we consider a sliding window of $(t + v)$ weeks duration. Figure 7 shows the sliding window of 3 weeks corresponding to our 2:1 training-validation ratio. The training is performed on the first t weeks. The predictions yielded in the training phase are validated on the v weeks of validation duration. Then we slide the window by a week, and repeat the procedure. Below we describe the training and validation phases in detail.

5.3.1 Training phase

The training phase is shown in Figure 7, with time points T_{start} and T_{end} indicating its start and end times. For every week of the

training period (2 weeks in our case), we bound the quiesces and resumes using polygons as shown in Figure 6. Then, we compute the maximum height of all the polygons seen during training. Let this maximum height be $Train_{max}$. Intuitively, $Train_{max}$ is the largest churn seen during training. As it turns out, $Train_{max}$ itself is in fact a decent indicator of the required reserve capacity for the following validation period. Therefore, we derive simple prediction schemes as functions of $Train_{max}$. The output of the training phase is the predicted reserve capacity C which is given by the following equation.

$$C = f(Train_{max}) \tag{5}$$

While complex prediction strategies are possible, our aim is to choose the simplest strategy that performs well in production. Therefore, we consider simple functions of $Train_{max}$ that tilt the predictions to be either aggressive or conservative. Some prediction strategies that we have considered are given below.

$$f(Train_{max}) = \begin{cases} 1.75 \times Train_{max}, \\ 1.50 \times Train_{max}, \\ 1.25 \times Train_{max}, \\ Train_{max}, \\ 0.75 \times Train_{max} \end{cases} \tag{6}$$

5.3.2 Validation phase

During the validation phase, we use the value predicted in the training phase (C) as the available reserve capacity. Then, the time series is played out for the entire validation period. While playing out the time series, the reserve capacity is used to allocate resources whenever resumes occur. Conversely, when quiesce occurs, the reclaimed capacity is added to the reserve, until the predicted capacity(C) is reached.

As an illustration, consider the validation phase shown in Figure 7. The time points V_{start} and V_{end} indicate the start and end points of the validation phase respectively. Let C denote the predicted reserve capacity based on the training. At the beginning of the validation phase (i.e. at V_{start}), some of the reserved capacity might already be in use, and hence we cannot assume that the entire capacity C is available. We therefore make a conservative estimate of the initial state of the reserve capacity during the validation phase. Let $DC[T_{start} : T_{end}]$ denote the series of quiescent cores during the training phase and $DC[V_{start}]$ denote the number of quiescent cores at time V_{start}. We conservatively assume that the reserve usage at the beginning of the validation phase ($ReserveUsage[V_{start}]$) is given by the following equation.

$$ReserveUsage[V_{start}] = max(DC[T_{start} : T_{end}]) - DC[V_{start}] \tag{7}$$

From Equation 7, it follows that the available reserve capacity at the start of the validation phase (denoted as $AvailableReserve[V_{start}]$) is given by

$$AvailableReserve[V_{start}] = C - ReserveUsage[V_{start}] \tag{8}$$

With the above reserve capacity, our validation phase starts, playing out every time point. Consider the time point V_a in Figure 7, a local minimum point during week 3. Between V_{start} and V_a, there is a steep decrease in the number of quiescent cores, which essentially means that there have been a series of resumes during this period. This intuitively makes sense since V_a corresponds to the monday activity after a weekend. By the time the validation phase reaches V_a, all the DBs that have been resumed between V_{start} and V_a have received resources from the reserve capacity. Therefore, the reserve capacity that is available at V_a ($AvailableReserve[V_a]$) would be

$$\begin{aligned} AvailableReserve[V_a] = {} & AvailableReserve[V_{start}] \\ & - (DC[V_{start}] - DC[V_a]) \end{aligned} \tag{9}$$

Consider another time point V_b in Figure 7, which corresponds to a local maximum point during week 3. Observe that there is a steep increase in the number of DBs quiesced just before V_b. By the time the validation phase reaches V_b, all the DBs quiesced prior to V_b lead to freed capacity, which in turn goes back to the reserve capacity. Thus, at every point through the validation phase, the *AvailableReserve* goes up and down according to the quiesce and resume actions that have occurred.

For each minute time unit during the validation phase, information about the state of the cluster is captured. We capture the following information which is then aggregated over the required duration:

- **Unavailable cores:** If there are many more DBs resumed than our forecasted value, the reserve capacity gets fully used up, thereby leading to a situation where resuming DBs can no longer be accommodated in the cluster. This means that some resume actions fail due to unavailable capacity; this points to a non-robust prediction strategy. At every time unit during validation, we keep track of the number of cores that were required for resume but were unavailable.

- **Unavailable minutes:** Similar to *unavailable cores*, this metric captures the duration in which DBs couldn't be resumed. This is a clear indicator of the robustness of the prediction strategies.

- **Unused reserve capacity:** If there are fewer observed DBs resumed than predicted by our model, the reserve capacity remains unused during that time period. Having a lot of unused reserve capacity reduces the utilization levels of the cluster, which is against our goal. Therefore, this metric captures the extent of reserve capacity that remains unused, due to our choice of the prediction strategy.

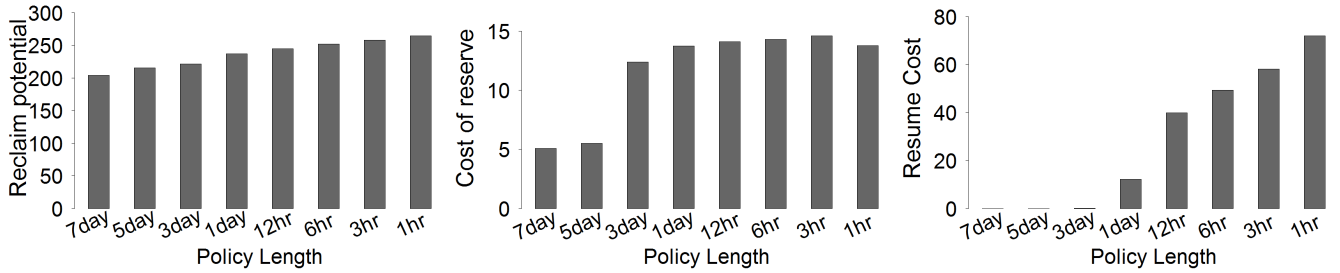
As we can see, these data points provide a generic way to compare different prediction strategies and measure their robustness and accuracy. An aggressive prediction strategy would have less unused reserve capacity, but may lead to more unavailable cores and minutes. On the other hand, a conservative prediction may avoid the situation of unavailable capacity for resumed DBs, at the cost of higher unused reserve capacity.

As we show in our experimental evaluation, this comparison helps in making a choice of how aggressive or conservative we wish to be, with our forecasting. At the end of the validation phase, the results are consolidated, and the sliding window moves ahead by a week. The entire forecasting process is repeated similarly.

6. RESULTS

We now present the results for our two stated problems in Section 3: (1) finding the right policy, and (2) how to properly forecast reserve capacities. These results are based on our analysis over the production traces of performance telemetry from two clusters over three months ending January 31st, 2016. One of the clusters is located in the US and the other cluster is located in Europe (denoted EU). Each of these datasets contains trillions of rows.

We note that the EU cluster has roughly four times more nodes than the US cluster and the EU cluster has a higher proportion of Basic and Standard DBs to Premium DBs in the population than the US cluster. For our model, we assume a one minute quiesce



(a) Reclaim Potential (US) (b) Cost of Reserve (US) (c) Resume Cost (US)

(d) Reclaim Potential (EU) (e) Cost of Reserve (EU) (f) Resume Cost (EU)

Figure 8: The total reclaim potential, reserve capacity cost, and resume cost for the US (a)-(c) cluster and the EU (d)-(f) cluster over a three month period as we vary the policy $P(T)$. Y-axis units are in millions of core minutes.

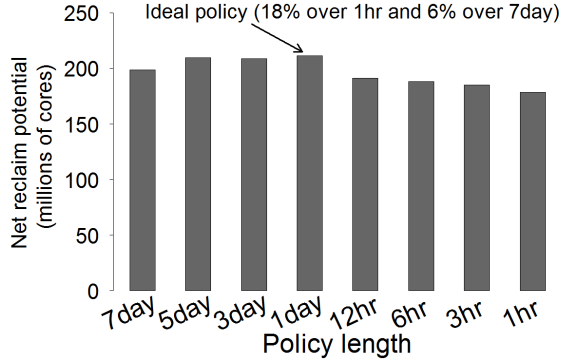


Figure 9: US cluster total *Net Reclaim Potential* over three months.

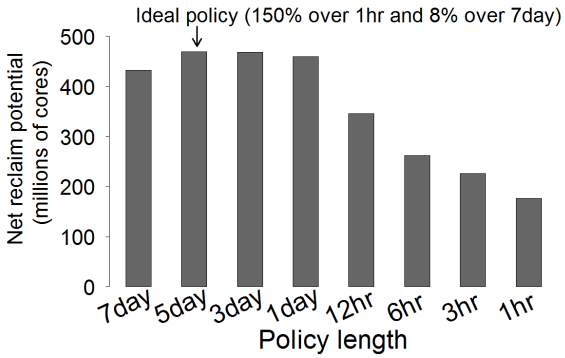


Figure 10: EU cluster total *Net Reclaim Potential* over three months.

and resume latency. Our analysis framework is implemented using SQL Server 2016 with R integration. Throughout the remaining discussion, we will highlight some key observations to take away.

6.1 Problem 1: Finding the Right Policy

In Section 5.1, we described three different metrics for our quiescence model: the reclaim potential, the reserve capacity cost, and the resume cost. By combining these 3 component metrics, we can determine the *net reclaim potential* per quiescence policy based on Equation 1, which answers problem 1.

6.1.1 Component Metrics

In Figures 8, for each metric, for each quiescence policy $P(T)$, we present the totals over the entire three month period as we described in 5.2. First, we consider the trend of reclaim potentials for each of the clusters shown in Figures 8(a) and (d). In both clusters, we see that the reclaim potential increases as we drop the policy length (increase idle sensitivity) from 7 days to 1 hour. The EU cluster (d) has substantially more Basic and Standard databases than the US (a) cluster and this visibly translates to a significantly higher reclaim potential. Perhaps what's interesting is the fairly linear increase in the measure as we change the policy's length non-linearly, which leads us to our first observation.

Observation 1: The amount of reclaimable capacity increases at a disproportionately higher rate as we decrease the quiescence policy (increase the idle sensitivity.)

However, we see that the rate of the increase in the reclaim potential between the two clusters is different: the relative difference between 7 day and 1 hour is 30% for the US cluster and almost 50% for the EU cluster.

Next, we examine the reserve capacity costs for the US and EU clusters (Figures 8(b) and (e) respectively). We see that both the 7 and 5 day policies for both clusters require relatively small reserve capacities. This is because with long policy lengths that span work

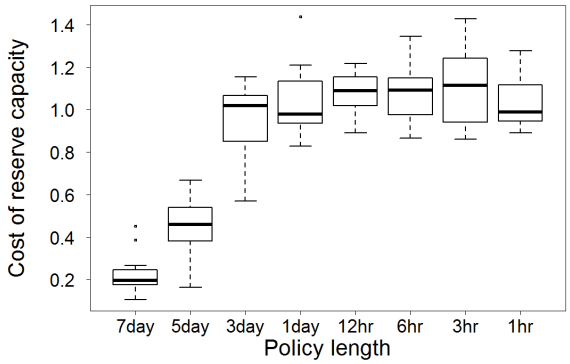


Figure 11: Distribution of the weekly cost of reserve capacity for the US cluster, for different quiescence policies. Y-axis units are in millions of core minutes.

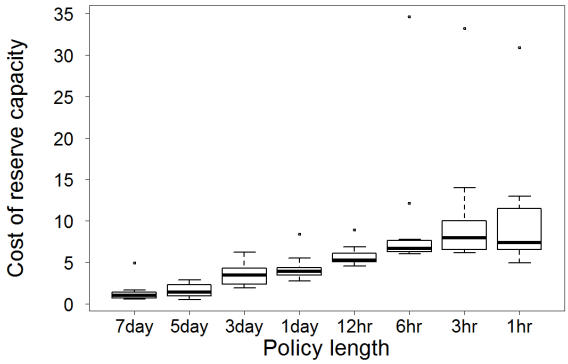


Figure 12: Distribution of the weekly cost of reserve capacity for the EU cluster, for different quiescence policies. Y-axis units are in millions of core minutes.

weeks, the analysis can no longer pick up on work week patterns. We observed this in Figure 5 where we saw virtually no periodicity in the Quiescent state in the 7 and 5 day policies. With the relatively linear trends in the quiescent state, indicating minimal net database churn, we don't need significant reserve capacity.

Observation 2: As we decrease the policy length $P(T)$, dropping shorter than the typical 5 day work week, (or other temporal patterns,) requires a substantial increase in reserve capacity.

Finally, in our examination of Figure 8, we look at resume costs as described in Section 5.1.3.What we immediately observe is that the results for the US (c) and EU (f) clusters look almost exactly the same except for the change in scale. Since this analysis is somewhat analogous to a histogram analysis on the idle periods of cloud databases, this indicates that despite the population size difference between the two clusters, the broad usage patterns in the populations are fairly similar.

Observation 3: We don't see significant resume activity until the policy length falls below 1 day indicating that if a database is idle for a day, it is likely to remain idle for a prolonged period.

6.1.2 Net Reclaim Potential

As we previously described, to determine the best policy $P(T)$ for a given cluster, we must be able to consolidate the three metrics presented in Figure 8 into the combined metric, the *net reclaim potential*. We present the net reclaim potential for all eight policies for both US and EU clusters in Figures 9 and 10 respectively.

It is important to see that even though in Figure 8(a) and (d), the reclaim potential for the shortest (most sensitive) policy we tested (1hr) was the highest for both clusters, this policy turns out to be the worst when the costs (reserve capacity cost and resume cost) are factored in. Furthermore, while we have included the resume cost here, there are other potential complexities in the system that increase with many databases constantly being quiesced and resumed. Of course, the resume cost model is not only hypothetical, but also conservative since the implementation of the resume operation will determine its user experience impact.

Observation 4: The high reclaim potential associated with shorter policies may not be able to overcome the potential cost of resume.

Finally, to directly answer problem 1, we find that the ideal policy length (according to the *net reclaim potential* measure) for the US cluster is 1 day while it is 5 days for the EU cluster. The 1 day policy in the US cluster provides 18% greater net benefit than the 1 hour policy and 6% greater benefit than the 7 day policy. In the EU cluster, the 5 day policy provides over 150% higher net reclaimable capacity over the 1 hour policy (owing to high resume costs, Figure 8(f)) and 8% more than the 7 day policy. The capacity reclaimed directly translates to additional profit as we can host other databases that we otherwise could not. Again, at billion dollar scale, such percentages over the life-time of the cluster mean 100s of millions of dollars in savings.

6.2 Problem 2: Forecasting and Evaluation

Now, we consider the problem of forecasting the reserve capacity. We first describe the characteristics of reserve capacity, and then show our results of comparing various prediction strategies.

6.2.1 Reserve capacity characteristics

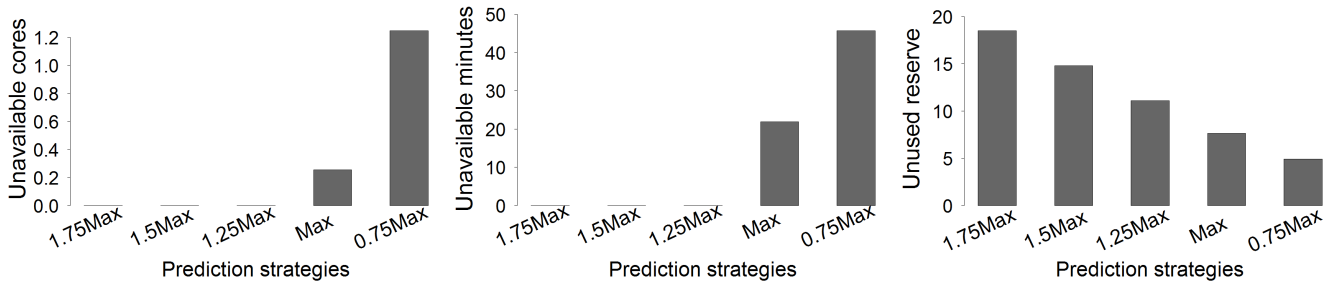
In order to forecast reserve capacity, it is important to understand the characteristics of the reserve capacity requirements in a cluster. In particular, it is useful to observe the variability of the required reserve capacity over time. We have therefore computed the cost of reserve on a weekly basis for the 3 month period. Figure 11 and Figure 12 show the distribution of these costs for the US and EU cluster respectively, for different quiescence policies.

The box and whisker diagrams of Figure 11 and Figure 12 depict the following information. The bottom and the top of each box are the first and third quartile respectively, and the band inside the box represents the median. The lower and upper ends of the whiskers represent the data points within 1.5 times the inter-quartile range of the lower and upper quartiles, respectively. The small circles depict the data points outside the whiskers, and represent outliers.

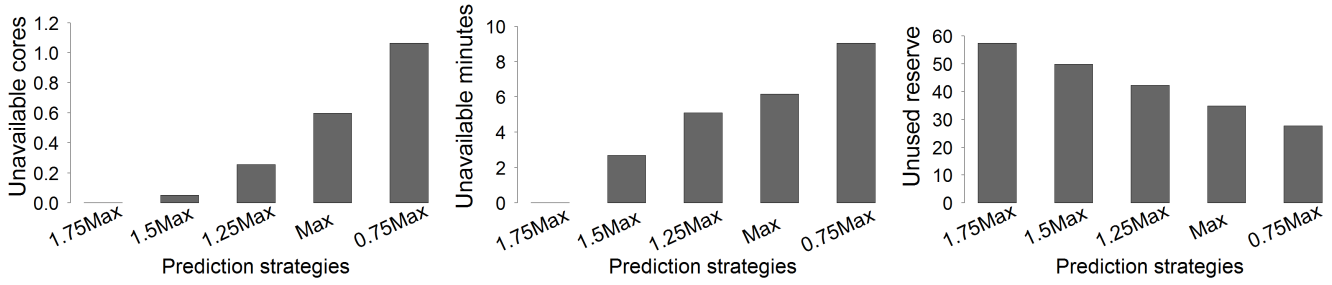
Observation 5: Irrespective of the quiescence policy chosen, the reserve forecasting has to be able to accommodate the variability observed in the cost of reserve capacity. We can also observe that the characteristics drastically vary across clusters, implying that the predictions have to be cluster dependent as well.

6.2.2 Comparing prediction strategies

We now show the results of our evaluation and comparison of various prediction strategies for forecasting reserve capacity. As described in Section 5, we have performed cross validation with a 2:1 training-validation ratio. During the validation phase, we play out the time series by using the predicted reserve capacity (Section 5.3.2) and capture information about the state of the cluster that helps in evaluating prediction strategies.



(a) Unavailable Cores (US) (b) Unavailable Minutes (US) (c) Unused Reserve (US)



(d) Unavailable Cores (EU) (e) Unavailable Minutes (EU) (f) Unused Reserve (EU)

Figure 13: The total unavailable cores (in millions), unavailable minutes (in thousands) and unused reserve capacity (in millions of cores) for the US (a)-(c) cluster (with quiescence policy $P(1day)$) and the EU (d)-(f) cluster (with quiescence policy $P(5day)$) over a three month period as we vary the prediction strategy.

Based on the observations we made in Section 6.1 for Problem 1, we have chosen the 1 day quiescence policy for the US cluster and the 5 day quiescence policy for the EU cluster. We have considered 5 prediction strategies as given by Equation 6. Figures 13 show the results of comparing the 5 strategies based on the metrics given in Section 5.3.2, aggregated over the entire 3 month period. The value $Train_{max}$ in Equation 6 is denoted as "Max" in Figures 13.

Figures 13(a) and (d) show the unavailable cores for different prediction strategies for the US and EU clusters respectively, and show similar trends.

Observation 6: The more aggressive the prediction strategy, the higher the number of unavailable cores, and hence the higher the number of DBs failing their resume.

The strategy "0.75Max" is the most aggressive among the strategies we have considered, and leads to many DBs failing their resume in both the clusters. The most conservative strategy is "1.75Max", which results in zero unavailable cores for both the clusters. Observe that "1.25Max" and "1.5Max' also lead to zero unavailable cores for the US cluster, but non-zero values for EU. Figures 13(b) and (e) show unavailable minutes (out of a total of 131,040 minutes), and follow the same trends as Figures 13(a) and (d). Using "1.75Max" on the EU cluster leads to zero unavailable time, but using "0.75Max" leads to 6% unavailable time (Figure 13(e)), .

Next, consider Figures 13(c) and (f). They show the aggregated unused reserve capacity over the 3 month period. Figure 14 and Figure 15 show the distribution of unused reserve over time, for the US and EU cluster respectively.

Observation 7: The more conservative the prediction strategy, the more the unused reserve capacity, and hence the lower the utilization of the cluster.

"1.75Max", being the most conservative of the strategies we consider, results in a lot of reserve capacity that remains unused. In fact, the unused reserve for "1.75Max" is at least 4 times more than that of "0.75Max" (the most aggressive strategy) in the US cluster.

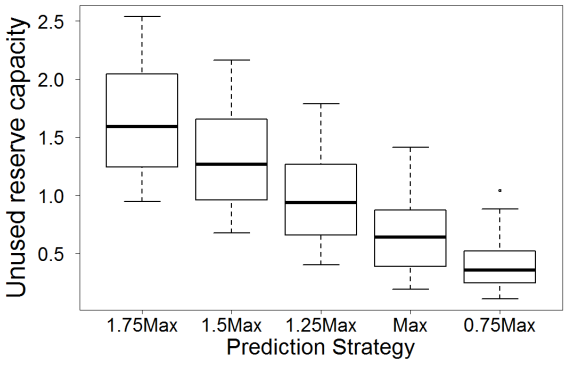


Figure 14: Distribution of weekly unused reserve capacity for the US cluster. Y-axis units are in millions of core minutes.

The differing variance across prediction strategies depicted in Figures 13, 14, and 15 also serve to reaffirm Observation 5 – that the policies and strategies chosen must be cluster dependent.

For instance, consider the US cluster (Figures 13(a)-(c)). Here, "1.25Max" is the best strategy to use, since it is always safe (i.e., zero unavailable cores in 3 months), and it uses the least reserve capacity compared to the more conservative predictions. However, this strategy is clearly not the right one for the EU cluster (Figures 13(d)-(f)), where "1.75Max" is the safest strategy. Also, observe that "Max" itself is not a safe enough forecast for either of the clusters. This is not surprising, given the growing number of DBaaS adoptions.

7. FUTURE WORK

In the context of this work, we believe there are at least two main categories of future work: (1) complex quiescence policies; and (2)

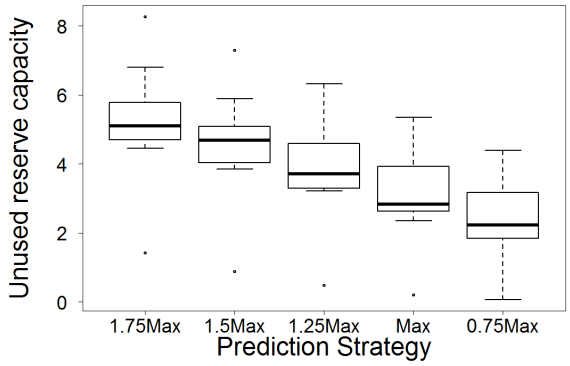


Figure 15: Distribution of weekly unused reserve capacity for the EU cluster. Y-axis units are in millions of core minutes.

pre-emptive resume. As we saw, even basic static policies can be very effective in reclaiming capacity. However, we did not consider dynamic policies or policies for sub-groups of user databases. As opposed to a static policy, a dynamic policy may change the idle length window depending on factors such as the time of the year or statistics of the user population. Furthermore, we employed a single policy for all databases whereas a more sophisticated scheme may consider classifying databases for different policies.

Another avenue of work involves attempting to pre-emptively provide resources and capacity to a database even before it needs it. In this way, we will not need to wait for a login or a query to be submitted before we decide to "resume" the database. This work can involve enhancing the mechanism, or the policy, or both. Determining when and which databases to resume, is an interesting area of future work.

8. CONCLUSIONS

In this paper, we described real-world problems that arise from over-booking databases onto a database-as-a-service cluster. We discussed these problems in the context of a simplified quiescence mechanism of detaching a database from its database engine process to reclaim cluster capacity. The problem is having to set aside reserve capacity to mitigate changes in the aggregate user behavior and handling databases that need to be resumed. In our work, we explored the policies that we may put in place to control the use of this mechanism as well as developed an analysis model and several metrics for evaluation of our scheme. The broader take-aways from our work are that, first, we must examine each cluster independently to determine the ideal policy to employ as the volume and usage patterns of the users vary. We showed that the benefits of choosing the right policy over a less-optimal one can very significant considering the scale at which we are operating. Secondly, if we continuously study historical data, we can provide accurate forecasts of the necessary amount of reserve capacity for the near future. This is required in order to to adapt to changes in the user population, the service offering, and even the infrastructure. We believe that this paper represents key steps towards capacity optimization in the cloud with many others still to follow.

9. ACKNOWLEDGEMENTS

The authors would like to thank Rohan Kumar and Mike Zwilling for their support and Jignesh Patel, Alan Halverson, Eric Robinson, Rimma Nehme, Raghu Ramakrishnan for their comments.

10. REFERENCES

[1] S. Aulbach, T. Grust, D. Jacobs, A. Kemper, and J. Rittinger. Multi-tenant Databases for Software as a Service: Schema-mapping Techniques. In *SIGMOD*, pages 1195–1206, 2008.

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

[3] 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.

[4] C. Curino, E. Jones, R. Popa, N. Malviya, E. Wu, S. Madden, H. Balakrishnan, and N. Zeldovich. Relational Cloud: A Database Service for the Cloud. In *CIDR*, 2011.

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

[6] 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.

[7] S. Das, V. Narasayya, F. Li, and M. Syamala. CPU Sharing Techniques for Performance Isolation in Multi-tenant Relational Database-as-a-Service. In *PVLDB*, 2013.

[8] S. Das, S. Nishimura, D. Agrawal, and A. El Abbadi. Albatross: Lightweight Elasticity in Shared Storage Databases for the Cloud using Live Data Migration. *PVLDB*, pages 494–505, 2011.

[9] J. Duggan, U. Cetintemel, O. Papaemmanouil, and E. Upfal. Performance Prediction for Concurrent Database Workloads. In *SIGMOD*, pages 337–348, 2011.

[10] A. J. Elmore, C. Curino, D. Agrawal, and A. El Abbadi. Towards Database Virtualization for Database As a Service. *PVLDB*, pages 1194–1195, 2013.

[11] A. J. Elmore, S. Das, D. Agrawal, and A. El Abbadi. Zephyr: Live Migration in Shared Nothing Databases for Elastic Cloud Platforms. In *SIGMOD*, pages 301–312, 2011.

[12] 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.

[13] A. Floratou and J. M. Patel. Replica Placement in Multi-tenant Database Environments. In *International Congress on Big Data*, 2015.

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

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

[16] 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.

[17] U. F. Minhas, S. Rajagopalan, B. Cully, A. Aboulnaga, K. Salem, and A. Warfield. RemusDB: Transparent High Availability for Database Systems. *VLDBJ*, 22(1):29–45, 2013.

[18] 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.

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

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

[21] V. Narasayya, I. Menache, M. Singh, F. Li, M. Syamala, and S. Chaudhuri. Sharing Buffer Pool Memory in Multi-tenant Relational Database-as-a-service. *PVLDB*, pages 726–737, 2015.

[22] V. R. Narasayya, S. Das, M. Syamala, B. Chandramouli, and S. Chaudhuri. SQLVM: Performance Isolation in Multi-Tenant Relational Database-as-a-Service. In *CIDR*, 2013.

[23] 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.

[24] C. D. Weissman and S. Bobrowski. The Design of the force.com Multitenant Internet Application Development Platform. In *SIGMOD*, pages 889–896, 2009.

[25] 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.