Analyzing and Optimizing NoSQL Workloads for Cosmos DB using Distressed Resource Volume Metric
[Experiment, Analysis & Benchmark]

Gunika
M365 Research, Microsoft
Bangalore, India

Aashutosh A V
M365 Research, Microsoft
Bangalore, India

Pooja Srinivas
M365 Research, Microsoft
Bangalore, India

Yogesh Simmhan
Indian Institute of Science
Bangalore, India
simmhan@iisc.ac.in

Ayush Choure
M365 Research, Microsoft
Bangalore, India
aychoure@microsoft.com

Harshit Shah
Azure, Microsoft
Bangalore, India
harshitshah@microsoft.com

Mayukh Das
M365 Research, Microsoft
Bangalore, India

Prashant Sasatte
Azure, Microsoft
Bangalore, India

Chetan Bansal
M365 Research, Microsoft
Redmond, USA

Abhijit Pai
Azure, Microsoft
Bangalore, India

Suraj Dixit
Azure, Microsoft
Bangalore, India

Achint Agrawal
Azure, Microsoft
Bangalore, India

ABSTRACT

Large scale managed cloud databases leverage sophisticated load packing algorithms, which provide the efficiencies and economy of scale necessary for running these services. Efficient packing of these database workloads onto resources is an active area of research. However, research into optimizing cloud databases at truly massive scales is limited due to a lack of access to public workloads, thus limiting the development of practical solutions that scale well. We address this in the context of *Cosmos DB*, Microsoft's flagship cloud-hosted NoSQL database service available in all Azure regions, through a novel problem definition of balancing packing efficiency against user-centric reliability metrics. We first propose *open-source NoSQL workloads* from real Cosmos DB clusters, and analyze these benchmark-workloads to derive a novel reliability metric, *Distressed Resource Volume (DRV)*, which captures the quality of service experienced by the end user. We then develop an *open-source policy simulation framework,* LoadStar, powered by a nonparametric statistical model of estimating the QoS of real traffic patterns. We define a *resource optimization problem* for placing Cosmos DB replicas onto VM nodes, develop a *forecasting model* for future load distributions, Luna, and a *placement algorithm* that uses these forecasts to trigger and rebalance stressed replicas, Orbit to reduce tail-errors. Our experiments, validated using LoadStar for these workloads, demonstrate Orbit's benefits over the existing Cosmos DB policy and a worst-fit optimized baseline, with higher load delivered at lower error rates and up to 35% reduction in resources. These have also been deployed in production, with potential annual savings of $100*M*s whilst also improving service reliability for millions of customers.

PVLDB Reference Format:
Gunika, Aashutosh A V, Pooja Srinivas, Yogesh Simmhan, Ayush Choure, Harshit Shah, Mayukh Das, Prashant Sasatte, Chetan Bansal, Abhijit Pai, Suraj Dixit, and Achint Agrawal. Analyzing and Optimizing Cosmos DB NoSQL Workloads using Distressed Resource Volume Metric [Experiment, Analysis & Benchmark]. PVLDB, 19(1): XXX-XXX, 2026.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://doi.org/10.5281/zenodo.15564578.

1 INTRODUCTION

The last decade has witnessed the proliferation of varied *NoSQL databases* to complement relational ones, including columnar databases, key-value stores and graph databases. Besides alternative data models, these also leverage sharding and replication to achieve high scalability on commodity hardware while ensuring durability. Alongside this has been the paradigm shift of business applications away from on-premise database deployments to large-scale *cloud-hosted managed NoSQL databases* such as Cosmos DB and Dynamo DB from Cloud Service Providers (CSPs) such as Microsoft Azure and Amazon AWS. Such a migration to the cloud lowers the capital expense costs for clients and out-sources their management overheads, besides enhancing availability and performance guarantees. At the same time, this has also shifted the responsibility for robust and efficient management of database operations to the CSPs.

The robustness guarantees of managed NoSQL databases have two competing objectives. *(1) Quality of Service (QoS) guarantees:* Ensuring high reliability and 6σ availability with strict adherence

Proceedings of the VLDB Endowment, Vol. 19, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

to Service Level Agreement (SLA) is essential to meet customer expectations. *(2) Resource efficiency:* Reducing the operational costs (*opex*, or Costs of Goods Sold, *COGS*) is fundamental for lowering the price for clients while ensuring profitability for the CSP. These are in opposition, with resource over-allocation being an easy approach to meet the high SLA but leading to higher opex costs. So, resource efficiency reduces to near-optimal *packing and allocation* of database entities onto compute resources, but is non-trivial in the presence of dynamic, sometimes unpredictable, load patterns that can impact reliability guarantees.

Resource management and load balancing are critical functionalities of cloud infrastructure orchestration that impacts cloud databases, with diverse CSP platforms such as AWS's Aurora [2], Google's Borg [32], Yahoo's Sherpa [12] and Microsoft's Service Fabric [11]. More specifically, resource management as the central theme of the design of enterprise relational databases such as BRAD [13] and Moneyball [24]. Alibaba's resource management framework, *Eigen* [15], optimizes utilization across active/warm/cold nodes for database workloads, with elements of forecasting and vector bin-packing across resource types. There is also much work on classical and cloud applications using vector bin-packing heuristics to solve this allocation problem [6, 21, 33, 34]. While bin-packing is NP-hard with poor approximation guarantees [5], the ability to forecast the node-level loads can make them tractable.

We consider these in the context of *Azure Cosmos DB*, a cloud-hosted NoSQL database from Microsoft that supports unstructured, semi-structured, structured, and vector data types. It uses a partitioned and replicated deployment for each customer database, with *replicas* placed on *nodes* hosted on Azure VMs with multi-tenancy.

Load-balancing optimization problem formulations often treat QoS or reliability as a *global constraint* in order to *optimize efficiency*. "Reliability" here is a general catch-all concept capturing end-user experience including all types of SLA guarantees. However, a key user-facing metric that is overlooked is one of *error rates* when performing database operations. These are affected by the stochastically varying demand and the replica placement due to stress on the underlying resource. These include latency, throughput, availability, etc. errors. Just setting minimum reliability constraints is either inadequate or, often, counter-productive to achieving efficiency and low error rates since the compute load has a direct probabilistic relationship with the nett load. Assuming all load distributions are equal, setting some global reliability constraints (e.g., CPU utilization $\leq 80\%$) does not eliminate *tail errors*. For products with large underlying infrastructure and a need to operate at very high availability, *these tail errors can add up to a failure event.*

We posit that developing a load optimizer that enhances usage efficiency (lower COGS) jointly with stochastic quantification of QoS, subject to varying and uncertain load patterns, is paramount. The key challenges in solving these are: (1) A dearth of principled and relevant real-world *workloads, metrics and techniques to quantify QoS (reliability)* against varying demand and load; (2) The absence of a *high-fidelity simulation framework* to emulate managed databases and rapidly validate strategies against this metric in an affordable manner, before deployment; and finally, (3) A lack of tractable, temporally-aware, adaptive *load balancing strategies* that optimize resource efficiency, subject to such a reliability metric. This EAB article addresses these gaps.

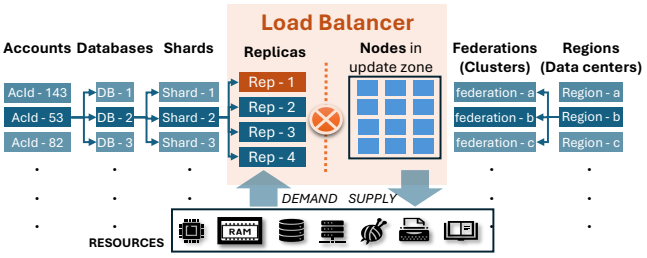


Figure 1: Cosmos DB resource provisioning hierarchy

In this paper, we make the following contributions:

(1) We propose a *open-source benchmark dataset* on NoSQL workloads over three Cosmos DB cluster, reporting resource loads and errors. We use this to develop a novel user-centric reliability measure, *Distressed Resource Volume (DRV)*, that reflects the QoS experienced by the end-user for the allocated compute capacity to their NoSQL database (Sec. 2).

(2) We develop a novel open-source *Policy Simulator framework, LoadStar*, which uses non-parametric statistical models based on real server logs from Cosmos DB clusters to accurately predict error rates with very low variance/noise for a given database workload and placement policy. This includes modeling node-level errors that help estimate DRV (Sec. 3).

(3) We define a resource optimization problem to place Cosmos DB shards and replicas onto VM nodes that uses the DRV metric. We design a *resource placement algorithm, Orbit*, to solve this optimization problem (Sec. 4), by leveraging a simple non-parametric distributional *forecast model, Luna*, for replica level loads based on a systematic study of load distributions across the replica population (Sec. 5).

(4) We perform detailed experiments to compare our proposed Orbit policy against the Simulated Annealing policy used in Cosmos DB and a worst-fit optimization baseline, evaluated using the policy simulator on the Cosmos DB workloads. We report the serving load at different error rates, the resource load uniformity, and the migration counts (Sec. 6). All these favor Orbit, and enable it to minimize resource usage – by up to 35% – and costs whilst keeping error rates within acceptable levels. These are being used in production in Cosmos DB, translating into resource savings of $100s of millions of dollars.

2 WORKLOAD ANALYSIS & DRV METRIC

Microsoft Cosmos DB [10] is a managed NoSQL database hosted on Azure cloud. Cosmos DB is active in 60+ global data centers, and it runs on 100,000s of nodes which host tens of millions of database shard replicas. The annual hardware costs for tens of millions of allocated CPU cores is significant. So, any improvements to COGS and QoS for Cosmos DB benefits the substantial user base and costs.

2.1 System Model of Azure Cosmos DB

A Cosmos DB *account* is an individual business client (Figure 1). Each account has multiple *databases*, each database has multiple collections, and every collection has several shards. *Shards* are formed from database partitioning and every shard generally has 4 *replicas*: one primary, which gets the first write and no read traffic,

Table 1: Workload Description of Different Clusters

Metric ↓ Cluster →	C1a	C1b	C2	C3
Time Duration (days)	60	13	54	54
# of Nodes in Cluster	196	196	196	190
# of Partitions	7,953	8,798	7,960	11,593
# of Replicas	$\approx 32k$	35,408	31,207	42,876
Max. Load on Cluster (RUs)	3.7M	3.7M	4.3M	8.5M
Max. CPU usage (agg. over cores)	1.83	.68	1.48	.79
Max. Mem. Usage (GB)	97.0	15.3	98.2	107.1
Max. Storage Usage (TB)	2.60	0.17	2.73	2.89

and three secondaries, which handle all read traffic and federated writes. Each replica is the unit of placement onto a node (VM).

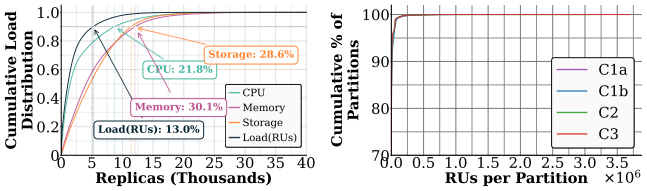
Cosmos DB uses the Azure Service Fabric (SF) [11] to manage Cosmos DB's infrastructure on 100,000s of *nodes*. A data center region has *clusters*, which contain 100s of nodes each. A shard usually has all its replicas in the same cluster. This streamlines the load balancing operations by limiting active load management and *replica migrations* to primarily within a cluster.

Each node has an upper bound on the number of replicas it can host, limited by a capacity constraint on CPU, Memory, Storage, I/O, Threads, Reads, and Writes. The individual resources are normalized into a standard measure, *Requested Units (RU)*, which is the unit of provisioning, allocation and billing of resources. RU is a unitless measure, and typically 1 RU is required to support a steady rate of 1 key-value lookup per second over a Cosmos DB database; create, update, delete and query operations are 5–10× costlier [20].

Each shard has a unique RU traffic profile. The *replica allocation strategy* determines if individual replicas will have enough resources. The general strategy to mitigate resource contention on a node is to redistribute some replicas to other nodes so that the new allocation is more balanced. However, this results in *replica migrations*, which consumes resource bandwidth and takes the migrating replicas temporarily offline. This evokes three key questions that form the *migration policy*: *who-to-migrate, when-to-migrate* and *where-to-migrate*, and impacts overall delivery stress, migration volumes, disruptive migration count, and the compute availability.

2.1.1 Simulated Annealing Policy. *Azure Service Fabric's Load Balancer (SFLB)* performs the load packing and migration based on *Simulated Annealing (SA)* [4, 26, 31], a statistical physics-based optimization technique for NP-complete scenarios. SFLB has two modes of triggering a re-balancing of replicas in a cluster: **(1)** Observing a *constraint violation* on an individual node, or **(2)** Observing the max/min "energy ratios" of SA crossing preset thresholds. Here *energy* is a user-defined abstract function that captures the instability of the system state, e.g., the ratio of maximum to minimum node utilization in the cluster. The underlying resource metrics (e.g., CPU usage, vacant RAM), the constraints on these metrics, and the energy function are configured for specific SFLB deployments. When a re-balancing is triggered, the algorithm does a time-bound search of neighboring configurations and moves to the (nearly) best configuration it finds. This reassignment is applied over a set of replicas-to-node mappings, with 100s of migrations done in parallel.

2.1.2 A major challenge? Although SA policies are expected to be efficient and locally effective, our early analysis showed a significant imbalance in replica assignments, which negatively impacts



(a) Cumulative Distribution across Replicas (all metrics) for C1b **(b) Cumulative Distribution across partitions**

Figure 2: Resource Usage Distribution

cluster-level utilization and migration costs at a cloud-scale. While load balancing and distributed database optimizations are well-studied, existing solutions take a restricted view. Some of them solve the system-level resource management problem irrespective of how cloud database workloads behave, while others try to address aspects of distributed databases, but either with constraints on stability and migration failures or with greater emphasis on query reliability [16, 27]. In particular, we identify the need for *fault-aware load packing and resource management* for managed NoSQL databases on the cloud that goes beyond reliability or cost constraints and jointly optimizes key dimensions of migration costs, workload performance, scalability and fault rates.

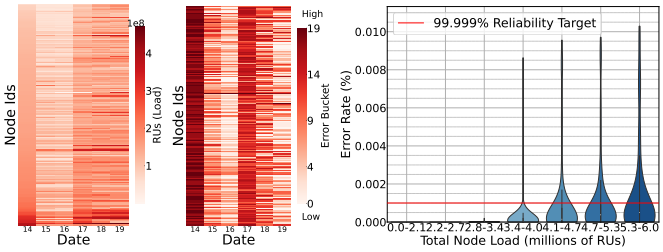
Some of the most crucial bottlenecks towards such scalable frameworks are: (1) The lack of open benchmarks of real *NoSQL workloads* and fault profiles of complex large-scale cloud-managed distributed databases, (2) The lack of *meaningful metrics* to jointly characterize relevant dimensions, and (3) A lack of a robust *policy simulator* to evaluate large-scale load profiles and fault/error distributions before deployment. The subsequent sections address these.

2.2 Cosmos DB NoSQL Workload Dataset

We propose an open benchmark dataset based on real-world NoSQL production workloads from Cosmos DB, and use it for our experimental analysis. We extract, transform, and consolidate performance telemetry of Cosmos DB and the underlying normalized hardware for NoSQL requests with varying proportions of loads (RUs). These workloads are from diverse applications, such as LLM infrastructure, e-commerce websites, financial products, and enterprise productivity software deployed in North American data centers. They have been suitably anonymized and masked so as to not reveal any specific (de-anonymizable) details about the clients, topology, or fine-grained infrastructure.

The workloads are from *three Cosmos DB clusters*, C1, C2 and C3, with over 190 VMs each hosting about 8,000 shards, serving up to 8.5M RUs of load (Table 1). Of these, C1a and C1b come from the same C1 cluster but for different time periods, with the smaller dataset C1b better suited for computationally expensive ablation studies or creating train–test data separation. All the clusters have homogeneous nodes with identical resources. Individual nodes have a memory threshold of 380GB and storage of 5.2TB. While the CPU core usage ranges from [0.0, 1.0], we do not normalize across cores; hence, the utilization per node can add up to more than 1 with the per core usage bounded at 0.6. The workload artifacts are at https://doi.org/10.5281/zenodo.15564578.

Figure 2a illustrates for **C1b** a Cumulative Distribution Function (CDF) plot of the four key resource dimensions: request load (RU),



(a) Total load (RUs) delivered by Nodes **(b) Total Error Rate across Nodes** **(c) Distribution of Error Rates across node load bins**

Figure 3: Load and error rate analysis for Cluster C1b. (a, b) Heatmaps of load and error distribution for nodes in March 2025. (c) Error rate distribution for node load bins.

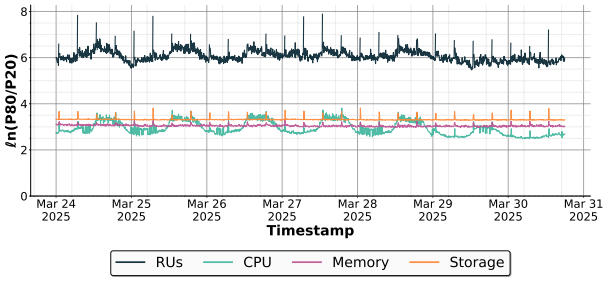
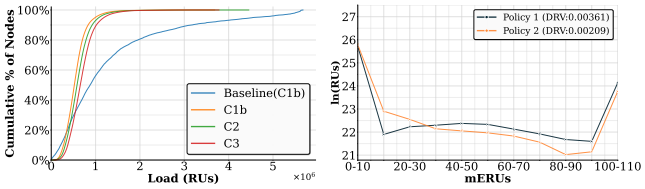


Figure 4: P80/P20 plots for all metrics for Cluster C1b.



(a) Cumulative Percentage of Nodes as a function of number of RUs on clusters Baseline(C1b), C1b, C2, C3

(b) Example DRV profiles for 2 migration policies (*ln* of requests)

Figure 5: DRV profile of observed and predicted ERU values

CPU, memory and storage, with increasing replica counts. Although correlated, differences in usage among these resources, especially at the *tails* – since the CDFs do not converge at the same point – indicate unbalanced load packing. The CDF of partition level load (RUs) in Figure 2b compares different clusters and shows that the same skew present across them. These tail-effects become crucial at our operating scale.

The heatmaps in Figure 3 compare the load (RUs) and errors on the C1b cluster through a week. This reveals the complexity of the problem. In (a), nodes are ordered by increasing RUs (load) from top to bottom, and (b) shows their error rates. We see no discernible pattern or visual correlation. However, at a finer granularity, error rates are almost always directly proportional to RU loads. *This highlights the limitation of current workload packing policies based purely on resource constraints.* Since Cosmos DB guarantees *five-nines* (99.999%) of availability, *tail errors* at 1 ppm merely manifest as borderline SLA violation. This requires us to jointly analyze the RU load and error rate distributions.

2.3 Reliability and Efficiency Tradeoffs

A key aspect of this article is to practically improve *experienced reliability*, beyond mere SLA constraints. While the load optimization policy aims to optimize efficiency of compute allocation, the reliability experienced by end users determines if a particular replica-node assignment is of acceptable *quality*. Cosmos DB has a target SLA of five-nines of availability. It has a comprehensive error tracking system that logs every low-level failure with specific *error codes*. As replicas run out of resources, errors related to latency, throughput, availability, etc. start growing. At certain threshold, SLA monitors will report *availability incidents*, requiring intervention. Reliability and utilization efficiency form competing objectives here, which makes this more nuanced than a simple bin-repacking.

Figure 3c shows a distribution of error rates reported on replicas as the load increases. Here, we simulate loads of up to 6.3M RUs per node on a cluster of 213 nodes, and sample logs every 5 mins to record the error rates. The min-max RU load range are binned into seven equal-width intervals, except for the initial underflow bin. We resample the binned data using probabilistic bootstrapping and Kernel Density Estimation (KDE) to ensure all bins have the same sample size and obtain a continuous data density across the bins.

The adverse impact of load on error is clear. At RU loads of up to 3.46M, the errors rates are low and well below the QoS threshold of 0.001% (99.999% reliability). Beyond this, however, the fourth bin has 1.2% of samples exceeding 0.001% error rate and last bin has 39.6% of violations exceeding 0.001%; errors above 0.01% are treated as anomalies and censored. There is a clear monotonic decline in reliability. While the throughput constraints are not yet being violated, the error rate is impacting the QoS for the packing policy used. Since errors climb non-linearly with load, a more uniform load distribution will always provide better reliability than a lopsided ones. While this follows from Jensen's inequality, we later observe this in our empirical tests too. These motivate the need for *non-traditional user-facing metrics of reliability*, and specialized optimization strategies to address this at scale.

To capture the *uniformity of load and utilization*, we use the ***P80/P20*** *ratio of utilization on high-load nodes to low-load nodes*, i.e., ratio of the utilization values of the 80^{th} percentile node to the 20^{th} percentile node. In a perfect system with uniform load, $\frac{P80}{P20} \rightarrow 1.0$. Fig. 4 shows the actual ratio distribution ($\ln \frac{P80}{P20}$) of various resources and total load (RUs) over a week. The resources (CPU/RAM/Storage) are unevenly utilized and the RU load is even more skewed. So, *even if the average load per node is low, a placement strategy that leads to uneven distribution can create QoS hotspots.*

Figure 5a shows the real distribution of RUs across nodes in our benchmark datasets, C1b, C2 and C3. *All* of them have nodes that are serving > 3M RUs, which can lead to policy violations as see in Figure 3c, while *none* of them have very high average load density. Such a *long-tail phenomena* leading to higher error rates can be mitigated only by controlling the utilization variance. Next, we propose a novel metric to capture this user-facing QoS.

2.4 Distressed Resource Volume (DRV) Metric

Cosmos DB maintains error rates for various hierarchical *error codes* (or types) as a time series of counts by error code at the replica or node level. They can be aggregated for a shard, node, account, etc.

A naïve reporting of total error counts over long time periods has no statistical merit and can be misinterpreted. The generation

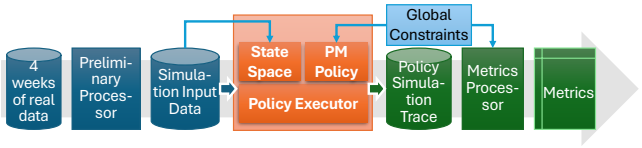


Figure 6: LoadStar Policy Simulation Framework for Cosmos DB Load Balancer

of an error code by itself may not be a concern. Although ideally no errors should occur, complex large-scale systems behave stochastically, even under no duress. At Cosmos DB's scale, with millions of partitions hosted on hundreds of thousands of nodes, some base emission of error codes will keep happening.

A clearer indicator of a fault is a *burst of error codes* emitted within a short duration. These indicate more significant non-transient SLA-impacting scenario. The RUs delivered to customers during these bursts are delivered at a QoS inversely proportional to the number of errors in that burst. This leads to our novel proposed metric, *Distressed Resource Volume (DRV)*, which reports the RUs (or any specific resource dimension, such as CPU cycles, RAM, storage, etc.) delivered at various levels of reliability (or conversely, distress).

Formally, let δ be a uniform time window (often a function of the sampling frequency of resource counters) within which we measure the burst volume for errors. For a given *node* M^i at *time* t, let $E^i([t-\delta, t])$ be the *number of errors* across all types emitted by this node between times $t-\delta$ and t. With $\mathcal{R}(M)$ as the set of *replicas* hosted on the node M, and I_t being the *resource consumption* of the specific replica, the RU-normalized error rate, called *ERU (Errors per RU)*, for node M at time t, is defined as:

$$\text{ERU}(M, t) = \frac{E_M([t-\delta, t])}{\sum_{I_t \in \mathcal{R}(M)} I_t}$$

This is essentially a measure of errors emitted *per unit* of load delivered (RU). Using this, we split the entire range of ERUs into a fixed number of uniform error bins, $EB_i (i \leq d)$, with the last overflow bin being unbounded. We then define the *DRV profile* as a d-dimensional vector, where the k^{th} component is the amount of load delivered at the ERU value belonging to the k^{th} bin. Formally,

$$\text{DRV}_k(\mathcal{M}, \mathcal{I}, \Omega) = \sum_{t, R(I_t) \in \mathcal{R}(M) : E(M,t) \in EB_k} I_t \qquad (1)$$

This novel profile vector captures the complete state of reliability and user-facing delivery quality.

Example. Figure 5b compares the impact of two different replica placement policies on the overall reliability, based on their DRV profiles. The Y-axis is the volume of compute delivered (RUs) and X-axis encodes the ERU bins. The area under this curve equals the total compute delivered. Different policies packing the same total RUs will have the same area under the curve but different slopes at various points. Ideally, the packing policy should have the most load delivered towards the left regions (low ERU and better reliability) and lower loads as we go right, having lower reliability. In this example, the orange policy in general delivers more RUs at a lower ERU rate compared to the blue policy.

3 LOADSTAR: A NOVEL POLICY SIMULATION FRAMEWORK

Before deploying any new resource management policies on massive-scale infrastructure, cloud providers validate their impact using policy simulators. This help uncover potential issues with poor performance, which can affect the experience of millions of users, or accrue tens millions of dollars of costs, before actual deployment. Existing packing and migration policy simulators used in industry tend to be proprietary. While they simulate the resource utilization behavior with the aim of minimizing constraint violations, they fail to provide insights into the experienced reliability of the end user.

We propose a new *open-source packing policy simulation framework, LoadStar,* which simulates the load and corresponding error rates for Cosmos DB. The key technical contribution is the *non-parametric modeling* of ERU–RU relationship; we demonstrate empirically the effectiveness of these techniques in the context of Cosmos DB workloads. This framework easily scales to multiple clusters across multiple weeks. The LoadStar simulation framework is now *actively used in Cosmos DB* to assess reliability profiles of various scenarios. Later in this paper, we use it to estimate the ERUs that will be generated for replicas based on the node-level RUs achieved by our packing policies for a given workload, i.e., go from estimated resource metrics to the estimated end-user QoS. LoadStar is available as part of the public artifact for this article.

3.1 Design

Figure 6 describes the overall schematic of LoadStar with the following key steps and modules. The *Preliminary Processing* module converts the raw server logs for a past workload into a standard form, addressing source level idiosyncrasies, e.g., some hosting systems change the identifier of a replica every time it moves, and assigns unique IDs to each replica. The resulting *Simulation Input Data* is in a standardized form with just the necessary columns, viz., each replicas node's location at every timestamp, the utilization of all resources, and the errors being emitted.

The *Policy Executor* generates the simulation trace corresponding to the *new* policy encoded in the policy sub-module. It interfaces with several sub-modules, which correspond to functions defined in the problem formulation in Section 4.1. The *State Space Manager* is a global configuration cache that tracks the current placement of all the replicas. The *Packing and Migration* policy module implements the packing policy $\Omega(.)$ that is invoked along with contextual parameters, such as load (RU) forecasts, and is used for generating a time ordered sequence of node–replica configurations which are emitted into the *policy simulation trace.*

The *Metrics Processor* module first enriches the generated policy trace with error rate estimates using the models discussed in Section 3.2. The simulation trace is then converted to the same schema as the input simulation data. The metrics processor then generates the aggregate metrics (such as DRV) into a final metrics file, which can then be analyzed to understand the impact of the policy on the system behavior for the given workload.

This framework can also be used to simulate an alternate cluster configuration, by updating the default global state space manager configuration used for simulation with the updated configuration. We use this when reducing the node footprints in Section 6.3.

Table 2: Actual and predicted ERUs for Clusters 1a and 1b.

Statistical Measure	Actual ERU		Pred. ERU		|Observed - Predicted|	
	C1a	C1b	C1a	C1b	C1a	C1b
Mean	0.0069	0.0086	0.0068	0.0087	0.0001	0.0001
Median	0	0	0	0	0	0
Variance	0.0050	0.0083	0.0044	0.0085	0.0006	0.0002

3.2 Non-Parametric Models to Generate ERU Estimates for RU Levels

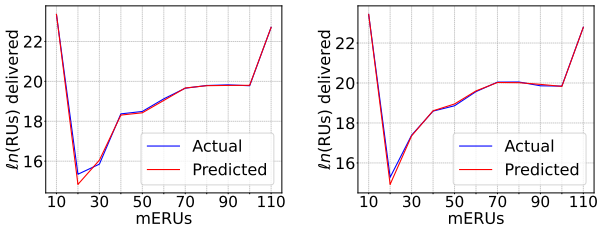
We use non-parametric statistical models to estimate the ERU values for given RU levels. These do not make any distributional assumptions and are effective in real data situations. We describe the method, followed by convergence tests on the C1 datasets.

3.2.1 Methodology. We split the given RU range in the input trace into bins, and for each bin we resample to achieve the target size of ERU entries per bin. For this, we randomly generate RU values within the bin range and find the corresponding ERU value using KDE, with a fallback on probabilistic bootstrapping. If the ERU data of the bin has enough variation (distinct values), KDE is used to smooth the estimates for ERUs corresponding to new load values in that bin. If the ERU data is sparse or has many duplicates, probabilistic bootstrapping is used by relying on frequencies of ERUs in that bin. Here, we resample ERU data points based on their probability of occurrence in that bin.

We group the data into 2000 bins based on the node's RU levels such that each bin represents an *equal percentile of total data*. Hence, for each bin, we store its node-level RU range and a distribution of the ERUs with their frequencies within that bin. To generate ERUs for a new policy based on node-level load, we identify the bin to which it belongs and use nonparametric estimation within that bin to generate corresponding ERUs using probabilistic sampling. This gives more weight to ERUs that occur more frequently for that bin.

3.2.2 Empirical Tests of Convergence Guarantees. We compare the predicted ERUs with the observed ones for the same load levels, and show qualitative and quantitative distributional similarities. These are done on **C1a** and **C1b** datasets. Figure 7 plots the observed and predicted DRV profiles for 100k data points and their quantitative metrics are shown in Table 2. The last column captures the l_1 norm of the prediction error. The l_2 norms of prediction error (RMS) are 0.1138 and 0.1254 for **C1a** and **C1b**, respectively. The substantial gap with l_1 error shows that the model is susceptible to outliers and the prediction errors in boundary regions (high ERUs) may be high. However, when used to validate system reliability, all ERUs beyond a certain level result in SLA failures and therefore their mutual distinctions are irrelevant. This is another reason why the last ERU bucket in DRV profiles is *censured* with no upper bound.

Lastly, we apply this non-parametric model to example the effects of a placement policy. Specifically, Figure 7 plots the *observed* DRV profiles in Cosmos DB for Cluster C1 using the Simulated Annealing policy, and the *estimated* DRV profile for the same policy and cluster given by LoadStar. The observed and estimated DRV profiles are almost indistinguishable. This is noteworthy since, in contrast, the heatmap in Figures 3a and 3b had no visually discernible correlations. But as claimed earlier, it was just an artifact of the granularity typical of statistical assessments; the ERU-RU model proposed in this section is able to learn those relationships.



(a) Cluster 1a (b) Cluster 1b

Figure 7: DRV profile of observed and predicted ERU values

4 FORECAST AWARE EFFICIENT PACKING POLICIES WITH ORBIT

In this section, we discuss the packing problem definition and the Orbit migration algorithm in detail. While we discuss the algorithm in reasonable detail, we keep the formal definitions concise and focus on the core decision points for the model.

4.1 Load Balancing Problem Formulation

4.1.1 Preliminaries. Let D^i indicate a database and P^i be partitions of a database. $\mathcal{R}$ is the set of all *partition replica instances* for partitions, with R^i indicating a single replica instance. $\mathcal{M}$ is a cluster and $M^i \in \mathcal{M}$ are *nodes* (VMs) in the cluster. $\mathcal{R}(M^i)$ is the set of replica instances hosted on a node M^i. In practice, all the nodes have identical resource capacities to ensure fungibility.

Let $I^i \in [0.0, 1.0]^d$ be the normalized $d-$dimensional *resource requirement vector* for a replica instance R^i. A value of 1.0 indicates 100% usage of that resource on a node. The key resource dimensions we consider are CPU, RAM, storage, read and write IO, and threads. I^i_t is the corresponding time-series object that captures the *resource usage vectors*, until time t, for that replica instance. Each node has *resource capacity constraints* on its total provisioning. So, the sum of resource requirement of replicas assigned to a node can never exceed 1.0 for each of its dimensions, $\| \sum_{R^j \in \mathcal{R}(M^i)} I^j \|_\infty \leq 1.0$. In practical scenarios, cluster owners will put additional *buffer* constraints on the node level provisioning, ensuring that the actual resource usage is even lower (e.g., CPU utilization is generally capped at 80%). We will refer to this constraint vector as $\mathbf{B}$; the node level resource bounds can now be expressed as $\sum_{R^j \in \mathcal{R}(M^i)} I^j \leq \mathbf{B}$.

Let C be the configuration matrix capturing the assignment of replicas to nodes; $C^i_j = 1$ if replica $R^j \in \mathcal{R}$ is assigned to node $M^i \in \mathcal{M}$, and 0 otherwise. Let $\mathcal{C}$ be the space of all these configurations. We define the *migration policy* as $\Omega : C \rightarrow C$ that, given the current configuration C_t at time t and the resource usage timeseries across all replicas $\mathcal{I}_t$ until now, finds the successor configuration C_{t+1} at time $t+1$ such that it has no capacity violations. For simplicity we assume such a feasible assignment always exists The policy will generate a new configuration only if the current configuration has violations, as discussed next. Treating C as a set of (replica-node) assignments, the symmetric difference between the two configurations, $|C_i \Delta C_j|$ is the *gap* between them, i.e., the number of replica migrations required to transition from C_i to C_j.

Let E^i_t be the total count of errors generated by node M^i at time t. $E^i(I_t)$ are the errors that will be generated by a node M^i when it is assigned some load of I_t. These I_t are $d-$dimensional resource vector which capture the *total* aggregated resource requirements

Algorithm 1 Triggering Re-balance *(Phase 1)*

Input: Configuration C, Predictions $P(I^j_{t+\Delta})\forall j$
Output: S, List of nodes whose total load needs reduction
for every node $M^i \in \mathcal{M}$ **do**
  if $\sum_{R^j \in \mathcal{R}(M^i)} I^j \not\leq \mathbf{B}$ **then** ▷ *Current Violation*
    $S = S \cup \{M^i\}$
  end if
  if $\sum_{R^j \in \mathcal{R}(M^i)} P(I^j_{t+\Delta}) \not\leq \mathbf{B}$ **then** ▷ *Predicted Violation*
    $S = S \cup \{M^i\}$
  end if
end for
return S

of all the replicas on a node. We have ${}^{C}E_t$ as the error generated jointly by all the nodes of the cluster in configuration C at time t.

4.1.2 Problem Definition. Given a cluster $\mathcal{M}$, node level resource provisioning constraint $\mathbf{B}$, a set of replica instances $\mathcal{R}$ assigned to it, along with their $d-$dimensional resource usage time series, a starting configuration C_0, and a migration policy Ω. $\Omega(C, \mathcal{R})$ is a *configuration path* in the space $\mathcal{C}$, the length of this path, denoted by $|\Omega(C, \mathcal{R})|$, is the number of times configuration change happened. Each of these changes may require multiple individual replica migrations, we refer to these as *composite* migrations. The goal is to find the best policy Ω that optimizes the loss metric defined below among these configuration paths. Since each cluster is self-contained, this problem is independently solved for each.

We define a *loss metric* $\mathcal{L}(\mathcal{M}, \mathcal{R}, \Omega)$ for the cluster specification $\mathcal{M}$, the instance loads for replicas $\mathcal{R}$, and the migration policy Ω to capture the cost of executing a specific migration policy. Some candidate loss metrics for evaluating Ω are:

(1) $|\Omega(\mathcal{R})|$ = the total number of *composite migrations*
(2) $\sum_0^{|\Omega(\mathcal{R})|} |C_{i+1} \Delta C_i|$ = the total number of *replica migrations*
(3) $\sum_{t, j \in \mathcal{R}, C \in \Omega(\mathcal{R}), t \in T} {}^{C}E_t$ the *total errors* emitted during the time period T.
(4) $\sum_{t, j \in \mathcal{R}, C \in \Omega(\mathcal{R}), t \in T} {}^{C}E_t . I^j_t$ the error level weighted sum of *compute load delivered* during the time period T.

Each of these metrics captures a key aspect of the load balancing costs. Optimizing the first one helps us contain the cost of actually computing the packing configurations repeatedly. In scenarios involving a large number of very small compute loads, calculating these configurations becomes expensive enough to preclude it from being done too frequently. The second metric that counts replica migrations focuses on the cost of moving a replica. Migration takes time, and creates temporary reduction in the RU serving capacity since the replica being moved cannot serve any read load till it comes online on the new node. The last metric is a QoS type metric focusing on keeping the error rates as low as possible. DRV is a denormalized version of this function and plays a pivotal role in formalizing our *reliable efficiency* objectives.

The *policy optimization* problem is to find an Ω which minimizes the total loss $\mathcal{L}$ over a replica population $\mathcal{R}$.

$$\Omega_{OPT} = \underset{\Omega}{\operatorname{argmin}}\, \mathcal{L}(\mathcal{M}, \mathcal{R}, \Omega) \tag{2}$$

where, $\mathcal{M}$ is a *fixed* cluster specification. We can say that the replica set $\mathcal{R}$ is sampled from an ideal distribution $\mathbb{R}$ to account for reasonable temporal variations in load time series. In our case, we have

Algorithm 2 Selection of Migrating Replica *(Phase 2)*

Input: Unstable node set S, Predictions $P(I^j_{t+\Delta})\forall j$
Output: R_S, List of replicas which should be moved out
for every node $M^i \in S$ **do**
  while $\sum_{R^j \in \mathcal{R}(M^i)} I^j \not\leq \mathbf{B}$ or $\sum_{R^j \in \mathcal{R}(M^i)} P(I^j_{t+\Delta}) \not\leq \mathbf{B}$ **do**
    if $\sum_{R^j \in \mathcal{R}(M^i)} I^j \not\leq \mathbf{B}$ **then** ▷ *Current Violation*
      $C = B_k$ ▷ *Pick resource dim. that is violated*
    end if
    if $\sum_{R^j \in \mathcal{R}(M^i)} P(I^j_{t+\Delta}) \not\leq \mathbf{B}$ **then** ▷ Future Violation
      $C = B_k$ ▷ *Pick resource dim. that is violated*
    end if
    $R^j = \textsc{BinaryFirst}(M^i, C)$ ▷ *Pick smallest replica that brings us within threshold*
    $R_S = R_S \cup \{R^j\}$
    $\mathcal{R}(M^i) = \mathcal{R}(M^i) \setminus R^j$
  end while
end for
return R_S

argued earlier that homogenizing utilizations leads to better QoS. In the following section, we propose a new model which achieves low P80/P20 ratios using load forecasts.

4.2 Proposed Load Balancing Model

Our Optimizing Resource Balancing in Time (Orbit) load balancing module is composed of three decision phases, viz., triggering, selection and placement determination. *Triggering* decides when to re-balance. These can be due to threshold breaches on resource capacity contentions, or more abstract reasons of attaining higher load uniformity across nodes. Once a trigger is provided, *selection* determines which replicas should be migrated. Finally, *placement* decides where each replica should be placed. Various approaches for these three phases can be combined for an overall load balancing strategy. We next describe the design of our load balancing model which utilizes the quantile forecasts proposed in Section 5.

4.2.1 Triggering Algorithm: When to Migrate Replicas out of a Node? We follow the standard node level constraint violation driven triggering policy with a simple generalization to accommodate the load forecasts. This algorithm marks a node for (out-)migration when either the current or predicted load (RU), CPU, memory, or storage utilization on a node violates capacity constraints. The capacity thresholds are usually set empirically based on past error behavior. Empirical proof of setting predictive windows is discussed in detail in the next section. This predictive mechanism aims to preemptively balance load, preventing a *late* migration when the node and replica are already overloaded and serving traffic, which can cause errors to start occurring and continue on/become worse till the migration is completed. Algorithm 1 formally defines these steps.

4.2.2 Selection Algorithm: Which Replica(s) Should Be Migrated from a Node? The various selection policies vary from lowest resource usage (minimize end user disruption) to the highest (minimize migration counts) [9]. To balance the user disruption and migration cost goals simultaneously, we instead take an intermediate approach. For every node marked for load reduction by the triggering phase, we arbitrarily choose a member from the set of

Algorithm 3 Placement of Migrating Replica *(Phase 3)*

Input: Unstable node set S, Migrating replica set R_S
Output: $\Omega : R_S \rightarrow \mathcal{M}$, map of replicas migrating into cluster
for replica $R^j \in R_S$, in decreasing order of median RU size **do**
  M = PolicyFit(R^j, $P(I^j_{t+\Delta})$)
  if $M == NULL$ **then**
    M = PolicyFit(R^j)
  end if
  $\Omega(R^j) = M$
  $\mathcal{R}(M) = \mathcal{R}(M) \cup R^j$ ▷ *R^j's load added into M for future decisions*
end for
return Ω

procedure PolicyFit(R, $P(I_{t+\Delta})$)
  Input: A replica R and its load forecast $P(I_{t+\Delta})$
  Output: Node M with the best score
  for node $M^i \in \mathcal{M}$ which can accommodate R **do**
    fit = WorstFitScore(M^i) ▷ *Empty load capacity available normalized by the maximum empty space*
    skew = $-\frac{1}{\pi}\arccos\frac{\langle I(P(M^i \cup I_{t+\Delta})), \not\in\rangle}{|I(M^i \cup P(I_{t+\Delta}))||\not\in|}$
    score[i] = fit + α.skew
    if $score[i] > max\_score$ **then**
      max_score = score[i]
      $max\_idx = i$
    end if
  end for
  return M^{max_idx}
end procedure

violated resource constraints and execute a binary search on an ordered (by the same resource) list of replicas to identify the *smallest replica set* that brings the breach below the threshold. We refer to this subroutine as **BinaryFirst**, with details given in Algorithm 2.

4.2.3 Placement Algorithm: Where to migrate the replica(s)? The third and last phase of load balancing determines the optimal destination node for every replica being moved. Our proposed algorithm uses a modified *worst-fit approach* that incorporates two scoring metrics that are combined linearly for a final score.
(1) *Load Factor:* The normalized available space on the node, calculated as the node's unused RU capacity divided by the maximum empty space across all nodes. This score captures the 'worst-fit' aspect of our policy and biases the selection towards *emptier* nodes.
(2) *Skewness:* This is a key component in all vector packing strategies [6, 15, 21] that helps reduce the original high-dimensional vector packing problem to a lower dimensional tractable version, while possibly losing some accuracy. We define skew as,

$$\text{skew} = -\frac{1}{\pi}\arccos\frac{\langle I(M^i), \mathbf{1}\rangle}{|I(M^i)||\mathbf{1}|} \tag{3}$$

Skew measures the angle between a node's normalized resource utilization vector $I(M^i)$ and the symmetric utilization vector (with all 1's). The minus sign and scaling normalize the value in the right direction. The policy goal is to find placements that minimize the angle. This helps avoid creation of node configurations with very skewed resource availabilities. E.g., a node with no storage available is unlikely to able to fill in its remaining CPU/RAM capacity as every replica will need some basic storage footprint.

The placement algorithm linearly combines these scores to generate a final worst-fit ranking for all potential destination nodes. The scale factor α used for combining the two scores is determined empirically in offline tests. Refer to Algorithm 3 for a formal outline of the Orbit placement algorithm. These policies depend on a suitable time series load forecasting model, which we propose next.

5 REPLICA LEVEL LOAD FORECASTING WITH LUNA

Mapping workloads to the available resources is a crucial scheduling problem to meet the required QoS. This reduces to an online stochastic bin packing problem which, in its most general form, is computationally intractable [21]. However, these are worst-case adversarial bounds and in cases like managed database services with long customer histories of compute requirements, *load forecasting* can potentially help overcome the complexity barriers and achieve better approximation ratios, as seen in earlier [9, 15]. Both the data context and the packing algorithm which will use the forecast are equally important in determining the outcome.

Specifically, the goal of our forecasting model is to *predict the demand for resources,* CPU, RAM, Storage and load (RUs). Formally, the *target variable* for the prediction is the resource usage $P(I^i_{t+\Delta}|I^i_{t-n})$ for a given temporal look-ahead window Δ, given the historical usage of n time units. Several salient aspects of our context govern our forecasting framework's design are:

- Low Granularity Forecasts: Since every node has substantial buffer capacity to support transient RU bursts, we ignore all volatility at time-granularity finer than 1-minute, and capture only systematic variations happening over 10s of minutes.
- Reasonable Forecast Window: Frequent replica migrations impacts both reliability and efficiency. A reliable forecast of, say, 2 hours into the future can help avoid consecutive migrations within that time window.
- Asymmetric Losses: Underestimating RU requirements impacts reliability due to over-packing of nodes while overestimating it causes efficiency loss due to under-used capacity. The optimal tradeoff point between reliability and efficiency is a product decision. The ideal forecasting model should work well on a wide range of these choices.
- Point vs. Distribution Estimates: While most predictive load balancing works rely on using good point forecasts [9, 16], they can be costly to compute. Instead, we posit that distributional forecasts are *much more robust* and provide sufficient information for a compatible packing policy.
- Spatial Scope: Our load forecasts are at the granularity of each replica instance, i, as required for the load balancing formulation in Section 4.1.

Load Utilization prediction for NoSQL workload Analytics (Luna) is simple predictive model that incorporates the above principles. The time series modeling problem is simplified to predicting the *peak load value* over 30 minute blocks. We use asymmetric loss functions that have a higher over-estimation penalty than under-estimation, implying reliability is more important than efficiency. Finally, we train one model for one specific quantile to keep the model complexity low.

We implement a time series forecasting framework for replica loads using Microsoft's *LightGBM* with quantile regression. We use *Whittaker-Eilers smoothing* to smooth replica-level RU time series

Table 3: Comparison of Our Forecasting Model with ARIMA and ETS (Pinball Loss) on RU Load for C1b

Percentile	Luna	ARIMA	ETS	Improvement over ARIMA	Improvement over ETS
0.001	58	3069	3751	98.1%	98.5%
0.25	1080	2800	3360	61.4%	67.9%
0.5	1420	2531	2968	43.9%	52.2%
0.75	1323	2262	2576	41.5%	48.6%
0.9	953	2100	2340	54.6%	59.3%
0.995	349	1997	2191	82.5%	84.1%

data, helping reduce noise while maintaining the trends in the data. The timestamps are grouped into *30-minute intervals*, ensuring a consistent time granularity for the prediction task. *Feature engineering* is driven by basic statistics, and includes time-based features such as the day of the week, hour of the day, and minute of the hour, rolling window statistics (mean, median, and quantiles) over multiple time horizons, along with higher-order moments (skewness and kurtosis) that help capture the temporal structure and volatility in the time series data. LightGBM's quantile regression uses Pinball Loss to predict specific percentiles of the time series.

5.1 Evaluation of Luna

Empirical tests are performed on the **C1b** workload, with 6 days of training data and another 6 days as test data. We first analyze the performance of predicting the total load (RUs) and benchmark against standard forecasting models. We then introduce a simple *coverage* metric relevant to our context that helps better visualize the performance of Luna's forecasts. We then briefly discuss the coverage behavior of RU and resource predictions.

5.1.1 Overall Load (RU) Prediction. We first evaluate Luna for RU predictions. We measure the Pinball Loss (quantile loss) [29] of these predictions to assess their statistical accuracy and compare our model with two classical time-series forecasting techniques:*ARIMA*, which trains a stationary time series model with seasonal adjustment enabled, and with autoARIMA for parameter tuning; and *Exponential Smoothing (ETS)* trained with a smoothing parameter of 0.2. Table 3 shows that our model has a much better accuracy than both ARIMA and ETS across load percentiles.

To better understand Luna's performance, we introduce a simple visual aid metric. Coverage Deviation $C(p_1, p_2)$ is the gap between the expected and the observed proportion of times when the actual load (RUs) l_t is falls between those percentile values, i.e., $C(p_1, p_2) = |\{t : l_t \in [L(p_1), L(p_2)]\}| - |p_2 - p_1|$. Here t is a *normalized* time between 0 to 1 and $L(p_i)$ is the predicted i^{th} percentile of the load (RU) value. The sign of this metric indicates in which percentile ranges the forecasts are systematically under/over-estimating. Figure 8 shows the coverage deviations for the RU predictions by Luna. The highest values appear in the top left region, which means that most of data appears between the lower percentile bands, e.g., [.01, .25]. This shows that our model is systematically *overestimating the loads*. An equivalent observation follows from focusing on the lowest negative value in the table and follows the same reasoning.

5.1.2 CPU/RAM/Storage Prediction. We repeat the same analysis with resource-level time series predictions. Figure 8 illustrate the

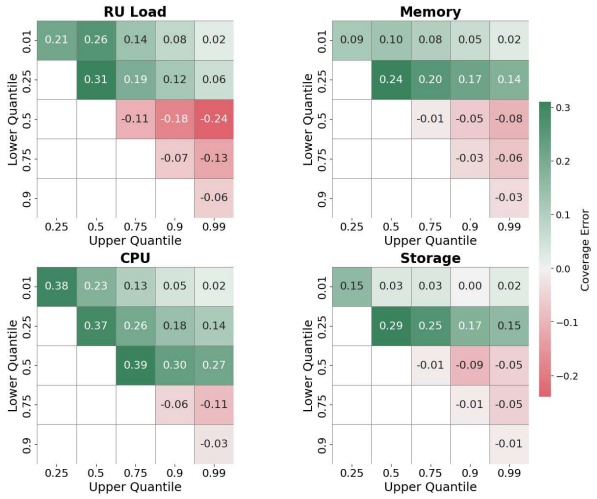


Figure 8: Coverage Deviations for RU Load, Memory, CPU, and Storage. Positive values indicate over-coverage; negative values indicate under-coverage for C1b.

coverage deviations of Luna on RAM, CPU, and Storage. While the pattern is similar to RU, that values are less extreme. So the model has an overestimation bias for all the resources' cases but it is weaker than that of RU. Table 3 shows that the absolute performance numbers, by themselves, it is not enough and the true test comes from it effectiveness for actual packing scenarios.

6 RESULTS ON PACKING POLICY

We evaluate our proposed Orbit packing policy based on the Luna prediction models using our LoadStar policy simulator, for the four different Cosmos DB cluster datasets we have provided. We first compare our Orbit policy against the existing Simulated Annealing (SA) method used in Cosmos DB and a baseline policy based on a Worst-Fit Optimization (WFO), primarily using 6 days of **C1b** workload for a detailed analysis (Sec. 6.1). We then show the generalizability of our approach to the other clusters, **C1a, C2, C3** (Sec. 6.2). Next, we examine the reduction in nodes possible using Orbit without significantly impacting the error rates (Sec. 6.3). Lastly, we discuss hyper-parameter tuning of the models (Sections 6.4, 6.5 and 6.6).

6.1 Comparing with Default & Baseline Policies

Besides the default SA algorithm used by Cosmos DB (Sec. 2.1.1), we design a non-trivial no-forecast baseline using a Worst-Fit Online (WFO) packing policy to illustrate the hardness of this problem. WFO uses constraint programming to formulate replica placement as a multi-dimensional bin-packing optimization problem [30] across the 4 resource dimensions of RU load, CPU, memory and storage. In each time period, it operates on each replica sequentially based on its current load, and chooses a node that it should be placed on based on maximum available capacity, i.e., worst fit. It preserves existing replica assignment to nodes across consecutive time periods if no change is required, and optimizes bin-packing only for newly-arriving replicas, with constraints updated by the removal of departing replicas. When incremental packing fails due to resource constraints, it triggers a full-repack, which is expensive.

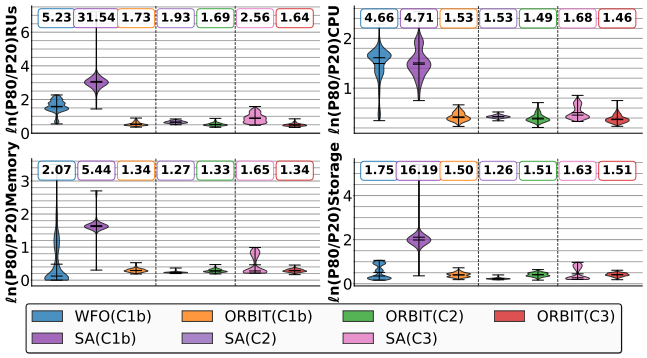


Figure 9: Comparison of $\ln\left(\frac{P80}{P20}\right)$ ratios between Orbit, WFO and SA on C1b, and Orbit and SA on C2 and C3. Mean value of load (non-ln) is shown in label.

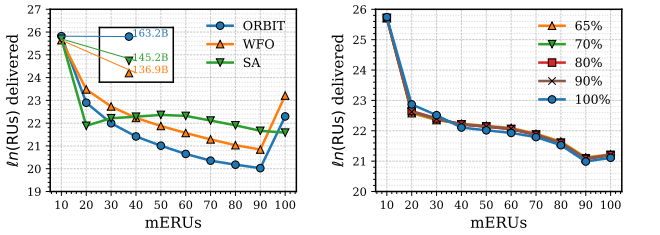
We implement WFO using Google's OR tools [23]. For the Orbit model, we use the default hyper-parameters of $\alpha = 4$, a forecast window of $2h$ ahead and a forecast percentile of $P50$ median chosen from the Luna model's output.

Fig. 9 (left) shows violin plot distributions of the natural log of the P80/P20 ratios for the four resource metrics for Cluster C1b, sampled every 5 mins, when this workload is packed using WFO (blue), SA (purple) and Orbit (orange), as reported by the LoadStar policy simulator. As we see, the ratios for WFO are better than SA, with lower variability and also resource ratios that avoid hotspots in the cluster. This confirms that WFO is a competitive baseline to compare against. We also see that Orbit does even better than WFO, and has an even tighter and lower violin with the median P80/P20 ratio ranging from 1.34–1.73 across the resource types. This shows substantial *improvement in the uniformity of utilization across nodes*, with little difference between cold (P20) and hot (P80) nodes. This improvement carries forth to C2 and C3 as well, with Orbit having lower ratios than SA; WFO could not complete on these larger clusters due to high time complexity.

This consequently improves the DRV profile of placements done using Orbit (Fig. 10a), which exhibits higher RUs delivered at lower ERUs, compared to SA and WFO. E.g., for the lowest error of 10 mERUs (Fig. 10a, inset), Orbit delivers 163B RUs, compared to 137B for WFO and 145B for SA. Since the total load is the same for all policies, the reliability is highest for Orbit.

The total number of migrations over 6 days is 19.3k for WFO, 34k for SA and just 47 for Orbit; but in the context of 35.4k total replicas, these are all modest migration numbers. While WFO accommodates the replica flux incrementally without requiring any regular migration, it does causes a full re-balance once during this period with 19.3k migrations taking place along with the associated flux; this peak is 18.7k migrations for SA and 7 for Orbit.

WFO cannot be used for online decisions due to its latency to solution. E.g., when < 500 replicas arrive/depart in a 5-min window, it takes 30–60s to solve, but this rises to 5–8 mins for 3000+ replicas and almost 1h for a full repacking. In contrast, Orbit takes under 30s to execute but, importantly, does not need to execute in real-time since it relies of Luna's forecasts that give it 10s of minutes if not hours of lead time to decide and execute the re-balancing.



(a) Comparison across strategies. (b) Node footprint % provided, with Orbit.

Figure 10: DRV profile for different strategies and hyper-parameters on Cluster C1b

6.2 Generalizability to Diverse Clusters

Next, we demonstrate the generalizability of the Orbit to different clusters, validating it over the bigger and qualitatively different cluster datasets, **C2** and **C3**. In Fig. 9 (right violins), we see that the P80/P20 ratios for our policy remains low using Orbit for C2 and C3 (green and red violins), showing a comparable if not better median ratio than even C1b with resource ratios ranging between 1.33–1.69 and 1.34–1.64, respectively. This confirms that our combine forecasting and policy approaches (Luna +Orbit) together are able to achieve a more uniform resource usage across all nodes, even with a longer time period of evaluation (54 days), and avoid hotspots that can lead to user-facing long-tail errors. The hyper-parameters for Orbit were themselves chosen based on a ablation study for the Cluster C1b, as discussed later, and tuning on such smaller datasets is still sufficient to generalize to larger clusters. The baseline WFO policy is not reported for these clusters due to the high time complexity (running into days/weeks) to solve for the large time periods for these datasets.

6.3 Resource Efficiency Gains

While the earlier results showed the improvement in tail errors and better user experience using Orbit, we use the same framework to explore potential reduction in resource allocation for then clusters to reduce the operational costs. Specifically, we *reduce the number of nodes* assigned to the C1b cluster in steps of 5%, from 100% to 65%, and use our Orbit for the placements; any lower and the replicas could not be placed. We report their DRVs in Fig. 10 and P80/P20 ratios in Fig. 11. As we see, as the resource footprint reduces, the P80/P20 resource deviations marginally change, e.g., from 1.73 for RU ratios with 100% nodes to to 1.60 for 65% nodes, which is actually an improvement. We also see that the DRV profiles are comparable even with fewer resources, with some higher RUs at higher error rates, e.g., delivering 151.2B RUs using all 100% nodes vs. a lower 147.5B RUs with only 65% nodes, at a low error rate of 10 mERU, and 1.47B RUs with 100% nodes vs. 1.65B RUs with 65% nodes, at a higher error of 100 mERU. Migrations caused remain small enough to not play a factor. Since the node resources are captively allocated to the cluster, a reduction in the node footprint directly results in *lower operational costs* for Cosmos DB.

6.4 Tuning Skewness Parameter α

Next, we outline the effect of different hyperparameter tuning on Orbit, with the best ones chosen in the results reported above.

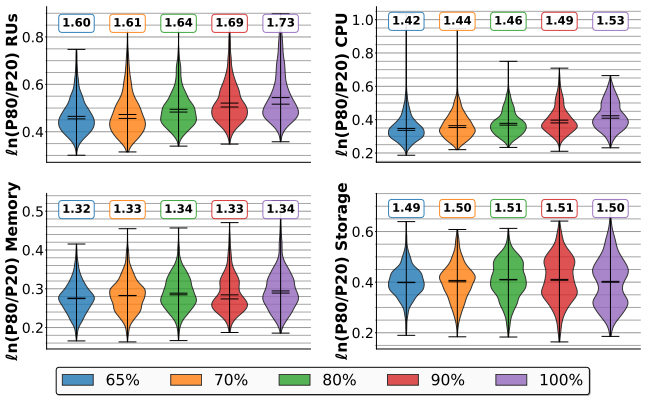


Figure 11: $\ln\left(\frac{P80}{P20}\right)$ ratio for each resource for nodes packed using Orbit for different *node footprints*, for Cluster C1b

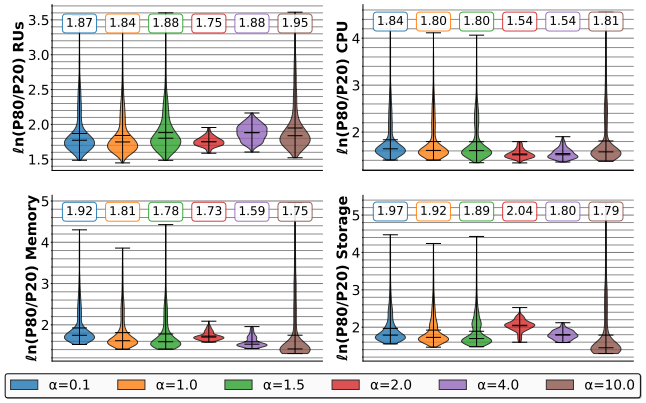


Figure 12: $\ln\left(\frac{P80}{P20}\right)$ ratio for each resource for nodes packed using Orbit at different α *skewness*, for Cluster C1b

We first compare different values of α, which is the weightage given to the *skewness* in Algorithm 3. This is to assess the relative impact on packing quality measured in terms of total migrations, utilization ratios, and skew distributions. Here we use a "perfect" (oracle) forecast since the goal is to select the most suitable value of α without biases introduced by our Luna forecasting model. Hence, we evalaute these on the first 6 days of the C1b dataset.

We try values of α ranging from 0.1 to 10.0, and observe their P80/P20 violins (Fig. 12). Values of skewness that cause upper long-tails and outliers are not suitable since they indicate hotspots among the nodes. As we eliminate such α values we are left with values in the middle ranges of Fig. 12. The α skewness also modestly affects the number of migrations that are triggered by our Orbit policy, but the counts are small enough (< 10) in all cases to not materially affect the performance. The DRV profiles at different α (not shown) are similar between 1.0 and 4.0, but worse at the extremes of 0.1 and 10. Hence, we fix the default as $\alpha = 4$ in all experiments.

6.5 Tuning Forecast Window for Packing Policy

We next evaluate various forecast time window sizes, i.e., how long ahead we forecast the resource loads using Luna model before passing it to the Orbit model. Typically, longer forecasts can lead to more stable packing but may also have higher forecasting errors. Forecasting noise can translate to more resource contention.

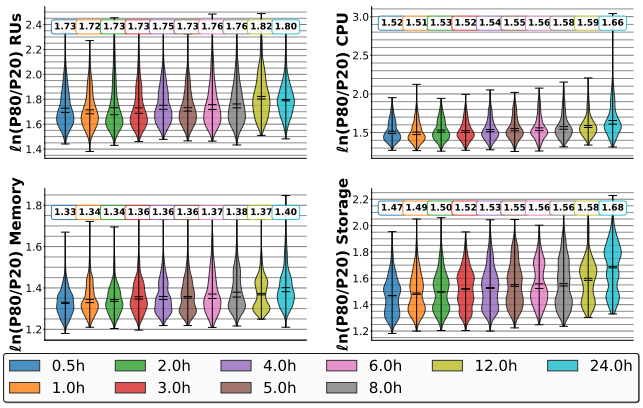


Figure 13: $\ln\left(\frac{P80}{P20}\right)$ ratio for each resource for nodes packed using Orbit at different *forecast windows*, for Cluster C1b

We evaluate time windows of 0.5h to 24h forward time windows, and include the *zero* time window as well, to be able to compare with the vanilla no-forecast version of the policy as a baseline. Fig. 13 reports the ln of the P80/P20 ratio for the resources. The first result that stands out is that not using forecasts (corresponding the 0 hr time window) creates multiple orders of magnitude of degradation in first few days and it continues to be poor even later. Secondly, among the different forecast windows $> 0h$, the relative differences in P80/P20 ratios and even the DRV profiles (not shown) are small, which indicates that even a 30 minutes advance knowledge of upcoming load can play a key role in better utilization ratios. Lastly, the performance of these models are not sensitive to small parameter changes, which makes them more generalizable.

As a result, based on the modestly lower P80/P20 ratios for RUs and fewer outliers for other resources, and with an intent to lower the cost and potential errors when computing longer windows, the $2h$ forecast window is used as the default parameter for Orbit.

6.6 Tuning Forecast Percentile

As seen in Section 5, the Luna forecasting model is tuned conservatively to have a higher under-estimation penalty than an over-estimation one, allowing for over-subscription buffer. From the forecast distribution, we need to pick one of the percentile values to pass onto the Orbit packing mode. The migration counts (not shown) for both very low and high forecast percentiles used for the packing policy gives suboptimal performance, leading to 1000s of migrations; the fewest migrations are for 25%'ile and 50%'ile. Low value < 20%'ile end up under-estimating the runtime traffic, causing migrations due to constraint violations, while high values > 60%ile estimates leads to frequent overestimation of future load and consequently preemptive migrations. This is further validated by Figure 14, where one can see the poor performance on both the $P10$ and $P90$ extremes. Consistently, the *DRV profiles* (not shown) for $P10$ and $P90$ are worse while $P25$, $P50$ and $P75$ are all better and comparable. Since ultimately the goal is to pack RUs we chose the $P50$ forecasts as the default in our Orbit model.

7 RELATED WORK

Benchmark Datasets for Cloud Workloads. Much the research around workload optimization, even for cloud-hosted databases, have leveraged Yahoo Cloud Service Benchmarking (YCSB) tool with Application Performance Manager (APM) modules [7, 25]. Another frequent benchmark for NoSQL DBs such as Redis is with YCSB tool [1]. This has even been used for testing different systems optimization frameworks [28]. SPEC IaaS application performance benchmark tool [22] is also a widely used for cloud applications. A recent survey enumerates numerous cloud traces [18]. However, almost all view this problem as a cloud resource management issue, and assume that generic solutions validated on the above benchmarks apply to large-scale managed cloud databases. *But they do not consider the nuances of distributed, managed and hosted database services, that bring with it subtleties such as redundancies (replicas), sharding, migrations, and faults.* Some works to attempt to narrow down on load optimization for database services the above, such as ResTune[35] that benchmark with OLTPBench TPC-C [8] and SYSBENCH. But they focus on either OLTP or OLAP databases but not both, and the benchmarks only characterize query distributions and transactions, not fault rates, packing densities or loads.

Database Partitioning and Placement. There are a number of existing works that examine placement of database partitions on distributed machines based on various quality and performance metrics. Some try and spread the load across servers for load balancing while others try and co-locate them to avoid distributed execution. *Zaharia et al.* [3] explore the placement of shards on different servers such that queries that access multiple shards can do so in parallel, i.e., shards that are concurrently accessed are placed on different servers to reduce the tail-latency for query execution in OLTP workloads. This is solved using an MILP formulation that decomposes the problem into sub-problems, which can each be solved within seconds.

Others have taken an opposing view, of trying to minimize the number of stores that are accessed to execute a singe query, with the goal of minimizing data movement and coordination [14]. *Accordion* [27] uses affinity between partitions, i.e., the those that are co-accessed within transactions, to estimate the load on servers hosting the partitions. It dynamically changes the partition placement on servers by solving a MILP optimization problem, to minimize the data movement while ensuring the throughput and memory capacity of a server is not violated and the replication factor is met.

In contrast, in our work, the database partitioning and replication factor have been decided by Cosmos DB. Our goal is limited to placing these partition replicas on the right set of machines, with fewer degrees of freedom. Further, we consider reliability as the primary metric rather than query latency. We also develop a forecasting model to predict and elastically change the placement.

Others have take a similar approach such as ours to use a forecasting model for query loads and a performance prediction model to decide placement strategies. *OptimusCloud* [19] does joint tuning of VM configuration and databases for NoSQL databases under dynamic workloads. *Wu et al.* [17] note that past and current load on the system is not a good indicator of future load. They predict the future transaction workload and use this for query routing, load balancing, re-partitioning and live migration. While we also employ forecasting models, in contrast, we use the Distressed Resource Volume (DRV) as a novel user-facing metric to optimize for. We leverage our non-parametric simulator, used in production, to evaluate our packing policies.

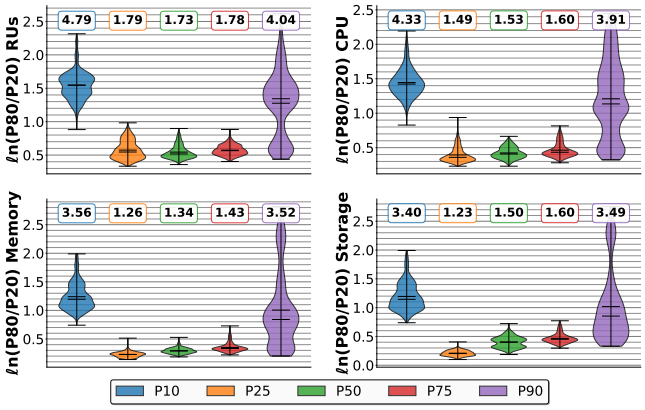


Figure 14: $\ln\left(\frac{P80}{P20}\right)$ ratio for each resource for nodes packed using Orbit at different *forecast percentiles*, for Cluster C1b

Eigen [16] uses a hierarchical scheduler for hot, warm and cold servers available to scheduling database instances. It does online resource allocation and offline rebalancing with migration for available machines, using vectorized resource optimizer (VRO), to pack instances into as few servers as possible. It uses warm machines as buffers to handle surges forecasted using exponential smoothing timeseries model. We only consider a single type of server and rather than the traditional approach of packing efficiency based on resource usage, we instead target the user-facing DRV metric, which better reflects the experienced reliability and any incidents that are raised. We also use a forecasting model to guide our placement, but also have a policy simulator to validate our proposed policies to offer high confidence before production deployments.

8 CONCLUSIONS

In this article, we have proposed open source benchmark datasets for NoSQL workloads based on real Cosmos DB traces, defined the DRV user-centric QoS metric, and developed the LoadStar simulator framework to evaluate the effect of placement policies on the SLA for Cosmos DB workloads. These are valuable resources for the cloud database community. Further, we have propose the Orbit algorithm that solves an optimization problem for replica placement to improve experienced errors by uses, which uses the Luna load forecasting model. These are validated using LoadStar for these workloads, and out-perform the Simulated Annealing model used in Cosmos DB and a worst-fit based solution. We see clear improvements in the load delivered at lower errors by Orbit, and can reduce the resource footprint by 35% without impact on the QoS. These are currently in production in Cosmos DB.

REFERENCES

[1] Mustafa Alzaidi and Aniko Vagner. 2022. Benchmarking Redis and HBase NoSQL databases using yahoo cloud service benchmarking tool. In *Annales Mathematicae et Informaticae*, Vol. 56. 1–9.

[2] Bradley Barnhart, Marc Brooker, Daniil Chinenkov, Tony Hooper, Jihoun Im, Prakash Chandra Jha, Tim Kraska, Ashok Kurakula, Alexey Kuznetsov, Grant McAlister, et al. 2024. Resource Management in Aurora Serverless. *Proceedings of the VLDB Endowment* 17, 12 (2024), 4038–4050.

[3] Nirvik Baruah, Peter Kraft, Fiodar Kazhamiaka, Peter Bailis, and Matei Zaharia. 2022. Parallelism-Optimizing Data Placement for Faster Data-Parallel Computations. *Proc. VLDB Endow.* 16, 4 (Dec. 2022), 760–771. https://doi.org/10.14778/3574245.3574260

[4] Dimitris Bertsimas and John Tsitsiklis. 1993. Simulated Annealing. *Statist. Sci.* 8, 1 (1993), 10 – 15. https://doi.org/10.1214/ss/1177011077

[5] Filipe Brandão and João Pedro Pedroso. 2016. Bin packing and related problems: General arc-flow formulation with graph compression. *Comput. Oper. Res.* 69 (2016), 56–67. https://doi.org/10.1016/j.cor.2015.11.009

[6] Henrik I Christensen, Arindam Khan, Sebastian Pokutta, and Prasad Tetali. 2017. Approximation and online algorithms for multidimensional bin packing: A survey. *Computer Science Review* 24 (2017), 63–79.

[7] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. 2010. Benchmarking cloud serving systems with YCSB. In *Proceedings of the 1st ACM Symposium on Cloud Computing* (Indianapolis, Indiana, USA) *(SoCC '10)*. Association for Computing Machinery, New York, NY, USA, 143–154. https://doi.org/10.1145/1807128.1807152

[8] Sudipto Das, Vivek R. Narasayya, Feng Li, and Manoj Syamala. 2013. CPU Sharing Techniques for Performance Isolation in Multitenant Relational Database-as-a-Service. *Proc. VLDB Endow.* 7, 1 (2013), 37–48. https://doi.org/10.14778/2732219.2732223

[9] Fahimeh Farahnakian, Tapio Pahikkala, Pasi Liljeberg, Juha Plosila, Nguyen Trung Hieu, and Hannu Tenhunen. 2016. Energy-aware VM consolidation in cloud data centers using utilization prediction model. *IEEE Transactions on Cloud Computing* 7, 2 (2016), 524–536.

[10] José Rolando Guay Paz. 2018. *Introduction to Azure Cosmos DB.* Apress, Berkeley, CA, 1–23. https://doi.org/10.1007/978-1-4842-3351-1_1

[11] Gopal Kakivaya, Lu Xun, Richard Hasha, Shegufta Bakht Ahsan, Todd Pfleiger, Rishi Sinha, Anurag Gupta, Mihail Tarta, Mark Fussell, Vipul Modi, Mansoor Mohsin, Ray Kong, Anmol Ahuja, Oana Platon, Alex Wun, Matthew Snider, Chacko Daniel, Dan Mastrian, Yang Li, Aprameya Rao, Vaishnav Kidambi, Randy Wang, Abhishek Ram, Sumukh Shivaprakash, Rajeet Nair, Alan Warwick, Bharat S. Narasimman, Meng Lin, Jeffrey Chen, Abhay Balkrishna Mhatre, Preetha Subbarayalu, Mert Coskun, and Indranil Gupta. 2018. Service fabric: a distributed platform for building microservices in the cloud. In *Proceedings of the Thirteenth EuroSys Conference* (Porto, Portugal) *(EuroSys '18)*. Association for Computing Machinery, New York, NY, USA, Article 33, 15 pages. https://doi.org/10.1145/3190508.3190546

[12] Markus Klems, Adam Silberstein, Jianjun Chen, Masood Mortazavi, Sahaya Andrews Albert, P.P.S. Narayan, Adwait Tumbde, and Brian Cooper. 2012. The Yahoo! cloud datastore load balancer. In *Proceedings of the Fourth International Workshop on Cloud Data Management* (Maui, Hawaii, USA) *(CloudDB '12)*. Association for Computing Machinery, New York, NY, USA, 33–40. https://doi.org/10.1145/2390021.2390028

[13] Tim Kraska, Tianyu Li, Samuel Madden, Markos Markakis, Amadou Ngom, Ziniu Wu, and Geoffrey X Yu. 2023. Check out the big brain on BRAD: simplifying cloud data processing with learned automated data meshes. *Proceedings of the VLDB Endowment* 11 (2023), 3293–3301.

[14] Chuan Lei, Abdul Quamar, Vasilis Efthymiou, Fatma Özcan, and Rana Alotaibi. 2022. HERMES: data placement and schema optimization for enterprise knowledge bases. *The VLDB Journal* 32, 3 (July 2022), 549–574. https://doi.org/10.1007/s00778-022-00756-y

[15] Ji You Li, Jiachi Zhang, Wenchao Zhou, Yuhang Liu, Shuai Zhang, Zhuoming Xue, Ding Xu, Hua Fan, Fangyuan Zhou, and Feifei Li. 2023. Eigen: End-to-End Resource Optimization for Large-Scale Databases on the Cloud. *Proc. VLDB Endow.* 16, 12 (Aug. 2023), 3795–3807. https://doi.org/10.14778/3611540.3611565

[16] Ji You Li, Jiachi Zhang, Wenchao Zhou, Yuhang Liu, Shuai Zhang, Zhuoming Xue, Ding Xu, Hua Fan, Fangyuan Zhou, and Feifei Li. 2023. Eigen: End-to-End Resource Optimization for Large-Scale Databases on the Cloud. *Proc. VLDB Endow.* 16, 12 (Aug. 2023), 3795–3807. https://doi.org/10.14778/3611540.3611565

[17] Yu-Shan Lin, Ching Tsai, Tz-Yu Lin, Yun-Sheng Chang, and Shan-Hung Wu. 2021. Don't Look Back, Look into the Future: Prescient Data Partitioning and Migration for Deterministic Database Systems. In *Proceedings of the 2021 International Conference on Management of Data* (Virtual Event, China) *(SIGMOD '21)*. Association for Computing Machinery, New York, NY, USA, 1156–1168. https://doi.org/10.1145/3448016.3452827

[18] Guozhi Liu, Weiwei Lin, Haotong Zhang, Jianpeng Lin, Shaoliang Peng, and Keqin Li. 2025. Public Datasets for Cloud Computing: A Comprehensive Survey. *ACM Comput. Surv.* 57, 8, Article 198 (March 2025), 38 pages. https://doi.org/10.1145/3719003

[19] Ashraf Mahgoub, Alexander Medoff, Rakesh Kumar, Subrata Mitra, Ana Klimovic, Somali Chaterji, and Saurabh Bagchi. 2020. OPTIMUSCLOUD: heterogeneous configuration optimization for distributed databases in the cloud. In *Proceedings of the 2020 USENIX Conference on Usenix Annual Technical Conference (USENIX ATC'20)*. USENIX Association, USA, Article 13, 16 pages.

[20] Microsoft. 2025. Azure Cosmos DB Capacity Calculator. https://cosmos.azure.com/capacitycalculator/.

[21] Rina Panigrahy, Kunal Talwar, Lincoln Uyeda, and Udi Wieder. 2011. Heuristics for vector bin packing. *research. microsoft. com* (2011).

[22] Alessandro Vittorio Papadopoulos, Laurens Versluis, André Bauer, Nikolas Herbst, Jóakim von Kistowski, Ahmed Ali-Eldin, Cristina L. Abad, José Nelson Amaral, Petr Tůma, and Alexandru Iosup. 2021. Methodological Principles for Reproducible Performance Evaluation in Cloud Computing. *IEEE Transactions on Software Engineering* 47, 8 (2021), 1528–1543. https://doi.org/10.1109/TSE.2019.2927908

[23] Laurent Perron and Vincent Furnon. [n. d.]. *OR-Tools*. Google. https://developers.google.com/optimization/

[24] Olga Poppe, Qun Guo, Willis Lang, Pankaj Arora, Morgan Oslake, Shize Xu, and Ajay Kalhan. 2022. Moneyball: proactive auto-scaling in Microsoft Azure SQL database serverless. *Proceedings of the VLDB Endowment* 15, 6 (2022), 1279–1287.

[25] Tilmann Rabl, Sergio Gómez-Villamor, Mohammad Sadoghi, Victor Muntés-Mulero, Hans-Arno Jacobsen, and Serge Mankovskii. 2012. Solving big data challenges for enterprise application performance management. *Proc. VLDB Endow.* 5, 12 (Aug. 2012), 1724–1735. https://doi.org/10.14778/2367502.2367512

[26] R.A. Rutenbar. 1989. Simulated annealing algorithms: an overview. *IEEE Circuits and Devices Magazine* 5, 1 (1989), 19–26. https://doi.org/10.1109/101.17235

[27] Marco Serafini, Essam Mansour, Ashraf Aboulnaga, Kenneth Salem, Taha Rafiq, and Umar Farooq Minhas. 2014. Accordion: elastic scalability for database systems supporting distributed transactions. *Proc. VLDB Endow.* 7, 12 (Aug. 2014), 1035–1046. https://doi.org/10.14778/2732977.2732979

[28] Gagan Somashekar, Karan Tandon, Anush Kini, Chieh-Chun Chang, Petr Husak, Ranjita Bhagwan, Mayukh Das, Anshul Gandhi, and Nagarajan Natarajan. 2024. {OPPerTune}:{Post-Deployment} configuration tuning of services made easy. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. 1101–1120.

[29] Ingo Steinwart and Andreas Christmann. 2011. Estimating conditional quantiles with the help of the pinball loss. *Bernoulli* 17, 1 (2011), 211 – 225. https://doi.org/10.3150/10-BEJ267

[30] Sebastián V. Romero, Eneko Osaba, Esther Villar-Rodriguez, Izaskun Oregi, and Yue Ban. 2023. Hybrid approach for solving real-world bin packing problem instances using quantum annealers. *Scientific Reports* 13, 1 (July 2023). https://doi.org/10.1038/s41598-023-39013-9

[31] Peter J. M. van Laarhoven and Emile H. L. Aarts. 1987. *Simulated annealing.* Springer Netherlands, Dordrecht, 7–15. https://doi.org/10.1007/978-94-015-7744-1_2

[32] Abhishek Verma, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. 2015. Large-scale cluster management at Google with Borg. In *Proceedings of the tenth european conference on computer systems*. 1–17.

[33] Prathamesh Saraf Vinayak, Saswat Subhajyoti Mallick, Lakshmi Jagarlamudi, Anirban Chakraborty, and Yogesh Simmhan. 2025. CARL: Cost-Optimized Online Container Placement on VMs Using Adversarial Reinforcement Learning . *IEEE Transactions on Cloud Computing* 13, 01 (Jan. 2025), 321–335. https://doi.org/10.1109/TCC.2025.3528446

[34] Wei Wang, Baochun Li, and Ben Liang. 2014. Dominant resource fairness in cloud computing systems with heterogeneous servers. In *IEEE INFOCOM*.

[35] Xinyi Zhang, Hong Wu, Zhuo Chang, Shuowei Jin, Jian Tan, Feifei Li, Tieying Zhang, and Bin Cui. 2021. ResTune: Resource Oriented Tuning Boosted by Meta-Learning for Cloud Databases. In *Proceedings of the 2021 International Conference on Management of Data* (Virtual Event, China) *(SIGMOD '21)*. Association for Computing Machinery, New York, NY, USA, 2102–2114. https://doi.org/10.1145/3448016.3457291