CDSBen: Benchmarking the Performance of Storage Services in Cloud-native Database System at ByteDance

Jiashu Zhang
Wen Jiang
Bo Tang*
Department of Computer Science and Engineering, Southern University of Science and Technology
zhangjs2018@mail.sustech.edu.cn
11810406@mail.sustech.edu.cn
tangb3@sustech.edu.cn

Haoxiang Ma
Lixun Cao
Zhongbin Jiang
Yuanyuan Nie
ByteDance
mahaoxiang@bytedance.com
caolixun@bytedance.com
jiangzhongbin@bytedance.com
nieyuanyuan@bytedance.com

Fan Wang
Lei Zhang
Yuming Liang
ByteDance
wangfan.666@bytedance.com
zhanglei.michael@bytedance.com
liangyuming@bytedance.com

ABSTRACT

In this work, we focus on the performance benchmarking problem of storage services in cloud-native database systems, which are widely used in various cloud applications. The core idea of these systems is to separate computation and storage in traditional monolithic OLTP databases. Specifically, we first present the characteristics of two representative real I/O workloads at the storage tier of ByteDance's cloud-native database veDB. We then elaborate the limitations of using standard benchmarks such as TPC-C and YCSB to resemble these workloads. To overcome these limitations, we devise a learning-based I/O workload benchmark called CDSBen. We demonstrate the superiority of CDSBen by deploying it at ByteDance and showing that its generated I/O traces accurately resemble the real I/O traces in production. Additionally, we verify the accuracy and flexibility of CDSBen by generating a wide range of I/O workloads with different I/O characteristics.

PVLDB Reference Format:
Jiashu Zhang, Wen Jiang, Bo Tang, Haoxiang Ma, Lixun Cao, Zhongbin Jiang, Yuanyuan Nie, Fan Wang, Lei Zhang, and Yuming Liang. CDSBen: Benchmarking the Performance of Storage Services in Cloud-native Database System at ByteDance. PVLDB, 16(12): 3584 - 3596, 2023.
doi:10.14778/3611540.3611549

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/DBGroup-SUSTech/CDSBen.

1 INTRODUCTION

The migration of many applications to the cloud has prompted the development of cloud-native database systems by large enterprises, such as Amazon Aurora [35], Microsoft Socrates [5], Huawei Taurus [13], and Alibaba PolarDB [8]. These database systems are designed to provide high availability, elasticity, and performance by decoupling the compute and storage tiers in traditional monolithic OLTP databases. This enables independent deployment and scaling of compute and storage services. In cloud-native database systems, application transactions are first converted to I/O requests through the compute tier, and then applied to the storage tier. Therefore, ensuring excellent storage services is crucial for achieving low application latency and high overall throughput.

Despite the importance of storage services, to the best of our knowledge, no tool could be utilized to evaluate the performance of storage tiers effectively. In this work, we focus on the performance evaluation of storage services in cloud-native database systems as the engineering team at ByteDance always faces the following difficult questions.

- How about the performance of the storage tier in veDB (the cloud-native database at ByteDance) if the transactions per second (TPS) doubles?
- What is the maximum throughput if the read-write ratio of transactions changes dramatically?
- What will happen if we change the techniques in the current storage service slightly, e.g., applying a new I/O requests scheduling strategy in veDB?

A comprehensive benchmark is necessary to effectively evaluate the performance of the storage tier in cloud-native database systems and answer the questions posed earlier. While benchmarks like TPC-C and YCSB have been proposed to evaluate monolithic database systems or KV storage engines [7, 11, 14, 16, 21], adapting them to evaluate storage services in cloud-native database systems is challenging. This is due to the lack of tuning knobs for simulating real workloads in production environments accurately, and none of these benchmarks inherently consider the impact of decoupling compute and storage in cloud-native database systems.

Another approach to answer these questions is to deploy and configure a cloud-native database and replay the corresponding transaction workload to collect measurements. However, this method is not always practical because the corresponding transaction workload may not be available in the production environment, and because obtaining the I/O workload of a transaction workload is expensive since it involves executing transactions exactly in the compute tier of cloud-native database systems.

*Dr. Bo Tang is the corresponding author.

Proceedings of the VLDB Endowment, Vol. 16, No. 12 ISSN 2150-8097.
doi:10.14778/3611540.3611549

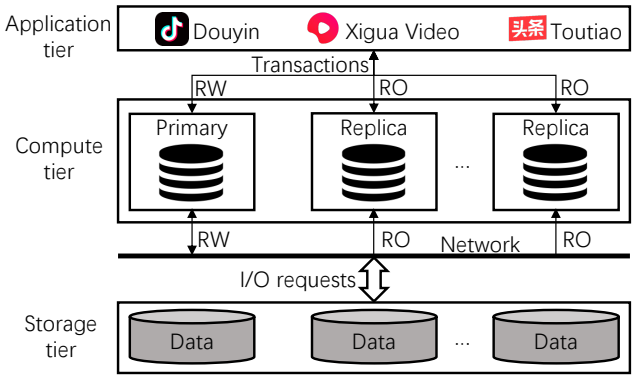


Figure 1: The architecture of ByteDance's veDB

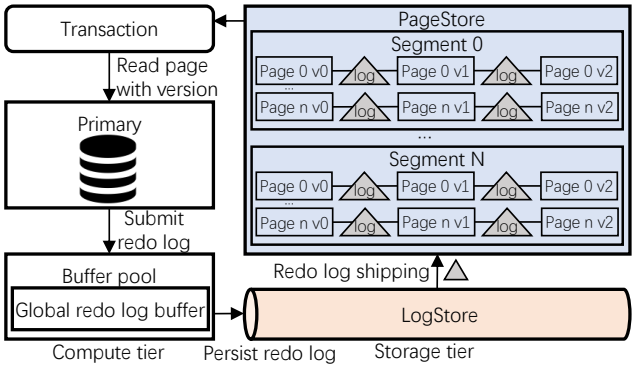


Figure 2: Transaction processing at veDB

In this work, we introduce CDSBen, a learning-based benchmark for evaluating storage service performance at ByteDance. To create this benchmark, we first analyzed two real I/O workloads, veDB_OSS and veDB_SYNC, from the storage tier of the veDB cloud-native database at ByteDance. Through this analysis, we identified key workload characteristics such as intensity, burstiness, and distribution skewness. We then created CDSBen, which utilizes two models: an IOPS sequence estimation model that uses a recurrent neural network (RNN) to estimate the IOPS sequence of the I/O workload, and a joint distribution estimation model that utilizes a Random Forest Regressor to estimate the joint distribution of I/O requests' segment address and buffer size. CDSBen then adapts the YCSB benchmark to generate I/O requests and execute them on the storage tier of veDB.

In summary, our CDSBen enjoys four nice properties:

- **High accuracy:** it generates realistic I/O workloads for storage tier performance evaluation in cloud-native database systems as it explicitly embeds the characteristics of transaction workloads via the RNN model and Random Forest Regressor. Moreover, the complex execution procedure in the compute tier also has been considered by the learning-based models implicitly.
- **Good flexibility:** it can be used to generate a wide range of realistic workloads, e.g., different read-write ratios and different TPS. In addition, CDSBen adapts YCSB to generate I/O requests, which does not rely on any physical data layout.
- **Ease of use:** CDSBen avoids lots of cumbersome engineering efforts (e.g., exact transaction execution in the compute tier) and makes the performance evaluation easy to compare by only requiring users to configure few parameters (i.e., overall TPS and the ratio of each transaction type) in it.
- **Great extensibility:** the design of CDSBen does not rely on any specific design of veDB and can be adapted to other cloud-native database systems (e.g., Amazon Aurora, Alibaba PolarDB) easily by training the models with the log statistics from the corresponding cloud-native database.

We believe that CDSBen will be widely used by the engineers, developers, and users of cloud-native database systems (i) as an evaluation and verification tool for the storage services development; (ii) to identify the proper storage engine to improve the overall query processing efficiency; and (iii) to tune the configurations in the storage tier (e.g., the number of storage nodes, the size of buffer pool) for better performance.

The remainder of this paper is organized as follows. We introduce the background of cloud-native database systems and present the characteristics of two real I/O workloads at ByteDance in Section 2. We discuss the limitations of existing benchmarks in Section 3. We propose a learning-based benchmark CDSBen in Section 4 and conduct extensive experiments to demonstrate its superiority in Section 5. We conclude this paper in Section 6.

2 PRELIMINARIES

In this section, we first introduce the architecture of cloud-native database systems by taking ByteDance's cloud-native database veDB as an example. Then, we abstract I/O requests and analyze two representative workloads at veDB to illustrate the characteristics of real I/O workloads in ByteDance's production environment.

2.1 Cloud-native database: veDB

Figure 1 illustrates the architecture of a typical cloud-native database, i.e., veDB at ByteDance. It consists of three tiers. The top application tier includes various applications, e.g., Douyin, Xigua video, and Toutiao News at ByteDance, which issue millions of transactions per second and pass them to compute tier in the middle. The compute tier includes a primary database kernel for read/write (RW) transactions and several replica database kernels for read-only (RO) transactions. The query capacity of cloud-native database systems could be dynamically adjusted by changing the number of read-only nodes. Each data record in the bottom storage tier can be accessed by the database kernels in the middle compute tier via I/O requests.

Figure 2 illustrates the query processing at veDB. The transactions from the top application tier are processed by the primary database kernel in the middle compute tier. Each transaction derives a set of redo logs that record the page modifications. The redo logs are first received by the global redo log buffer and then are persisted at the append-only LogStore. After that, the redo logs are shipped to PageStore for page modifications. PageStore manages the multiple pages as segments and provides segment-based interfaces for page reading and writing. The segment is the basic

unit in the storage tier of veDB. It is a widely used concept in other cloud-native database systems (e.g., Aurora, PolarDB). Specifically, each segment in PageStore of veDB has several GBs and contains many chain maps. Each chain map stores multiple page versions and the logs among them, as shown in Figure 2.

It is challenging to benchmark the performance of storage services in the cloud-native database veDB. On the one hand, a monolithic database does not decouple compute and storage tiers. Thus, existing end-to-end benchmarks (e.g., TPC-C) are not suitable for cloud-native database systems as they do not inherently consider the decoupled storage tier in cloud-native database systems. On the other hand, the storage tier I/O requests in cloud-native database systems are significantly different from these I/O requests in typical KV storage engines or traditional file systems as they are generated by the complex database kernel in the compute tier.

2.2 Segment-based I/O request abstraction

As shown in Figure 1, the I/O requests at storage tier are derived by the compute tier in cloud-native database systems. To design tools or benchmarks that accurately estimate the performance of the storage tier, it is essential to abstract the I/O requests properly.

Without loss of generality, the I/O requests on the storage tier of veDB at ByteDance include two fundamental operations read and write. There are two options to abstract the read and write operations: (i) fine granularity, which considers the read and write operations on the physical hard disk unit (a.k.a. page); (ii) coarse granularity, which considers the read and write operations on the basic unit of storage tier (i.e., segment in veDB's PageStore) in cloud-native database systems.

In this work, we abstract the read and write operations at the storage tier with coarse granularity, i.e., the format of data read and write the on storage tier can be represented by read(timestamp, offset in segment X, fixed buffer size) and write(timestamp, offset in segment X, variable buffer size). The key reason to use segment-based I/O abstraction is 'the log is the database' in cloud-native database systems. Specifically, the storage tier of cloud-native database systems should be capable of log replaying and multi-version page management, e.g., generating an intermediate version of a page by applying redo log. Segment abstracts read/write operations and ignores the details of storage layout and architectures, which are likely to be different among different systems. In other words, the segment-based I/O abstraction is applicable to any cloud-native database, which follows 'the log is the database' principle.

2.3 Two real I/O workloads in veDB

With segment-based I/O abstraction, it is essential to study the characteristics of real I/O workloads in production to design a benchmark that accurately evaluates the storage tier performance in cloud-native database systems. After investigating hundreds of real I/O workloads in veDB, which are from various applications at ByteDance, we have the following observations: (i) all are OLTP workloads with short transactions; (ii) the burstiness of different workloads are varying, e.g., some of them being highly bursty, while others being relatively steady; and (iii) the skewness of all workloads are obvious, e.g., the segment distribution of read and write operations.

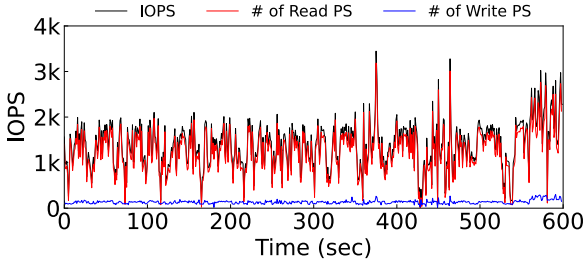


Figure 3: IOPS sequence of veDB_OSS with TPS 1,040

In subsequent, we conduct a detailed analysis of two representative real I/O workloads veDB_OSS and veDB_SYNC in the production environment of ByteDance's cloud-native database veDB. The analysis covers intensity, burstiness, and skewness (i.e., read-write ratio, write buffer size distribution, and segment ID distribution). For intensity and burstiness, we analyze the IOPS sequence of each I/O workload. For skewness, we unify read-write ratios, write buffer sizes, and segment IDs into a joint distribution matrix and conduct analysis on it. As will be demonstrate shortly, veDB_OSS is bursty and read-intensive, while veDB_SYNC is relatively steady and write-intensive.

Read-intensive I/O workload veDB_OSS**.** It is a distributed object storage system at ByteDance, which provides storage services for non-structured data. For example, it updates the metadata of the objects in the database when users post videos in Douyin. When users click and view videos published by others, veDB_OSS sends select queries to the database to get their metadata. Figure 3 illustrates a 600-second I/O workload of veDB_OSS in ByteDance's cloud-native database veDB with TPS 1040. The IOPS of veDB_OSS ranges from 39 to 3,447, and it has an average value of 1,382 with a standard deviation of 505. 90.4% of operations are read operations. Therefore, the I/O workload of veDB_OSS is read-intensive, and its burstiness is high.

We investigate other characteristics of veDB_OSS in Figure 4. We first plot the total number of read operations in each segment among these 600 seconds in Figure 4(a). Obviously, the distribution of read operations in veDB_OSS is highly skewed, i.e., 94.0% of read operations are on segments 7 and 8. In comparison, the distribution of write operations in Figure 4(b) is also skewed, but the pattern is different. Figure 4(c) shows the distribution of write operations w.r.t. their write buffer sizes. As the write buffer size grows, the number of operations falls. The buffer sizes of most write operations are quite small (e.g., less than 4KB). In Figure 4(d), we visualize the joint distribution of the segment ID and buffer size via heatmap. The deeper the color is in the cell, the more operations there are. We can clearly observe that the distributions of operations w.r.t. segment ID and buffer size are not independent.

Takeaway: *The real I/O workloads on the storage tier of cloud-native database systems have high bursty, and the corresponding segment ID and buffer size joint distribution are skewed, which is complex and cannot be easily described by a few intuitive parameters.*

Write-intensive workload veDB_SYNC**.** Many hardware devices have been sold by ByteDance and are used by billions of users. veDB_SYNC is designed to synchronize data between servers and

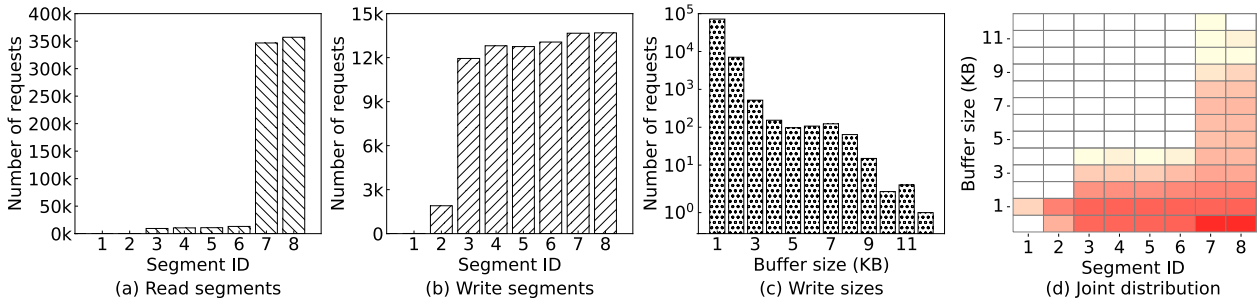


Figure 4: The characteristics of real I/O workload of veDB_OSS with TPS 1,040

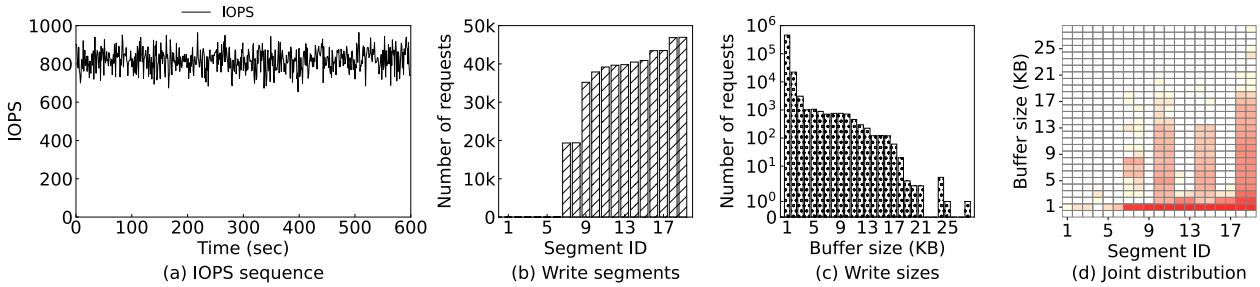


Figure 5: The characteristics of real I/O workload of veDB_SYNC with TPS 13,108

Table 1: Comparison of two real workloads at veDB

Characteristics	veDB_OSS	veDB_SYNC
IOPS intensity	39 to 3,500	600 to 1,000
Read/write ratio	9:1	0:1
Burstiness	High	Low
Hot/cold date ratio	1:3	11:8
Buffer size	0 to 12 KB	1 to 28 KB

these devices. As veDB_SYNC pushes data to devices, the status of devices will be updated in the database. Figure 5 shows a 600-second IOPS of veDB_SYNC with TPS 13,108 at ByteDance. Almost all of the operations of veDB_SYNC are write operations as the devices update their status in the database. Thus, we omit the curves of read and write operations per second in Figure 5(a). The number of write operations per second ranges from 654 to 964, the average is 820, and the standard deviation is 56. Obviously, the veDB_SYNC workload is write-intensive and its burstiness is relatively low. Figure 5(b) shows the distribution of operations on segment ID, which is skewed and is obviously different from the I/O workload of veDB_OSS, e.g., 19 different segments are accessed in veDB_SYNC, instead of 8 segments in veDB_OSS. Figure 5(c) shows the distribution of operations on buffer size. Similar to veDB_OSS, the number of operations decreases with the rising of the buffer size. However, both the maximum buffer size and the average buffer size are larger than those of veDB_OSS. Last, we present the joint distribution of the segment ID and the buffer size among all write operations of veDB_SYNC in Figure 5(d). Interestingly, the write operations with large buffer sizes (e.g., ≥ 4KB) only appear in several segments, i.e., segments 11, 14, and 15. Moreover, the joint distribution heatmap of veDB_SYNC in Figure 4(d) is visually different from veDB_OSS's in Figure 5(d).

Table 1 summarizes the characteristics of two representative real I/O workloads in ByteDance's veDB.

Takeaway: *While the real I/O workloads on the storage tier are bursty and skewed in general, the degree of these characteristics (e.g., IOPS intensity, I/O requests burstiness, distribution of target address, buffer size, and the joint distribution) in various I/O workloads can be significantly different.*

3 EXISTING SOLUTIONS

We analyze the limitation of existing solutions and discuss the most relevant studies in this section.

3.1 Existing benchmarks analysis

Many (if not all) existing benchmarks for database system performance evaluation are in two categories, micro-benchmarks and macro-benchmarks [34]. We elaborate on how to adapt them to evaluate the performance of the storage tier in cloud-native database systems as follows.

Micro-benchmarks evaluate the performance of a specific component in the database system. YCSB is a representative of micro-benchmarks for storage tier performance evaluation [23, 25, 33]. It is straightforward to adapt YCSB to measure the performance of the storage tier in cloud-native database systems as the YCSB-generated I/O trace can be applied to the storage tier easily. Macro-benchmarks evaluate the end-to-end query performance of the tested database system. TPC-C is a macro-benchmark widely used for OLTP database system performance evaluation [8, 9, 13, 35]. To evaluate the performance of the storage tier, we first generate

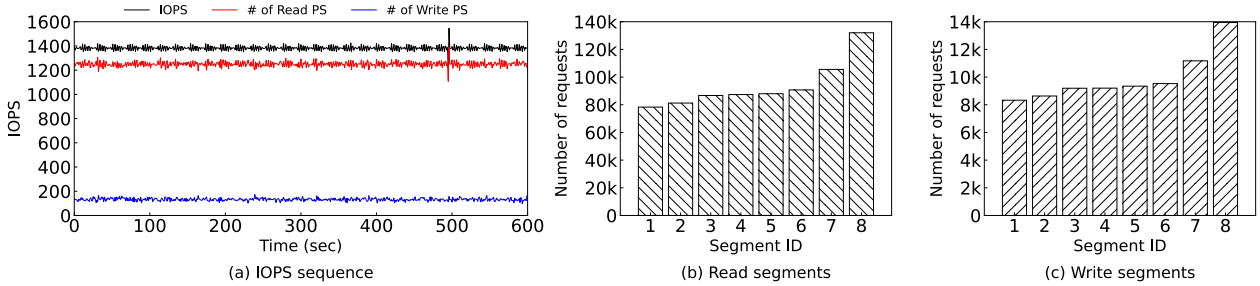


Figure 6: The characteristics of YCSB generated I/O workload with IOPS 1,382

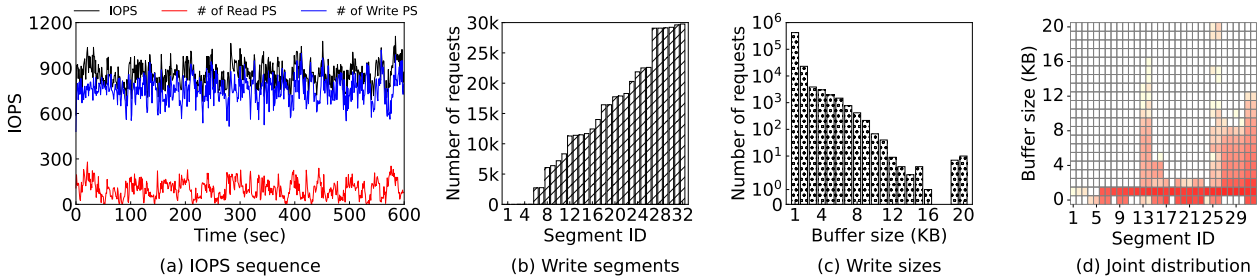


Figure 7: The characteristics of I/O workload of TPC-C generated transaction workload with TPS 80, IOPS 860

TPC-C transaction workloads, then execute them on the compute tier to obtain the corresponding I/O workloads, and last measure the performance of the storage tier by the obtained I/O workloads.

With the above adaption, both micro-benchmark (YCSB) and macro-benchmark (TPC-C) can be used to evaluate the storage tier performance in cloud-native database systems. In subsequent, we elaborate on the limitations of both micro- and macro-benchmarks by analyzing how far the benchmarks' generated I/O workloads are from two real I/O workloads veDB_OSS and veDB_SYNC in production at ByteDance. Specifically, we configure the knobs in micro-benchmark YCSB and macro-benchmark TPC-C to simulate veDB_OSS and veDB_SYNC at ByteDance, respectively[1].

YCSB knobs configuration. YCSB generates various I/O traces (each including a sequence of read and write operations) on the storage tier by offering several knobs, e.g., IOPS, read-write ratio, and target address distribution [11, 25, 33]. We tried our best to tune these knobs to make the YCSB-generated I/O workload as close as possible to the real I/O workload veDB_OSS at ByteDance. Specifically, we set the IOPS and the ratio of read operations of YCSB generated I/O workload as 1,382 and 90.4%, which are the same as the average IOPS and the read ratio in veDB_OSS. Since YCSB does not offer knobs to configure the write buffer size distribution, we set the ratio of write operations with buffer size 1KB and 16KB as 9.5% and 0.1%, respectively, such that the average buffer sizes of YCSB and veDB_OSS are the same. We use Zipfian distribution in YCSB to simulate the accessed segment ID distribution in veDB_OSS as YCSB does not support exact segment ID distribution configuration.

TPC-C knobs configuration. TPC-C generates five types of transactions on the computer tier. Users can tune the ratio of each type of transaction and the overall TPS. First, the real I/O workload veDB_SYNC is write-intensive as in Section 2.3. To obtain the I/O workload generated by TPC-C which most resembles veDB_SYNC, we maximize the ratio of the most write-intensive transaction *New-Order* in TPC-C, which is 45%. The other four types, *Payment*, *Order-Status*, *Delivery*, and *Stock-Level* are set to 43%, 4%, 4%, and 4%, which are the minimum ratio required by TPC-C [12]. Second, since TPC-C generates transaction workloads instead of I/O workloads, we cannot set its IOPS directly. We manually tune the TPS (as 80) so that the average IOPS (i.e., 860) of its corresponding I/O workload (after compute tier execution) is close enough to the average IOPS (i.e., 820) of the real workload veDB_SYNC at ByteDance.

We next analyze the limitations of these existing benchmarks by analyzing the characteristics of their generated I/O workloads in Figures 6 and 7, w.r.t. the real I/O workloads in Figures 3, 4, and 5.

Q1 Accuracy: how close are the benchmark generated I/O workloads to real ones? Even though we tried our best to tune the knobs in YCSB to generate an I/O workload that is close to the real I/O workload veDB_OSS, the characteristics of generated I/O workload (see Figure 6) are quite different from those of veDB_OSS (see Figure 4). For example, the standard deviation of YCSB-generated IOPS is only 19.4, which is 505 in veDB_OSS. The IOPS sequences are also obviously different when we compare Figure 3 with Figure 6(a). The same conclusions hold when we compare the TPC-C generated I/O workload in Figure 7 with veDB_SYNC at ByteDance in Figure 5. For example, the I/O workload generated by TPC-C in Figure 7(a) contains approximately 10% of read operations, but veDB_SYNC is write-only, as shown in Figure 5(a).

[1]We omit the discussions of using TPC-C to simulate veDB_OSS and using YCSB to simulate veDB_SYNC as they share the same conclusions.

Conclusion: *Both micro-benchmark YCSB and micro-benchmark TPC-C fail to generate I/O workloads which accurately resemble the real I/O workloads in the production environment of ByteDance.*

Q2 Flexibility: how flexible are the micro- and macro- benchmarks in the production environment? The flexibility of a benchmark is its ability to generate I/O workloads with various characteristics, e.g., intensity, burstiness, and skewness. YCSB provides several knobs for users to directly tune the intensity and read-write ratio. However, for the distribution of segment ID, users can only choose from fixed options like Zipfian distribution and uniform distribution, instead of specifying any real and complex distributions at will. Thus, the segment ID distributions of read and write operations in YCSB-generated I/O workloads in Figures 6(b) and (c) differ from those of veDB_OSS in Figures 4(a) and (b) significantly. Moreover, YCSB does not have knobs to tune the joint distribution of segment ID and buffer size, which is essential for the real I/O workload of cloud-native database systems in production. In TPC-C, users are only able to tune the TPS and the ratio of five types of transactions, which seriously constrains its flexibility. For example, it cannot generate a write-only I/O workload as veDB_SYNC at ByteDance.

Conclusion: *While micro-benchmark YCSB and macro-benchmark TPC-C provide knobs for workload configurations, the flexibility of the generated I/O workloads is very limited for the purpose of storage tier performance evaluation in cloud-native database systems.*

Q3 Usability: are the micro- and macro- benchmarks easy to use? A benchmark with high usability in cloud-native database systems should generate I/O workloads directly on the storage tier with simple user configurations. While users of TPC-C are only required to configure the TPS and the ratio of transaction types, TPC-C issues transactions on the compute tier and does not generate I/O workloads directly in cloud-native database systems. To use TPC-C to benchmark the storage tier, deploying the compute tier is necessary, which introduces auxiliary engineering effort and expensive extra costs. YCSB does not require deploying the compute tier, but users have to manually configure the intensity and the skewness of the generated I/O workload in YCSB, which hurts its usability as these configurations require the experience and effort of users.

Conclusion: *For the storage tier performance evaluation in cloud-native database systems, the micro-benchmark YCSB has better usability than the macro-benchmark TPC-C, but the usability of YCSB should be improved significantly as it requires tedious effort to set the detail configurations properly.*

3.2 Trace replayers analysis

Trace replayers are widely used for storage systems performance evaluation. They replay traces of previously executed I/O workloads [3, 24, 34]. They are easy to use because they directly use the prepared I/O traces. The major disadvantage is their reliance on traces, which severely limits their flexibility because I/O traces are not always available. Specifically, for what-if performance evaluation scenarios, such as when users want to evaluate the performance of the cloud-native database systems with a doubled TPS of a deployed application, traces are not available because such workload never occurred in the production environment before.

Table 2: Summary of solutions

Solution	Accuracy	Flexibility	Usability
YCSB	low	medium	medium
TPC-C	medium	medium	low
Trace replayer	high	low	high
Our CDSBen	**high**	**high**	**high**

We summarize the existing studies in Table 2. To sum up, none of the existing work achieves high accuracy, good flexibility, and easy-to-use simultaneously in benchmarking the storage tier of ByteDance's cloud-native database veDB.

3.3 Other relevant studies

Cao et al. characterized three real key-value workloads on RocksDB at Facebook. They then proposed a key-range-based model to better synthesize key/value workloads on RocksDB [10]. Asyabi et al. proposed a benchmark for state storage of stream processing systems in [6]. Many studies focused on the performance evaluation of storage services such as cloud file systems [1, 4, 27–29]. The common limitation of all the works above is that these benchmarks or workload generators still rely on the user to directly configure the workload to generate. Therefore, although they characterized I/O workloads from different aspects, the complexity of determining the exact I/O workload to generate is not reduced and still falls on the user, which makes them in the middle between micro-benchmarks like YCSB and macro-benchmarks like TPC-C, but not significant advancements. ₀ In comparison, as will be presented shortly, CDSBen leveraged models to reduce such complexity and achieve high accuracy, good flexibility, and ease of use at the same time. In addition, some of the relevant studies characterized the inter-arrival time of IO requests for burstiness [1, 4, 27–29]. We do not do so because the inter-arrival time distribution in our experiments spans five orders of magnitude, and small errors in the estimated frequency of high inter-arrival time lead to large errors in intensity.

4 LEARNING-BASED SOLUTION: CDSBEN

Figure 8 depicts the architecture of our proposed learning-based benchmark CDSBen. It consists of two key models: (i) the IOPS sequence estimation model, which estimates the IOPS sequence of an I/O workload; and (ii) the joint distribution estimation model, which estimates the joint distribution of the I/O requests' segment ID and buffer size. The working procedure of CDSBen is as follows. These two models in CDSBen are trained with the features which are extracted from the logs of real workloads at first. We will present the details of model training shortly. With the trained models, CDSBen then takes the features of the (expected) transaction workloads as input and estimates the IOPS sequence and the joint distribution of the desired I/O workload, respectively. CDSBen next adapts YCSB to generate the exact I/O requests of the workload accordingly. Last, CDSBen executes the generated I/O workload and evaluates the performance of the storage tier in cloud-native database systems. We next present the key techniques of CDSBen.

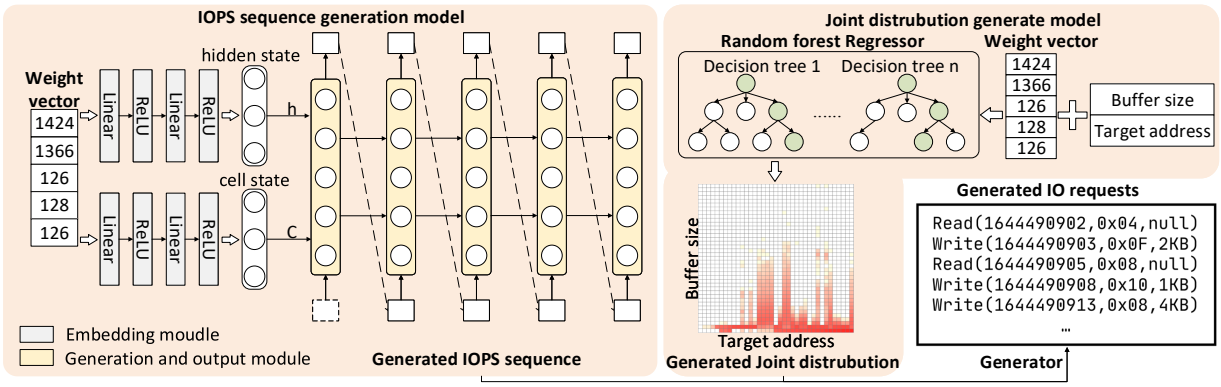


Figure 8: The architecture of our proposed learning-based benchmark CDSBen

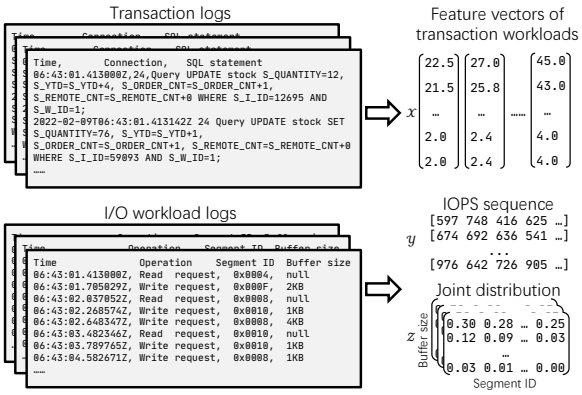


Figure 9: The example of feature embedding

4.1 Feature embedding

The logs of transaction workloads and the corresponding I/O workloads are embedded into feature vectors. Both the IOPS sequence estimation model and the joint distribution estimation model are trained from the embedded feature vectors. In particular, the transaction workloads are a mixture of several types of transactions. For example, the veDB_OSS workload includes four types of transactions, SELECT, INSERT, UPDATE, and DELETE. We parse the compute tier log and count the TPS of each transaction type as the features of the transaction workloads. As shown in Figure 9, the left transaction logs are embedded into the right feature vectors $\mathcal{X}$.

For the IOPS sequence estimation model training, we also implement a parser of the real I/O workload logs and extract feature vectors from them, which are used for the IOPS sequence estimation model. As illustrated in Figure 9, the IOPS sequence feature vector $\mathcal{Y}$ is encoded by counting the IOPS in the log. For example, the IOPS of the first four seconds are 597, 748, 416, and 625, which embeds the intensity and burstiness of the real I/O workload implicitly.

For the joint distribution estimation model training, we embed accessed segment IDs and buffer sizes of all the I/O requests into a joint distribution matrix, as $\mathcal{Z}$ shows in Figure 9. Moreover, we associate different weights to different buffer sizes as the the performance of the storage tier to process the write requests with large buffer sizes is obvious different from those with small buffer sizes. Hence, each element $\mathcal{Z}[i][j]$ shows the percentage of I/O requests in segment ID i with buffer size j-KB after normalization.

Discussion. The advantages of the feature embedding methods above are two-folds: (i) it is simple and generic as it extracts feature vectors from the workload logs; and (ii) it builds upon the segment-based I/O abstraction (see Section 2.2) and does not rely on any specific designs of ByteDance's veDB. Hence, it can be extended to other cloud-native database systems easily.

4.2 IOPS sequence estimation model

In CDSBen, the ideal model to estimate the IOPS sequence of an I/O workload should (i) capture the long-distance dependence relationship well, because the I/O workload is always not short, the IOPS sequence of an I/O workload is a time series, and the adjacent number of I/O requests per second is highly dependent; (ii) achieve good performance even when the training data is small, as the training data may not always be as large as expected in production.

Recurrent neural network (RNN) [2] is characterized by maintaining state continuity and capturing the dependency in sequence context. Moreover, its structure inherently supports variable output length, which allows us to freely output the required length of the IOPS sequence without changing the network structure. Advanced RNN models, e.g., long short-term memory (LSTM), address the limitation of the basic RNN model that it forgets information of long-distance sequence. Thus, we employ LSTM in our IOPS sequence estimation model. We omit the discussion of gated recurrent unit (GRU), another advanced RNN model as LSTM outperforms GRU in our internal testing experiments.

Model design. As shown in Figure 8, the IOPS sequence estimation model consists of three modules: *embed module*, *LSTM module*, and *output module*. The embed module is used to map the feature vector of the transaction workload to dense embedding cell state c and hidden state h. We employ two two-layer fully connected neural networks with *ReLU* activation function in the embed module. The LSTM module takes cell state c, hidden state h and $\mathcal{Y}_{t-1}$ (the IOPS value at timestamp $t-1$) as input to estimate $\mathcal{Y}_t$, the IOPS value at timestamp t by Equation (1), where W_f, W_i, W_o and W_c are parameter matrices of input connections, U_f, U_i, U_o and U_c are

parameter matrices of recurrent connections. b_f, b_i and b_o are bias vectors, $\circ$ denotes element-wise multiplication, σ_g is the activate function *sigmoid*, σ_c and σ_h is the hyperbolic tangent function. f_t, i_t, o_t and $\tilde{c}_t$ are the activation vector of forget gate, input gate, output gate, and cell input respectively. c_t and h_t are the cell state vector and hidden state vector, where the initial c_0 and h_0 are computed by embed module. h_t is also the output $\mathcal{Y}_t$ of the LSTM module. The output module performs a linear transformation on the output of the LSTM module to adjust its dimension to the output length of the required IOPS sequence. We use a one-layer fully connected neural network as our output layer.

$$\begin{aligned} i_t &= \sigma_g(W_i \mathcal{Y}_{t-1} + U_i h_{t-1} + b_i) \\ o_t &= \sigma_g(W_o \mathcal{Y}_{t-1} + U_o h_{t-1} + b_o) \\ \tilde{c}_t &= \sigma_c(W_c \mathcal{Y}_{t-1} + U_c h_{t-1} + b_c) \\ c_t &= f_t \circ c_{t-1} + i_t \circ \tilde{c}_t \\ h_t &= o_t \circ \sigma_h(c_t) \end{aligned} \tag{1}$$

Learning models always trade-off between the model accuracy and training/inference cost. To provide a highly accurate IOPS sequence estimation model with acceptable training and inferring cost in CDSBen, as shown in Figure 8, we stack five 1024-unit LSTM layers together. The first layer takes the cell state c and hidden state h (from embed module) and the IOPS $\mathcal{Y}_0$ of timestamp 0 as input and its output is a 1024-sized vector. The next four layers in turn receive the output vector of the previous LSTM layer as input and return a 1024-sized vector accordingly. In order to enhance the stability of the output of the model and to further reduce its accumulated error during the generation of the IOPS sequence, we use kernel regularizers to punish the weight of the last LSTM layer. We avoid gradient disappearance or gradient explosion issues in deep neural network (as we concatenated five LSTM layers) by utilizing the structure of ResNet [17]. In particular, we connect the input and output of LSTM layers by skip-connection and use $f(\mathcal{Y}_{t-1}, c, h) = \mathcal{Y}_t - \mathcal{Y}_{t-1}$ in LSTM module, instead of the original $f(\mathcal{Y}_{t-1}, c, h) = \mathcal{Y}_t$.

Model training. The IOPS sequence estimation model in CDSBen is an end-to-end model and we use backpropagation to optimize it. The loss function is the mean absolute error (MAE) $\sum_{i=0}^{n} |\hat{\mathcal{Y}}_i - \mathcal{Y}_i|$, where $\mathcal{Y}$ is the real IOPS sequence, and $\hat{\mathcal{Y}}$ is the estimated IOPS sequence. The Adam optimizer [19] is used for model training and the learning rate is 1×10^{-4}. The training process terminates when the loss of the model fluctuates slightly.

Model inference. During the inference phase, CDSBen first takes the TPS of each type of transaction and the initialized IOPS at timestamp 0 as input. Then, it estimates the rest IOPS sequence via the IOPS sequence estimation model accordingly. We employ a three-layer fully connected neural network to initialize the IOPS value at timestamp 0 (i.e., $\mathcal{Y}_0$).

4.3 Joint distribution estimation model

In this section, we design the joint distribution estimation model to estimate the joint distribution of segment ID and buffer size of each read or write operation in the workload.

Model design. In the literature, there are two categories of models that can be adapted to estimate the joint distribution matrix in our case. The first category estimates the entire joint distribution matrix $\mathcal{Z}$ by the model. For example, it treats the joint distribution matrix as an image and uses a generative adversarial net (GAN) [15] or variational auto-encode (VAE) [20] to generate it. However, the major limitation of the models in this category is that they do not have the ability to distinguish the hot/cold segments via the generated joint distribution matrix as the segment ID is meaningless in these models. The second category estimates each element in the joint distribution matrix one by one. For example, models such as decision tree [32], support vector machine [30], and logistic regression [18] take the feature vector of transaction workload, segment ID i, and buffer size j-KB tuple as input, then estimates $\mathcal{Z}[i][j]$ independently. Our solution adapts the model in the second category. Specifically, we design a model for joint distribution based on random forest regressor[26, 31], which includes a number of different and independent decision trees (i.e., there are 1,200 decision trees in our experiments) as our joint distribution estimation model. It shows the superiority in the following two aspects: (i) it estimates the joint distribution of I/O workload accurately by exploiting the property of multiple decision trees, as we will elaborate on shortly; and (ii) it enjoys high efficiency as different decision trees can run in parallel as they are mutually independent.

Model training. In the training phase of each decision tree in the random forest regressor, it randomly selects n feature vectors of transaction workloads and their corresponding joint distribution matrices to form its training dataset (a.k.a., bootstrap dataset). Thus, the bootstrap dataset includes n pairs of $((\mathcal{X}, i, j), \mathcal{Z}[i][j])$ in Figure 9, where the 3-tuple $(\mathcal{X}, i, j)$ is the input and $\mathcal{Z}[i][j]$ is the output.

Each decision tree is trained as follows [26]. We first put all input 3-tuple $(\mathcal{X}, i, j)$ into an input matrix $\mathcal{M}$ and set it as the input of the decision tree. The label is all elements $\mathcal{Z}[i][j]$ of all these n joint distribution matrices $\mathcal{Z}$. The decision tree recursively splits the feature space in $\mathcal{M}$ by grouping the feature vectors with the same value or close values in the label. Suppose the data at node a in the decision tree is a vector Q_a with N_a samples, for each candidate split $\theta = (k, t_a)$ which consists of feature k and its threshold t_a, the data is divided into two subsets $Q_a^{left}(\theta)$ and $Q_a^{right}(\theta)$ by the following rules:

$$Q_a^{left}(\theta) = \{(x, z) | x_k \le t_a\} \text{ and } Q_a^{right}(\theta) = Q_a \setminus Q_a^{left}(\theta)$$

We use mean squared error (MSE) as the loss function $\mathcal{H}$ in Equation (2), where $\bar{z}$ is the mean of all z values in N_a.

$$\mathcal{H}(Q_a) = \frac{1}{N_a} \sum_{z \in Q_a} (z - \bar{z}_m)^2 \tag{2}$$

The function G (in Equation 3) measures the quality of split θ on node a.

$$G(Q_a, \theta) = \frac{N_a^{left}}{N_a} \mathcal{H}(Q_a^{left}(\theta)) + \frac{N_a^{left}}{N_a} \mathcal{H}(Q_a^{right}(\theta)) \tag{3}$$

The training of each decision tree is to find the parameters to minimize the loss:

$$\theta^* = \text{argmin}_\theta G(Q_a, \theta) \tag{4}$$

The decision trees recursively split the two subsets $Q_a^{left}(\theta)$ and $Q_a^{right}(\theta)$ until (i) the decision tree reaches the maximum allowable depth or (ii) $N_a = 1$. In order to reduce the variance of estimation results and to improve the accuracy of the joint distribution estimation model, we employ two stochastic processes during the above model training phase. In particular, the probability of each $((\mathcal{X}, i, j), \mathcal{Z}[i][j])$ tuple to be included by a specific bootstrap dataset is $1 - (1 - \frac{1}{n})^n$. It is $1 - \frac{1}{e} = 0.632$ when n is large. In other words, almost 37% of the whole training dataset will not be used in a specific decision tree. We use them to improve the generalization ability of the trained decision tree. Second, it only uses 50% of randomly selected transaction workload features for each decision tree branching.

Model inference. During inference, we obtain the whole joint distribution matrix by estimating the value of each segment ID and buffer size, with TPS of each transaction type in the workload one by one. Specifically, for each 3-tuple of transaction workload feature, segment ID, and buffer size $(\mathcal{X}, i, j)$, we take the mean of all outputs from each trained decision tree in the Random Forest Regressor as the estimated value.

After joint distribution is estimated, it needs normalization to be used in workload generation. Since the joint distribution matrix is normalized by considering weights of different buffer sizes, we reverse this process and remove the weights. Then, we divide the joint distribution matrix by the sum of every element, so that the joint distribution matrix is the joint discrete probability distribution of segment ID and buffer size, which is used in workload generation.

4.4 YCSB-adapted I/O workload generation

With the estimated IOPS sequence and the joint distribution in an I/O workload, we adapt YCSB to generate the exact I/O requests in it. For each timestamp t in the IOPS sequence with IOPS $\mathcal{Y}_t$, we uniformly distributed these I/O requests in the timestamp t. For each request, it randomly selects segment ID and buffer size from the joint distribution. If the buffer size is 0, then this request is a read request. Otherwise, it is a write request. After the requests are generated, we modify YCSB to bypass its configurations, load the generated requests, and execute them on the storage tier of veDB. We rely on the built-in implementation of YCSB for metrics collection. After the execution is complete, the time consumed to execute each I/O request will be reported by YCSB, where we conduct analysis for performance evaluation.

4.5 Discussions of CDSBen

Usage. With the proposed CDSBen, we present how to use it to evaluate the performance of the storage tier in cloud-native database systems. First, we collect the workload logs from the compute tier and the storage tier and embed them to train the models in CDSBen. Then, the user inputs the feature vector of their desired workload to CDSBen and obtains the corresponding generated I/O workload. Last, we execute the generated I/O workload on the storage tier with adapted YCSB and measure the performance metrics accordingly.

Because of the robust design of CDSBen's architecture, it also can be used to generate the I/O workloads which have diurnal patterns or seasonality. For example, assume a transaction workload with strong seasonality, e.g., the average TPS of a transaction workload being 1,000 in the morning, increasing to 20,000 at noon, and dropping back to 500 in the evening. To generate realistic I/O workloads for it, the user inputs three feature vectors to CDSBen, whose average TPS are 1,000, 20,000, and 500. CDSBen generates the corresponding I/O workloads for each input feature vector. All generated I/O traces are concatenated together as an overall generated I/O workload for a transaction workload with seasonality.

By exploiting CDSBen, all the questions in Section 1 can be answered effectively. To exemplify, if the user wants to evaluate the performance of the storage tier when the TPS of the transaction workload doubles or the read-write ratio changes, he only needs to modify the input feature vectors of the transaction workload in CDSBen, and CDSBen will generate the corresponding I/O workload for subsequent performance evaluation on storage tier. For the techniques changed in the underlying storage tier, the users only need to execute the CDSBen generated I/O workload on the latest version, then measure the performance metrics and compare them with the previous ones.

Extensibility. It consists of three steps to use CDSBen for storage tier performance evaluation in cloud-native database systems: (i) feature embedding, (ii) model training, and (iii) I/O workload generating. For the feature embedding, as our discussion in Section 4.1, our method is generic as (i) systems need little to no modification to generate the logs needed, and (ii) CDSBen builds upon the segment-based I/O abstraction, which supports 'the log is the database' principle in all cloud-native database systems. For the model training, both models (in Sections 4.2 and 4.3) do not rely on any specific design of ByteDance's veDB. In addition, they are trained offline. Thus, they do not incur extra runtime overhead. For the I/O workload generating, we adapted the widely used YCSB to generate the estimated I/O workload, which confirms that our CDSBen is applicable to other cloud-native database systems as YCSB does not rely on any physical data layout. In conclusion, CDSBen holds great extensibility, and it can be adapted to evaluate the performance of the storage services in other cloud-native database systems, e.g., Aurora and Socrates.

Limitations. Even though CDSBen has multiple desirable qualities for performance evaluation of the storage tier of cloud-native database systems, including high accuracy, good flexibility, ease of use, and high extensibility, it also has several limitations. First, the models of CDSBen must be trained individually for each workload. This is because CDSBen is semantic oblivious. In the embedding of transaction workloads, we do not consider the definition of transactions or tables, and only consider TPS. Although due to the lightweight design the models, it only takes minutes in our experiments to train them for each workload, we are working on a semantic aware version to reduce this overhead and further improve flexibility. Second, CDSBen works the best only on directly accessible storage tiers where the workloads are derived by compute tier on the above, because essentially CDSBen leverages models to learn the behaviors of the compute tier and estimate the outputs on the storage tier. When the storage tier is not directly accessible, such as to benchmark monolithic databases like MySQL, end-to-end benchmarks like TPC-C are better options. When the compute workloads are

not accessible for model training, such as when we are testing the storage tier with workloads from an external compute tier, CDSBen's functions are the same as YCSB, because it relies on YCSB for workload execution.

5 EXPERIMENTAL EVALUATION

In this section, we first demonstrate the the high accuracy of CDSBen by comparing its generated realistic I/O workloads with the corresponding real I/O workloads (i.e., veDB_OSS and veDB_SYNC) in production at ByteDance (see Section 5.1). We then evaluate the flexibility and usability of CDSBen by measuring its ability to generate various realistic I/O workloads in Section 5.2. Last, we report the measured tail latency of CDSBen's generated workloads on veDB, which is very close to the performance of running real I/O workloads (see Section 5.3).

Experimental setting. We deploy the cloud-native database veDB in a mini-cluster with 4 nodes at ByteDance. We use one as the RW computation node in the computation tier, its CPU is two-way Intel(R) Xeon(R) Gold 6130 CPU @ 2.10GHz and the memory size is 376GB. The rest three are storage nodes in the storage tier. Each of them has a 4TB SSD.

5.1 Case study

Case study on veDB_OSS**.** We first investigate how close the CDSBen generated I/O workload is to the real veDB_OSS I/O workload. We collect the logs of veDB_OSS service in production at ByteDance. The TPS of veDB_OSS ranges from 1,000 to 10,000. We use the veDB_OSS transaction workload feature vectors, their corresponding IOPS sequences, and joint distributions whose TPS is from 1,000 to 6,5000 as training data. All veDB_OSS transaction workloads with TPS larger than 6,500 are used as testing data. We input the transaction workload feature vector into CDSBen to estimate the IOPS sequence and joint distribution and compare them with the ones we obtained by executing the transaction workloads on veDB. Figure 10 shows the comparison between the real veDB_OSS I/O workload, i.e., computing the veDB_OSS transactions (its TPS is 9,218) on compute tier and obtaining its derived I/O workload, and CDSBen generated veDB_OSS I/O workload.

IOPS sequence: As shown in Figure 10(a), the IOPS sequence of veDB_OSS estimated by CDSBen shares a similar trend (e.g., intensity and burstiness) with the real IOPS sequence of veDB_OSS. In particular, the mean and the standard deviation of the IOPS sequence in real veDB_OSS I/O workload (i.e., computed) are 8,658 and 2,120, respectively. The corresponding values of CDSBen's estimated IOPS sequence are 8,538 and 2,303. Obviously, our estimated IOPS sequence is very close to the real IOPS sequence of veDB_OSS. It confirms the high accuracy of the designed IOPS sequence estimation model of CDSBen in Section 4.2.

Joint distribution: Figure 10(b) and (c) visualize the joint distributions of the computed I/O workload and CDSBen's estimated I/O workload of veDB_OSS. Visually, the hot/cold access patterns of our estimated joint distribution in Figure 10(c) and the computed joint distribution in Figure 10(b) are similar. For example, the write requests with large buffer sizes are on segment 7 and 8 in both joint distributions. This shows that our joint distribution estimation model in Section 4.3 achieves high accuracy. Additionally, we compute the Kullback-Leibler divergence (K-L divergence), which measures the difference between two distributions in mathematical statistics. It ranges from 0 to $+\infty$ and smaller values indicate higher similarity [22]. For the distributions in Figure 10(b) and (c), this value is 0.00714, indicating an extremely small difference between the computed and estimated joint distributions of veDB_OSS. This confirms the high accuracy of the designed joint distribution estimation model of CDSBen in Section 4.3.

Table 3: Evaluation of veDB_OSS **services by varying TPS.** μ_{IOPS} **and** $\hat{\mu}_{IOPS}$ **are computed and estimated average IOPS.** σ_{IOPS} **and** $\hat{\sigma}_{IOPS}$ **are standard deviations of IOPS.**

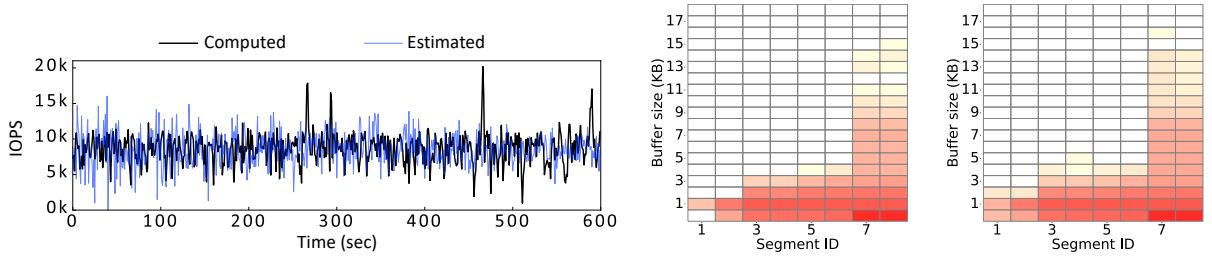
TPS	μ_{IOPS}	$\hat{\mu}_{IOPS}$	σ_{IOPS}	$\hat{\sigma}_{IOPS}$	K-L divergence
6,892	7,337	6,493	1,863	1,738	0.00711
7,700	7,792	7,245	2,029	1,946	0.00713
8,457	8,328	7,851	1,949	2,113	0.00713
9,218	8,658	8,538	2,120	2,303	0.00714
9,693	8,958	8,890	2,004	2,399	0.00719

We summarize the statistics of CDSBen estimated I/O workloads and real I/O workloads of veDB_OSS with different TPS (from 6,892 to 9,693) in Table 3. μ_{IOPS} and $\hat{\mu}_{IOPS}$ are the computed and estimated average IOPS respectively, and σ_{IOPS} and $\hat{\sigma}_{IOPS}$ are the standard deviation of the computed and estimated IOPS sequences. In this table, the small difference between the computed and estimated average IOPS and standard deviations, as well as the small K-L divergence values indicate that the models in CDSBen can estimate the I/O workloads of veDB_OSS with high accuracy even when the TPS of the transaction workload for which we estimate I/O workloads is different from the transaction workloads we use for model training.

Case study on veDB_SYNC**.** We next demonstrate the high accuracy of CDSBen by the case study on real veDB_SYNC workload. Similarly, we use the veDB_SYNC transaction workloads with low TPS (i.e., $\leq 10,000$) as training data, and estimate the corresponding I/O workloads for the veDB_SYNC workloads with high TPS (i.e., 17,354).

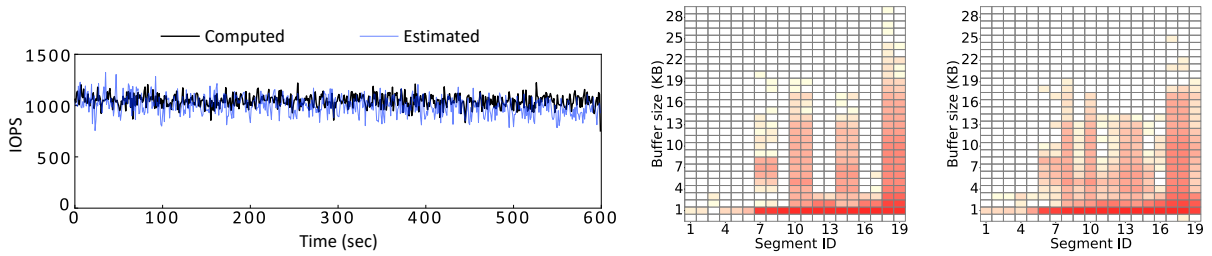
IOPS sequence: Figure 11(a) shows the CDSBen's estimated and real computed IOPS sequences of veDB_SYNC workload when the TPS is 17,354. The estimated veDB_SYNC IOPS sequence has high accuracy as (i) the estimated curve overlapped the computed curve (real IOPS sequence) significantly; and (ii) the characteristics of two IOPS sequences are very similar. For example, the mean of computed and estimated IOPS sequences are 1,046 and 999 respectively.

Joint distribution: We then investigate the accuracy of the CDSBen's estimated joint distribution. The heatmaps in Figures 11(b) and (c) show the joint distribution of real computed and estimated veDB_SYNC I/O workload. First, the K-L divergence of these two joint distributions is 0.00332, which verifies the high accuracy of the joint distribution estimation model in CDSBen from the statistical aspect. Second, due to the power of random forest regressor, our joint distribution estimation model predicts the non-uniform distribution of read/write requests accurately. For example, the distribution of write requests with large buffer sizes in the estimated



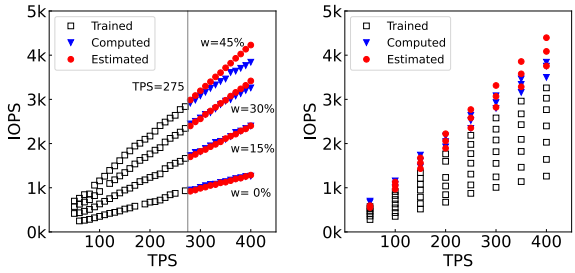
(a) IOPS sequence (b) Computed joint distribution (c) Estimated joint distribution

Figure 10: Real veDB_OSS **I/O workload (Computed) vs.** CDSBen **generated** veDB_OSS **I/O workload (Estimated)**



(a) IOPS sequence (b) Computed joint distribution (c) Estimated joint distribution

Figure 11: Real veDB_SYNC **I/O workload (Computed) vs.** CDSBen **generated** veDB_SYNC **I/O workload (Estimated)**



(a) Estimating high TPS (b) Estimating high write ratio

Figure 12: Effectiveness evaluation of CDSBen **on TPC-C**

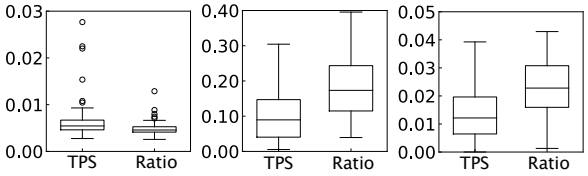
joint distribution heatmap is almost the same as those in the real computed heatmap, e.g., segments with ID 10, 14, 18 and 19.

5.2 Effectiveness evaluation

In this section, we elaborate good flexibility and ease of use properties of CDSBen. Taking the reproducibility and availability into consideration, we verify the good flexibility and ease of use of CDSBen by conducting extensive experiments on standard benchmark TPC-C. We use the implementation of TPC-C in BenchBase [14] and set its scale factor as 1,000. We compare the estimated I/O workloads with the computed I/O workloads of TPC-C transaction workloads with different TPS and different mixtures of five types of transactions. In particular, the TPS of TPC-C ranges from 50 to 400 in our experiments. We fix the ratios of transaction type *Payment*, *Delivery*, and *Stock-Level* to 43%, 4%, and 4%, same as the required minimum values in TPC-C specification respectively [12], and vary the ratios of the most write intensive transaction type *New-order* and the most read intensive transaction type *Order-Status* to generate workloads with different write-intensiveness. Specifically, the ratio of *New-Order* increases from 0% to 45%. Accordingly, the ratio of *Order-Status* drops from 49% to 4%.

Evaluation of IOPS sequence estimation model. We first use the logs from TPC-C transactions whose TPS is smaller than 275 as training data and estimate the TPC-C I/O workloads with high TPS (i.e., $\geq$ 275) to evaluate the flexibility of CDSBen's IOPS sequence estimation model. Figure 12(a) plots the average value of each estimated IOPS sequence under different TPC-C transaction workloads. Here, w means the ratio of *New-Order* transaction type. For demonstration purposes, we only plotted values with w changing from 0% to 45% with a step size of 15%. As shown in Figure 12(a), the average values of IOPS sequences in our CDSBen-estimated I/O workload (see red dots) are close to (or even identical with) the corresponding average values of the computed I/O workload (see blue triangles). The errors between these two sets of values are from -3.18% to 10.1%, which confirms that CDSBen's IOPS sequence estimation model is flexible enough to estimate I/O workloads with varying characteristics, i.e., TPS and RW ratios, while preserving high accuracy. We next use the TPC-C workloads with low write ratios (the ratio of *New-Order* smaller than 30%) to estimate its I/O workloads with high write ratios (the ratio of *New-Order* ranges from 30% to 45%) among all levels of TPS, i.e., TPS ranges from 50 to 400 with step size 50. As shown in Figure 12(b), the average absolute error is 4.46% in all cases, which also shows the good flexibility of CDSBen's model to function properly for different I/O workloads.

Evaluation of joint distribution estimation model. We next demonstrate the flexibility of CDSBen's joint distribution estimation model by comparing the joint distributions of the estimated



(a) K-L divergence (b) RW ratio (c) Avg. buffer size

Figure 13: TPC-C joint distribution evaluation

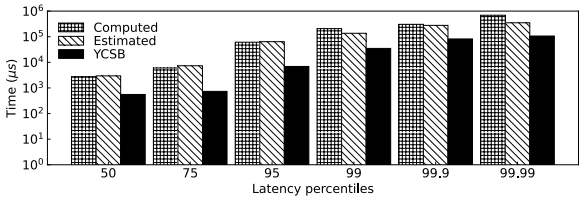


Figure 14: Tail-latency evaluation

and computed I/O workloads for various TPC-C transactions via box plots in Figure 13. In each experiment, we use the trained models in Figure 12(a) and (b) to estimate the workloads with high TPS (denoted as TPS) and high write ratio (denoted as Ratio).

K-L divergence: Figure 13(a) plots K-L divergence values of the joint distributions for every the estimated and computed TPC-C workloads. All values are very small, i.e., less than 0.03, in both estimating high TPS and high write ratio cases.

Read-write ratio: Figure 13(b) compares the read-write ratios of the estimated and computed I/O workloads for all TPC-C workloads. The average error is less than 10% and 20% for high TPS and high write-ratio workloads respectively. On the one hand, it shows the superiority of CDSBen to accurately resemble real I/O workloads. On the other hand, it confirms that it is challenge to generate I/O workloads with exact read-write ratios as the real ones.

Buffer size: Figure 13(c) investigates the average buffer sizes in the estimated and computed I/O workloads for all tested TPC-C workloads. The errors between the estimated workloads and computed workloads are smaller than 0.05. It confirms that the average buffer size in CDSBen's estimated I/O workloads is almost the same as the exact one for all tested cases.

Besides the evaluated good flexibility, the experiments in Figures 12 and 13 also indicate that CDSBen is easy to use. During the experiments, after the models are trained, the only input from the user is the feature vector of the target workload. By changing the input feature vectors for different intensity or read-write ratios, corresponding I/O workloads will be generated for performance evaluation and the questions in Section 1 are answered by executing the I/O workloads. For example, we can change the input feature vector from $(60, 116, 8, 8, 8)$ (overall TPS being 200, ratio of *New-Order* being 30%) to $(120, 232, 16, 16, 16)$ (overall TPS being 400) to generate I/O workloads for which the TPS doubles, as in Figure 12(a), or to $(90, 86, 8, 8, 8)$ (ratio of *New-Order* being 45%), as in Figure 12(b) to simulate the case where the read-write ratio changes dramatically. Instead of tuning complex knobs or deploying the compute tier, we only have to change the human-understandable input feature vector in CDSBen, which is simple-and-effective.

5.3 Evaluation of tail latency

In this experiment, we show that the tail latency of running CDSBen's estimated I/O workload is significantly closer to the tail latency of running real I/O workload on veDB compared with the tail latency of running YCSB generated I/O workload.

We first configure CDSBen and YCSB to generate I/O workloads, which simulate the real veDB_OSS workload of veDB. Specifically, we use a veDB_OSS transaction workload with TPS 9,218. The average IOPS and the standard deviation of the corresponding I/O workload is 8,658 and 2,120 respectively. For CDSBen's estimated veDB_OSS I/O workload, the estimated average IOPS is 8,538 and the standard deviation is 2,303. For YCSB-generated I/O workload, we set the average IOPS as 8,658.

We then run these three workloads (i.e., CDSBen's estimated, YCSB's generated, and the real one) on veDB and measure its tail latency. Figure 14 plots the measured tail latencies of running three I/O workloads of veDB_OSS in microseconds. Compared with the tail latency of YCSB-generated I/O workloads, the tail latency of CDSBen's estimated I/O workload is much closer to the real computed veDB_OSS I/O workload. In addition, the tail latency of YCSB generated I/O workload is always smaller than real computed veDB_OSS I/O workload as real veDB_OSS service has high burstiness in production, as shown in Figure 3.

6 CONCLUSION

In this work, we present CDSBen, a new benchmarking tool designed to evaluate the performance of storage services in cloud-native database systems. Firstly, we introduce two real-world workloads, veDB_OSS and veDB_SYNC, which are currently in production at ByteDance. We then discuss the limitations of existing micro- and macro-benchmarks, such as YCSB and TPC-C, in capturing the characteristics of actual I/O workloads. Next, we propose a novel learning-based solution, CDSBen, which consists of two key models, the IOPS sequence estimation model and the joint distribution estimation model, to overcome the limitations of current solutions. To validate the superiority of CDSBen for storage services performance evaluation in cloud-native database systems, we conduct extensive experiments using ByteDance's veDB. Our results demonstrate that CDSBen outperforms existing benchmarking tools and it provides more accurate performance predictions for storage services in cloud-native database systems. CDSBen is open sourced at Github. We plan to extend it for other cloud-native database systems in the future as it does not reply on any specific design of veDB at ByteDance.

ACKNOWLEDGMENTS

We are grateful to the anonymous reviewers for their constructive comments and insightful suggestions to improve the quality of this paper. This work was partially supported by Shenzhen Fundamental Research Program (Grant No. 20220815112848002) and the Guangdong Provincial Key Laboratory (Grant No. 2020B121201001). Dr. Bo Tang is also affiliated with the Research Institute of Trustworthy Autonomous Systems, Southern University of Science and Technology, Shenzhen, China.

REFERENCES

[1] Cristina L. Abad, Mindi Yuan, Chris X. Cai, Yi Lu, Nathan Roberts, and Roy H. Campbell. 2013. Generating Request Streams on Big Data Using Clustered Renewal Processes. *Performance Evaluation* 70, 10 (2013), 704–719.

[2] Oludare Isaac Abiodun, Aman Jantan, Abiodun Esther Omolara, Kemi Victoria Dada, Nachaat AbdElatif Mohamed, and Humaira Arshad. 2018. State-of-the-art in Artificial Neural Network Applications: A Survey. *Heliyon* 4, 11 (2018), e00938.

[3] Ibrahim Umit Akgun, Geoff Kuenning, and Erez Zadok. 2020. Re-Animator: Versatile High-Fidelity Storage-System Tracing and Replaying. In *SYSTOR*. 61–74.

[4] Ahmad Al-Shishtawy and Vladimir Vlassov. 2013. ElastMan: Elasticity Manager for Elastic Key-Value Stores in the Cloud. In *CAC*. 1–10.

[5] Panagiotis Antonopoulos, Alex Budovski, Cristian Diaconu, Alejandro Hernandez Saenz, Jack Hu, Hanuma Kodavalla, Donald Kossmann, Sandeep Lingam, Umar Farooq Minhas, Naveen Prakash, Vijendra Purohit, Hugh Qu, Chaitanya Sreenivas Ravella, Krystyna Reisteter, Sheetal Shrotri, Dixin Tang, and Vikram Wakade. 2019. Socrates: The New SQL Server in the Cloud. In *SIGMOD*. 1743–1756.

[6] Esmail Asyabi, Yuanli Wang, John Liagouris, Vasiliki Kalavri, and Azer Bestavros. 2022. A New Benchmark Harness for Systematic and Robust Evaluation of Streaming State Stores. In *EuroSys*. 559–574.

[7] Dina Bitton, David J. DeWitt, and Carolyn Turbyfill. 1983. Benchmarking Database Systems A Systematic Approach. In *VLDB*. 8–19.

[8] Wei Cao, Zhenjun Liu, Peng Wang, Sen Chen, Caifeng Zhu, Song Zheng, Yuhui Wang, and Guoqing Ma. 2018. PolarFS: An Ultra-Low Latency and Failure Resilient Distributed File System for Shared Storage Cloud Database. *PVLDB* 11, 12 (2018), 1849–1862.

[9] Wei Cao, Yingqiang Zhang, Xinjun Yang, Feifei Li, Sheng Wang, Qingda Hu, Xuntao Cheng, Zongzhi Chen, Zhenjun Liu, Jing Fang, Bo Wang, Yuhui Wang, Haiqing Sun, Ze Yang, Zhushi Cheng, Sen Chen, Jian Wu, Wei Hu, Jianwei Zhao, Yusong Gao, Songlu Cai, Yunyang Zhang, and Jiawang Tong. 2021. PolarDB Serverless: A Cloud Native Database for Disaggregated Data Centers. In *SIGMOD*. 2477–2489.

[10] Zhichao Cao, Siying Dong, Sagar Vemuri, and David H. C. Du. 2020. Characterizing, Modeling, and Benchmarking RocksDB Key-Value Workloads at Facebook. In *FAST*. 209–223.

[11] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. 2010. Benchmarking Cloud Serving Systems with YCSB. In *SoCC*. 143–154.

[12] Transaction Processing Performance Council. 2010. *TPC Benchamrk C Standard Specification*. Retrieved July 3, 2023 from https://www.tpc.org/tpc_documents_current_versions/pdf/tpc-c_v5.11.0.pdf

[13] Alex Depoutovitch, Chong Chen, Jin Chen, Paul Larson, Shu Lin, Jack Ng, Wenlin Cui, Qiang Liu, Wei Huang, Yong Xiao, and Yongjun He. 2020. Taurus Database: How to Be Fast, Available, and Frugal in the Cloud. In *SIGMOD*. 1463–1478.

[14] Djellel E. Difallah, Andrew Pavlo, Carlo Curino, and Philippe Cudre-Mauroux. 2013. OLTP-Bench: An Extensible Testbed for Benchmarking Relational Databases. *PVLDB* 7, 4 (2013), 277–288.

[15] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative Adversarial Nets. In *NeurIPS*. 2672–2680.

[16] Jim Gray. 1992. *Benchmark Handbook: For Database and Transaction Processing Systems*. Morgan Kaufmann Publishers Inc.

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *CVPR*. 770–778.

[18] David W. Hosmer Jr, Stanley Lemeshow, and Rodney X. Sturdivant. 2013. *Applied Logistic Regression*. John Wiley & Sons.

[19] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *arXiv preprint arXiv:1412.6980* (2014).

[20] Diederik P. Kingma and Max Welling. 2013. Auto-Encoding Variational Bayes. *arXiv preprint arXiv:1312.6114* (2013).

[21] Scott T. Leutenegger and Daniel Dias. 1993. A Modeling Study of the TPC-C Benchmark. In *SIGMOD*. 22–31.

[22] Miodrag Lovric et al. 2011. *International Encyclopedia of Statistical Science*. Springer Berlin Heidelberg.

[23] Lanyue Lu, Thanumalayan S. Pillai, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. 2016. WiscKey: Separating Keys from Values in SSD-conscious Storage. In *FAST*. 133–148.

[24] Michael P. Mesnier. 2007. //TRACE: Parallel Trace Replay with Approximate Causal Events. In *FAST*. 153–167.

[25] Zhu Pang, Qingda Lu, Shuo Chen, Rui Wang, Yikang Xu, and Jiesheng Wu. 2021. ArkDB: A Key-Value Engine for Scalable Cloud Storage Services. In *SIGMOD*. 2570–2583.

[26] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *JMLR* 12 (2011), 2825–2830.

[27] Rekha Pitchumani, Shayna Frank, and Ethan L. Miller. 2015. Realistic Request Arrival Generation in Storage Benchmarks. In *MSST*. 1–10.

[28] Zujie Ren, Weisong Shi, Jian Wan, Feng Cao, and Jiangbin Lin. 2017. Realistic and Scalable Benchmarking Cloud File Systems: Practices and Lessons from AliCloud. *TPDS* 28, 11 (2017), 3272–3285.

[29] Zujie Ren, Biao Xu, Weisong Shi, Yongjian Ren, Feng Cao, Jiangbin Lin, and Zheng Ye. 2016. iGen: A Realistic Request Generator for Cloud File Systems Benchmarking. In *CLOUD*. 343–350.

[30] Johan A. K. Suykens and Joos Vandewalle. 1999. Least Squares Support Vector Machine Classifiers. *Neural Processing Letters* 9, 3 (1999), 293–300.

[31] Vladimir Svetnik, Andy Liaw, Christopher Tong, J. Christopher Culberson, Robert P. Sheridan, and Bradley P. Feuston. 2003. Random Forest: a Classification and Regression Tool for Compound Classification and QSAR Modeling. *Journal of Chemical Information and Computer Sciences* 43, 6 (2003), 1947–1958.

[32] Philip H. Swain and Hans Hauska. 1977. The Decision Tree Classifier: Design and Potential. *IEEE Transactions on Geoscience Electronics* 15, 3 (1977), 142–147.

[33] Rebecca Taft, Irfan Sharif, Andrei Matei, Nathan VanBenschoten, Jordan Lewis, Tobias Grieger, Kai Niemi, Andy Woods, Anne Birzin, Raphael Poss, Paul Bardea, Amruta Ranade, Ben Darnell, Bram Gruneir, Justin Jaffray, Lucy Zhang, and Peter Mattis. 2020. CockroachDB: The Resilient Geo-Distributed SQL Database. In *SIGMOD*. 1493–1509.

[34] Avishay Traeger, Erez Zadok, Nikolai Joukov, and Charles P. Wright. 2008. A Nine Year Study of File System and Storage Benchmarking. *ACM Trans. Storage* 4, 2 (2008), 1–56.

[35] Alexandre Verbitski, Anurag Gupta, Debanjan Saha, Murali Brahmadesam, Kamal Gupta, Raman Mittal, Sailesh Krishnamurthy, Sandor Maurice, Tengiz Kharatishvili, and Xiaofeng Bao. 2017. Amazon Aurora: Design Considerations for High Throughput Cloud-Native Relational Databases. In *SIGMOD*. 1041–1052.