TexBench: A Unified Benchmarking Suite for Shifting Workloads

Abhishek Chanda*
Independent Researcher
USA
abhishek.becs@gmail.com

Shubham Kaushik*
Brandeis University
USA
kaushiks@brandeis.edu

Artem Lavrov
Brandeis University
USA
artemlavrov@brandeis.edu

Subhadeep Sarkar
Brandeis University
USA
subhadeep@brandeis.edu

ABSTRACT

Key-value stores are widely adopted as the storage engine for modern applications as they offer high throughput for writes, support for heterogeneous workloads, and easy tunability. Given the large number of key-value stores available and how widely their performance varies with workload characteristics, finding the suitable data store and tuning for a specific workload and performance target often entails extensive benchmarking and analysis. State-of-the-art key-value benchmarks, such as YCSB, db_bench, and KVBench, however, are unable to capture several key characteristics of modern application workloads, such as dynamically shifting workload characteristics, data with varied degrees of sortedness, or application-specific data formats. Further, existing benchmarking tools do not provide a unified interface to benchmark and compare multiple databases against the same workload.

We present *TexBench*, a unified key-value benchmarking suite that enables benchmarking key-value stores against dynamically shifting and production-like workloads and comparing their performance side by side. *TexBench* is built on top of Tectonic, a highly configurable, Rust-based key-value workload generator that can generate multi-phased shifting workloads, supports a rich set of operations and operation-specific distributions, variable data sortedness, and custom data formats. *TexBench*'s unified framework also enables its users to perform an apples-to-apples comparison of multiple databases against the same workload, and compare the benchmarking results readily within a single interface. Lastly, we augment *TexBench* with an LLM core that allows users to describe a workload in natural language, have that translated into a custom key-value workload, and benchmark and compare multiple databases against it in parallel. The demo is available at https://github.com/SSD-Brandeis/TexBench.

PVLDB Reference Format:
Abhishek Chanda, Shubham Kaushik, Artem Lavrov, and Subhadeep Sarkar. *TexBench*: A Unified Benchmarking Suite for Shifting Workloads. PVLDB, 14(1): XXX-XXX, 2020.
doi:XX.XX/XXX.XX

1 INTRODUCTION

Key-Value Stores are Everywhere. Key-value stores offer high throughput for data ingestion, support efficient storage of heterogeneous data, and are easily tunable for specific applications [5]. Thus, key-value storage engines, including RocksDB at Meta, Rockset at OpenAI, BigTable and LevelDB at Google, Cassandra and HBase at Apache, and FASTER and Cosmos at Microsoft, have gained notable traction in recent years. However, the performance of these data stores varies significantly – by several orders of magnitude – as the workload composition and distribution change [1]. Thus, given a workload and a target performance, finding the suitable storage engine and its optimal tuning is a non-trivial task that often entails extensive large-scale benchmarking [8].

Key-Value Benchmarks & their Limitations. Behind the simplistic *put-get* APIs of key-value stores, there is a rich design space of key-value operations and distributions that encapsulates the diverse access patterns of modern applications [1, 8]. Table 1 presents a list of diverse key-value operations and workload characteristics that are common in production workloads. While existing key-value benchmarking suites, such as YCSB [2], db_bench [4], and KVBench [9], support a set of these operations and distributions, (A) they are *unable to emulate several key characteristics* of real-world workloads. For instance, as shown in Table 1, YCSB supports only a few key distributions (uniform and Zipfian) and cannot generate point or range deletes; db_bench is tightly coupled with RocksDB; KVBench does not support read-modify-writes and incurs a high memory footprint; and none of them support benchmarking against data with varied degrees of sortedness [6]. Further, (B) none of the existing suites can benchmark databases against dynamic workloads, where the workload characteristics shift over time. Shifting workloads are very common in production – for instance, workloads like UDB, ZippyDB, and UP2X at Meta vary across multiple dimensions over a 7-day period, including (i) varied key-value sizes, (ii) operation mix, (iii) operation distributions, and (iv) access patterns [1, 3] – yet existing benchmarking suites are unable to capture such dynamic workloads. Lastly, (C) existing benchmarking suites lack a unified, user-friendly interface that allows users to compare multiple database benchmarks side by side.

The *TexBench* Benchmarking Suite. To this end, we introduce *TexBench*, a unified key-value benchmarking suite that allows users to benchmark databases against dynamically shifting and production-like workloads, and compare the database performance side-by-side. *TexBench* builds on top of our prior work Tectonic[1], which is a scalable, resource-conscious, and highly configurable Rust-based key-value workload generator, designed to generate *multi-phased shifting workloads*, alongside a *rich set of operations* and *operation-specific distributions*, variable *data sortedness*, and *custom data formats* [6]. We augment *TexBench* with an interactive user interface (UI) that allows users to benchmark multiple databases against the same workload, and compare and visualize the performance results. Further, we integrate *TexBench* with a

*Authors with equal contribution.

Proceedings of the VLDB Endowment, Vol. 14, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

[1] GitHub: https://github.com/SSD-Brandeis/Tectonic.

large language model (LLM) core that allows users to describe a workload in natural language and have that translated into a custom key-value workload and benchmark databases against it.

Goal. We designed *TexBench* with the goal of making database benchmarking accessible, straightforward, and insightful for the broader database community – from expert practitioners and researchers to inexperienced, yet curious users. We will demonstrate to the participants *TexBench*'s ability to:

(1) benchmark databases using key-value benchmarking suites, such as YCSB, KVBench, db_bench, and Tectonic,
(2) generate and benchmark databases against shifting workloads that change properties over time,
(3) benchmark multiple databases against the same workload, and compare their performance side-by-side, and
(4) generate complex, custom workloads from simple natural language descriptions and benchmark databases against them.

TexBench encapsulates all of this in a single UI, simplifying the process of database benchmarking. It also allows users to customize, verify, modify, and download the workload and benchmarking log from within the same interface.

2 THE *TexBench* BENCHMARKING SUITE

The *TexBench* benchmarking suite is designed to abstract out the complexities of database benchmarking and analysis, and make it an accessible and insightful undertaking. The *TexBench* framework has two main components. (A) At its core is Tectonic, a scalable, resource-conscious, and highly configurable Rust-based workload generator, curated to support complex workload properties, such as shifting workloads, variable data sortedness, and custom data formats [6]. (B) An interactive UI that allows users to (i) describe complex workloads in natural language and (ii) benchmark and compare the database performance against the workloads.

2.1 Tectonic

First, we describe the architecture of Tectonic, the workload generation process, and its key features. Tectonic is a highly configurable Rust-based workload generator that supports a wide range of key-value operations and operation-specific distributions, along with several sophisticated structural controls for generating realistic workloads. To the best of our knowledge, Tectonic is the first key-value benchmark that allows benchmarking of databases under dynamically shifting workloads and comparing their performance separately for each phase.

Design and Overview. Figure 1 shows the architecture of Tectonic, comprising five key components.

1) *Workload specification:* A JSON file is used to describe the workload specifications *phase-wise* using *Sections* and *Groups* (phases). A *Section* is a collection of *Groups* that share the same key domain, and each *Group* is a collection of interleaved key-value operations, such as inserts, updates, point and range queries, read-modify-writes, and deletes. Phases that operate on disjoint key domains are put into separate *Sections*. Tectonic generates each *Section* and *Group* sequentially.
2) *Parser:* The parser parses the JSON specification and initializes the required libraries in Tectonic. It also sets up the coroutines used by subsequent components, including the distributions of the different operations. Note that Tectonic supports a large number of distributions, and each operation in a workload may follow a different distribution.
3) *ActiveKeySet:* This data structure is used to store all the "live" keys during workload generation in a *KeySet*. The *KeySet* is accessed when generating or selecting keys based on their existence in the database, i.e., for operations such as unique inserts, updates, point queries on existing (or non-existing) keys, etc. Tectonic provides six types of *KeySets*: VectorKS, HashSetKS, HashMapIndexKS, OptionKS, B^+-TreeKS, and BloomFilterKS. Each *KeySet* is a specialized data structure for efficiently supporting specific types of operations.
4) *Generator & Selector:* These modules generate (from unique inserts) and select keys for queries, respectively, based on some specified distributions, and pass them on to the writer.
5) *Writer:* The writer is responsible for projecting the operations to the target database API or to an output stream. Tectonic generates values for inserts, updates, and read-modify-writes on the fly to significantly reduce the memory footprint.

Next, we highlight a few of the salient features of Tectonic that are critical for accurate modeling of real-world application workloads, and yet remain unsupported by existing key-value benchmarks.

Generating Shifting Workloads. To emulate the different phases of a shifting workload with changing properties, Tectonic uses *Groups* and *Sections*. Workload phases that operate on the same key domain are specified within the same *Section*, while phases operating on disjoint key domains (typical for different applications) are separated by *Sections*. The *Groups* within a section are generated sequentially, while within a *Group* all operations are interleaved.

Table 1: Tectonic supports a diverse set of operations, distributions, and structural controls, to model real-life workloads.

		YCSB	KVBench	db_bench	Tectonic
operations	insert	✓	✓	✓	✓
	update	✓	✓	✓	✓
	read-modify-write	✓		✓	✓
	point query	✓	✓	✓	✓
	empty point query		✓	✓	✓
	range query	✓	✓	✓	✓
	point delete		✓	✓	✓
	empty point delete		✓		✓
	range delete		✓	✓	✓
distributions	uniform	✓	✓	✓	✓
	normal		✓	✓	✓
	beta		✓		✓
	zipfian	✓	✓		✓
	exponential			✓	✓
	log normal				✓
	poisson				✓
	weibull				✓
	pareto			✓	✓
properties	dynamic workload shifts		*	*	✓
	context-aware shifting				✓
	data sortedness				✓
	variable query selectivity	✓		✓	✓
	variable key-value length				✓
	temporality-based access			✓	✓
	customizable key prefix			*	✓
	composite keys				✓

* requires significant manual intervention

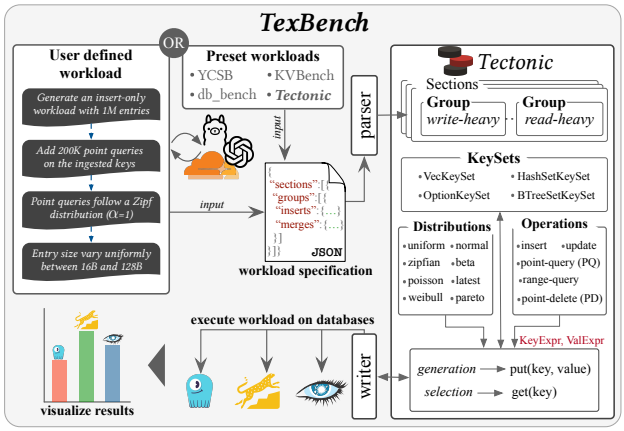

Figure 1: *TexBench* design and flow of workload generation.

Configurable Data Sortedness. Production workloads often reveal a nearly sorted nature in their ingestion and access patterns [7]. Tectonic allows users to generate workloads with varying degrees of sortedness. It uses the K, L-sortedness metric to generate arbitrarily ordered key sets, where K denotes the number of keys that are unordered and L denotes the maximum displacement from the correct position. Unlike BoDS, Tectonic supports data sortedness over an alphanumeric key domain while respecting specified distributions with interleaved operations.

Finer Control Over Key Construction. A fine-grained control over the key generation process allows Tectonic to produce composite keys, such as "`user:12345/profile`" or "`org:abc:project:xyz`", that are very common in production data stores. Tectonic offers better control over key segments and their distributions. Users can specify (i) fixed or varied key lengths, (ii) segments and their distributions, and (iii) segment construction methods using expressions.

More examples of production workloads that can be generated using Tectonic are presented in the base paper [6].

2.2 The *TexBench* UI

Overview. Figure 2 presents a snapshot of the *TexBench* UI. The starting interface allows users to either (i) choose from an existing benchmark family, such as YCSB, KVBench, db_bench, and Tectonic, or (ii) design a custom workload by describing it. Based on the input, *TexBench* generates a JSON specification that is automatically validated against the schema provided by Tectonic. The user may choose one or more databases, including RocksDB, Cassandra, and ScyllaDB, for benchmarking against the specified workload. *TexBench* first generates the workload and then executes it on selected databases one at a time, before visualizing the results.

We also integrate LLMs into *TexBench*, which allows for efficient translation of natural language workload descriptions to Tectonic specifications. In this mode, the application sends the current workload context and the user's prompt to the LLM layer and expects back a valid Tectonic JSON specification, which can be iteratively refined. This integration is provider-agnostic and presently supports Ollama and OpenAI at the backend.

Pipeline flow. *TexBench*'s **input panel** provides a flexible *workload editor* and an intuitive *workload runner*, allowing participants to benchmark key-value stores, such as RocksDB and Cassandra, and visualize their performance. Workloads can be defined in two ways. First, users may select an existing benchmark family (1) and choose a corresponding workload specification file (2). For example, a database can be evaluated using the YCSB benchmark family with a *Workload A* specification file. Alternatively, users may describe their custom workload in the input field of panel (3) and click *Apply*. Using the input description, *TexBench* leverages LLMs to construct a workload composition, which is populated in the workload editor. Users can verify and modify the workload by editing different *Sections* (4), workload phases (*Groups*) (5), and the operations in each phase (6). Checkboxes under the **operations panel** allow users to include or exclude operations from the selected workload phase. The workload editor allows users to (i) modify operation-specific distributions (e.g., Zipfian for point queries, uniform for range queries), (ii) update the key structures (e.g., specifying a prefix or segment), and (iii) adjust the number of operations.

The *workload generator* and **runner panel** (right side) provide a description of the workload and buttons to view, copy, and download the specification (7). The functionality facilitates generating specifications when integrating Tectonic across multiple projects to experiment with realistic and dynamically shifting workloads. *TexBench* abstracts away the benchmarking complexities, enabling users to compare multiple databases with minimal effort. Users select the desired databases (8) and click *Run Workload* (9) to get the comparison. Finally, *TexBench* provides a detailed breakdown of metrics, including overall throughput, average latency, per-operation throughput and latency, and phase-level metrics (10). This simplifies the process of database benchmarking, enabling users to perform effective comparisons across multiple databases.

3 DEMONSTRATION SCENARIOS

We demonstrate *TexBench* across two scenarios, with opportunities for interested participants to explore the advanced features.

Scenario 1: Comparing Databases against Preset Benchmarks. We introduce the participants to the *TexBench* UI by allowing them to compare database performance using simple preset benchmarks. Participants can select one out of the 23 preset workloads from four benchmark families: (i) YCSB, (ii) KVBench, (iii) db_bench, and (iv) Tectonic, and pick two or more databases to benchmark. Once the participant initiates the benchmarking process, *TexBench* generates the workload and executes it on the selected databases one at a time. Upon completion, results are projected to the *Benchmark Results* panel both in the form of absolute numbers and comparative plots across several performance metrics, including throughput, mean and tail latency, alongside a per-operation analysis. This provides a unified platform for easy, out-of-the-box comparison of multiple databases with minimal engineering effort.

Scenario 2: Benchmarking against Shifting Workloads. Next, we show the participants how Tectonic generates multi-phased shifting workloads and how to interpret the benchmarking results for shifting workloads. For this, participants can either (i) choose one of the preset dynamic workloads, namely, `Tec-1`, `Tec-4`, or `Tec-5` [6], or (ii) manually plug in the workload specification by

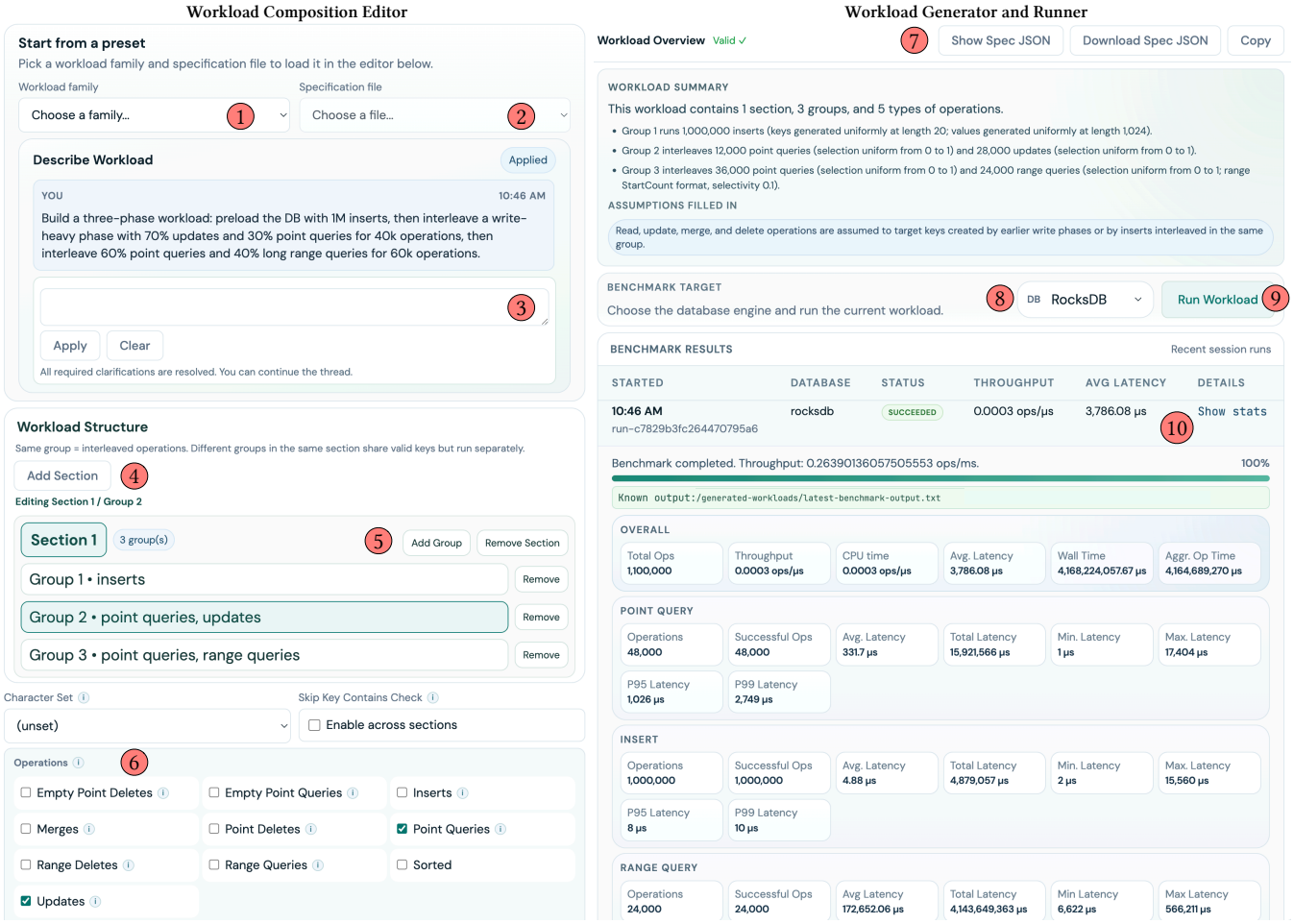

Figure 2: *TexBench* enables users to benchmark and compare databases against customized, dynamically shifting workloads.

filling out input fields in the UI. After execution, results are presented both as an aggregate and separately for each phase. This allows the participants to compare the performance of the databases phase-wise and analyze their implications at a granular level.

Bonus Scenario: LLM-Assisted Custom Workload Generation. Finally, for the interested participants, we demonstrate *TexBench*'s capability of generating custom workloads by interpreting natural language descriptions. *TexBench*'s interactive prompt allows the participants to iteratively describe the desired workload and have it translated on the fly to a Tectonic JSON specification. Specifically, we demonstrate how, based on a participant's workload description, *TexBench*'s LLM core can split the workload into multiple phases.

4 CONCLUSION

In this demonstration, we introduce *TexBench*, a unified key-value benchmarking suite that allows for benchmarking databases against dynamically shifting workloads and comparing the performance of multiple databases side by side. We demonstrate how the interactive UI of *TexBench* makes database benchmarking and analysis accessible, yet insightful for all classes of database users.

REFERENCES

[1] Zhichao Cao, Siying Dong, Sagar Vemuri, and David H. C. Du. 2020. Characterizing, Modeling, and Benchmarking RocksDB Key-Value Workloads at Facebook. In *Proc. of the USENIX Conference on File and Storage Technologies (FAST)*. 209–223.

[2] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. 2010. Benchmarking Cloud Serving Systems with YCSB. In *Proc. of the ACM Symposium on Cloud Computing (SoCC)*. 143–154.

[3] Siying Dong, Andrew Kryczka, Yanqin Jin, and Michael Stumm. 2021. RocksDB: Evolution of Development Priorities in a Key-value Store Serving Large-scale Applications. *ACM Transactions on Storage (TOS)* 17, 4 (2021), 26:1—-26:32. https://doi.org/10.1145/3483840

[4] Facebook. 2024. db_bench. *https://github.com/facebook/rocksdb/wiki/Benchmarking-tools* (2024).

[5] Stratos Idreos and Mark Callaghan. 2020. Key-Value Storage Engines. In *Proc. of the ACM SIGMOD International Conference on Management of Data*. 2667–2672.

[6] Alexander H. Ott, Shubham Kaushik, Boao Chen, and Subhadeep Sarkar. 2025. Tectonic: Bridging Synthetic and Real-World Workloads for Key-Value Benchmarking. In *Performance Evaluation and Benchmarking - TPC Tech. Conf. (TPCTC)*.

[7] Aneesh Raman, Konstantinos Karatsenidis, Subhadeep Sarkar, Matthaios Olma, and Manos Athanassoulis. 2022. BoDS: A Benchmark on Data Sortedness. In *Performance Evaluation and Benchmarking - TPC Tech. Conf. (TPCTC)*. 17–32.

[8] Subhadeep Sarkar, Dimitris Staratzis, Zichen Zhu, and Manos Athanassoulis. 2021. Constructing and Analyzing the LSM Compaction Design Space. *Proc. of the VLDB Endowment* 14, 11 (2021), 2216–2229.

[9] Zichen Zhu, Arpita Saha, Manos Athanassoulis, and Subhadeep Sarkar. 2024. KVBench: A Key-Value Benchmarking Suite. In *International Workshop on Testing Database Systems (DBTest)*. 9–15.