DB-MAGS: Multi-Anomaly Data Generation System for Transactional Databases

Yiqi Shen
East China Normal University
yqshen@stu.ecnu.edu.cn

Sijia Li
East China Normal University
sjli@stu.ecnu.edu.cn

Miaodong Shen
East China Normal University
mdshen@stu.ecnu.edu.cn

Peng Cai
East China Normal University
pcai@dase.ecnu.edu.cn

Weiyuan Xu
Meituan
xuweiyuan02@meituan.com

Kai Li
Meituan
likai44@meituan.com

Jinlong Cai
Meituan
jinlong.cai@meituan.com

ABSTRACT

Existing database performance anomaly datasets have the problems of comprehensiveness in anomaly types, coarse-grained root causes, and unrealistic simulation for reproducing concurrent anomalies. To address these issues, we propose a data generation system tailored for Multi-Anomaly Reproduction in Databases (DB-MAGS). DB-MAGS guarantees unified, authentic, and comprehensive data generation, while also providing fine-grained root causes. In the case of only a single anomaly occurred in the database, we categorize the factors affecting database performance anomalies, select five major categories of anomalies, and further subdivide each category into eighteen minor categories. This finer granularity of anomaly classification facilitates more specific and targeted anomaly remediation. For multiple anomalies simultaneously occurred in a database system, we categorize the relationships between anomalies into causal and concurrent, and enumerate different combinations of multiple anomalies, making the simulation of multiple anomaly scenarios more comprehensive and enhancing the diversity of generated data.

PVLDB Reference Format:
Yiqi Shen, Sijia Li, Miaodong Shen, Peng Cai, Weiyuan Xu, Kai Li, and Jinlong Cai. DB-MAGS: Multi-Anomaly Data Generation System for Transactional Databases. PVLDB, 17(12): 4497 - 4500, 2024.
doi:10.14778/3685800.3685909

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/qifeng1128/DB-MAGS.

1 INTRODUCTION

In modern database systems, performance anomalies are a prevalent and intricate issue. They often result from various factors like resource bottlenecks and lock conflicts instigated by problematic SQL, and usually don't occur in isolation. This phenomenon, known as multi-anomaly, can be concurrent or causal, where one anomaly triggers another, creating a cause-and-effect anomaly. The relationship between anomalies and their root causes adds to the complexity. Typically, a single anomaly corresponds to a single root cause. However, in multi-anomalies, if there's a cause-and-effect relationship, the root cause of the cause anomaly is considered most crucial. Concurrent anomalies, however, suggest multiple root causes. These unexpected anomalies can significantly disrupt database services, potentially leading to service crashes. Thus, accurately identifying and resolving these root causes is critical for stable database operation.

Data-driven root cause diagnosis methods, such as DBSherlock [4] and BALANCE [1], are constrained by the availability of training data. A unified, comprehensive, and real performance anomaly dataset would allow for a more accurate evaluation of different root cause diagnosis algorithms, driving further advancements in this field. However, there's a shortage of publicly available datasets, both in industry and academia. Therefore, creating a system capable of generating comprehensive and realistic database performance anomaly data is urgently needed.

Limitations of Existing Performance Anomaly Dataset. Although transactional database systems serve for mission-critical applications, to the best of our knowledge, there is only one performance anomaly benchmark (i.e. DBPA [2]) released publicly. However, it inadequately covers the breadth of anomaly categories. For single anomaly, one major category may include various subtypes, and the healing methods for different subtypes are distinct. DBPA has no detailed classifications for these subtypes. For multiple anomalies, existing benchmarks typically assume independence among them, overlooking their potential dependencies. Simulating real-world multiple anomalies is also challenging. DBPA [2] has not successfully injected multiple anomalies into the running database system, instead synthesizing data generated by single anomaly injection. Authentic data, not artificially generated, is essential for accurately evaluating root cause localization techniques. Furthermore, DBPA falls short in providing corresponding anomaly types and root causes, including detailed root cause SQLs. Different diagnosis techniques vary in the granularity of root cause localization. For instance, DBSherlock localizes to the type of anomaly, while BALANCE and PINSQL [3] localize to the detailed root cause SQLs.

Proceedings of the VLDB Endowment, Vol. 17, No. 12 ISSN 2150-8097.
doi:10.14778/3685800.3685909

In this paper, we propose a data generation system tailored for Multi-Anomaly Reproduction in Databases. We make following key contributions:

- Our data generation system incorporates comprehensive single and multiple anomaly injection combinations. For single anomaly types, we encompass a variety of principal anomaly categories and their respective subtypes. In terms of multi-anomaly injections, we consider combinations with both causal and concurrent relationships. This design enables our testing framework to thoroughly cover common database performance anomaly types, fulfilling a comprehensive performance anomaly dataset.
- Our demo includes a database performance anomaly injection and metric collection system. Users can recreate corresponding anomaly scenarios in the database system by inputting single or multiple anomaly types as per their requirements. The metric collection system can gather database performance anomaly data upon the occurrence of anomalies, thereby providing substantial data support for subsequent anomaly analysis and database healing stages.
- The dataset generated by DB-MAGS encompasses various single and multiple anomaly scenarios, along with corresponding anomaly types and root cause SQL. The construction of this dataset not only provides abundant experimental data for the study of database performance anomalies but also serves as empirical evidence for the localization and recovery of database performance anomalies.

2 DESIGN OF DATA GENERATION SYSTEM

Anomalies in database performance can manifest as a decrease in throughput or an increase in latency. We categorize database performance anomalies into single anomaly and multiple anomalies, and construct an anomaly reproduction framework to simulate the occurrence of database performance anomalies.

2.1 Anomaly Reproduction Framework

The architecture of Anomaly Reproduction Framework, as depicted in Figure 1, comprises four layers: User Interaction Layer, Anomaly Design Layer, Workload & Injection Layer, and Server Layer.

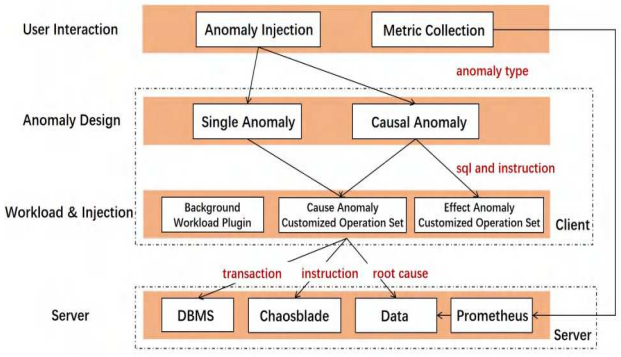

Figure 1: The architecture of Anomaly Reproduction Framework.

The User Interaction Layer serves as an interactive platform that facilitates user input, selection, and operation of various functionalities. In this layer, we design an intuitive user interface and develop a fault injection scheduling system to efficiently manage and monitor the fault injection process with a open-source tool, Grafana[1], creating a result display system to view the results and impacts of fault injection.

The Anomaly Design Layer's primary function is to generate specific operations for anomaly creation based on the anomaly type parameters provided by the user design layer. Initially, the framework breaks the anomaly type into single anomaly and causal anomaly which are in concurrent relations. Subsequently, based on the customized operation sets, it calls the backend interface and generates respective injection SQLs and instructions.

The Workload & Injection Layer mainly combines the injection SQLs and instructions obtained from the design layer into cause and effect anomaly operation set. It first runs the background workload (e.g. TPC-C, JOB, TATP, etc.). Subsequently, it initiates a connection to the database system in the server layer and injects the cause anomaly customized operation set. If there exists an effect anomaly customized operation set, it will be injected afterwards, thereby implementing multiple anomaly injections. This customized module ensures the system's high scalability and adaptability, capable of simulating diverse workloads and anomaly types effectively.

In the Server layer, the database system accepts connections initiated by the client and executes transactions passed from the background workload & injection layer. Chaosblade[2] receives instructions for resource bottleneck anomalies and injects these anomalies into the server. Additionally, the Prometheus[3] tool is responsible for collecting metric data, which includes the internal database status, performance metrics, types of anomalies, and root cause SQL, and stores them for further analysis.

2.2 Single Anomaly

The single anomaly classification are shown in Table 1, which categorizes anomalies into five major types: lock conflicts, traffic surges, slow SQLs, resource bottlenecks, and database table backups.

Simulating single anomaly typically involves three approaches: query injection, parameter tuning, and utilizing the Chaosblade tool. For instance, lock conflict anomalies are generated by injecting SQLs with locking on various objects such as table-level, metadata and row-level. Slow SQL anomalies are triggered by queries that operate on tables without indexes or with excessive indexes, as well as complex joins involving multiple tables, among others. Database backup anomalies are simulated using SQL injections with `mysqldump`[4] commands. Adjusting the concurrency of injected SQLs allows us to generate the surge in traffic caused by individual SQL queries. Moreover, by modifying the maximum number of threads connected to the database and the delay after task submission, we generate traffic surges across the entire workload. Lastly, utilizing Chaosblade, a tool designed to stress the fundamental resources of an operating system, we reproduce resource bottleneck anomalies such as CPU and network surges.

[1] https://github.com/grafana/grafana
[2] https://github.com/chaosblade-io/chaosblade
[3] https://github.com/prometheus/prometheus
[4] https://dev.mysql.com/doc/refman/5.7/en/using-mysqldump.html

Table 1: Single Anomaly Classification

Classification	Subtype	Reproduction Method
Lock Conflict	Record Lock Table Lock Metadata Lock	Query Injection
Traffic Surge	Single SQL Overall Workload	Parameter Tuning
Slow SQL	Missing Index Excessive Index Implicit Conversion Multi-table Join Order By Group By Large Table Scan	Query Injection
Resource Bottleneck	CPU I/O Network Memory Disk	Chaosblade
Database Backup	Database Table Backup	Query Injection

2.3 Relationship between Anomalies

The relationship between anomalies can primarily be categorized into two types: Causal Relationship and Concurrent Relationship. Identifying these relationships poses a significant challenge for database administrators (DBAs) in root cause detection process as they often exhibit similar performance metrics.

Concurrent Relationship. The concurrent relationship describes the simultaneous occurrence of two or more anomalies, but there is no direct causal relationship between these anomalies. The occurrence of these anomalies is independent of each other and needs to be triggered by separate anomaly injection processes. Therefore, for concurrent relationship anomalies, each anomaly needs to be independently located.

Causal Relationship. The causal relationship delineates an intrinsic causality, that is, under specific workloads or system environments, the occurrence of one anomaly (cause anomaly) may trigger another anomaly (effect anomaly). These two anomalies have a sequential relationship, but they often appear to be simultaneous from the perspective of DBAs. The crux of this relationship lies in the fact that without the triggering of cause anomaly, the effect anomaly loses its conditions for occurrence. Therefore, for causal relationship anomalies, the focus should be on locating and handling the cause anomaly. Due to workload constraints, the cause anomaly may can't directly trigger the effect anomaly without additional operations, such as constructing specific SQL injections.

Drawing upon the professional knowledge of DBAs and the recommendations of our industry partners, we have compiled a list of causal anomalies that are commonly encountered in production. These are presented in Table 2.

For instance, table lock conflict can extend SQL query execution time of SQL queries, particularly when executed on the same table and should be injected as the effect SQL. Surges in overall workload traffic can lead to record lock conflicts or resource bottleneck anomalies. This is due to the increased number of threads per second requesting transaction processing from the database, consequently escalating the transactions adding identical record locks to the same table. Concurrently, the augmented number of transaction processing threads can exhaust server resources, potentially causing resource bottlenecks. CPU resource bottlenecks can prolong the execution time of query SQL, potentially triggering slow query anomalies. Backups of Database table can induce anomalies in record lock conflicts, as the employment of the mysqldump command-line utility for backing up tables implicated in the background workload can obstruct SQL statements that are writing to the identical table.

Table 2: Common Causal Anomaly Classification

Cause Anomaly	Effect Anomaly	Effect SQL
Lock Conflict (Table Lock)	Slow SQL	YES
Traffic Surge (Overall Workload)	Lock Conflict (Record Lock)	NO
Traffic Surge (Overall Workload)	Resource Bottleneck	NO
Resource Bottleneck (CPU)	Slow SQL	NO
Database Table Backup	Lock Conflict (Record Lock)	NO

2.4 Enumerating All Multi-Anomalies

To cover a broader spectrum of multi-anomaly scenarios, including various single anomaly subtypes and diverse inter-anomaly relationships, we propose a methodology for enumerating all possible combinations of anomalies across N anomaly classifications, where each of these N classifications must be triggered.

Initially, we consider the condition where at least one subtype is selected for each classification. For anomaly subtype set A, set B, ..., the total number of anomaly combinations is calculated as follows:

Number of Total Combinations = $(2^{|A|} - 1) \times (2^{|B|} - 1) \times \ldots$

We denote these anomaly combinations as T. Each anomaly combination in T may involve causal relationship, concurrent relationship between single anomaly subtypes and anomalies with causal relationship, and concurrent relationship between single anomalies.

Due to the existence of causal relationships among anomalies, the effect anomaly can be triggered by injecting the cause anomaly. Consequently, for each pair of anomalies with causal relationships, we inject only the cause anomaly, excluding the effect anomaly. Therefore, for each anomaly combination t in T, we identify all causal relationship pairs contained in t to form the set t_{pair}. We enumerate the non-empty subsets of t_{pair}, remove the effect anomalies from t based on the non-empty subsets, and consider the resulting t' as a new anomaly injection method. All unique t' constitute the set T'. Finally, the anomaly injection methods covering all N anomaly classifications are the union of T and T'.

2.5 Monitored Performance Metrics

We utilized an open-source tool, Prometheus, to collect metrics of the operating system and database. For operating system metrics, our primary focus lies in monitoring the usage of CPU, memory, disk, network bandwidth and the time spent on doing I/Os. Regarding database metrics, our major emphasis is on monitoring the thread count, connection count, questions, slow queries count and table lock count.

3 DATA ANALYSIS

3.1 Data Differences

We illustrate the differences between real data from DB-MAGS and synthetic data from DBPA, demonstrating DBPA's inability to accurately reflect performance metrics.

Figure 2 shows the changes in CPU User and Com Commit metrics for DBPA and DB-MAGS under missing indexes and lock conflicts anomaly. The CPU User metric in DB-MAGS sharply increases, while DBPA shows stable trends, indicating that their trends are different. Similarly, the Com Commit metric declines in DB-MAGS but significantly rises in DBPA. Theoretically, CPU User should rise and Com Commit should decrease under multiple anomaly.

3.2 Inauthenticity of DBPA Data

The multi-anomaly data generated by DBPA model training shows inconsistency in the time of anomaly indicator changes. This inconsistency arises primarily from the varied indicator change points of different single anomalies. For example, Figures 3 and 4 show that memory usage and active thread indicators for lock conflict anomalies change at the 40th second, while those for missing index anomalies change at the 7th second. These change points between different single anomalies do not align. In Figure 5, memory usage indicators for lock conflict and missing index anomalies show significant changes at the 25th second, aligning with the missing index trend, while active thread indicators change at the 45th second, aligning with the lock conflict trend. Theoretically, active thread indicators should rise at the 25th second due to missing index anomalies, but the synthesized data remains stable. Thus, the different change points make the model-generated multi-anomaly data unreliable and unrealistic.

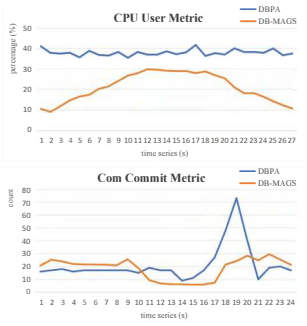

Figure 2: CPU User and Com Commit of DBPA and DB-MAGS under multiple anomaly.

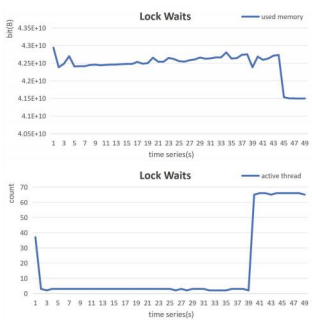

Figure 3: Used Memory and Active Thread of DBPA under lock waits anomaly.

4 DEMONSTRATION

In this demo, we will illustrate how to inject custom database performance anomalies, encompassing single and multiple anomaly scenarios, and collect the corresponding monitoring metric data. Prior to anomaly injection, as depicted in Figure 6, system collects the inputted parameters for anomaly injection. Users specify the anomaly categories and combinations to custom anomaly type and

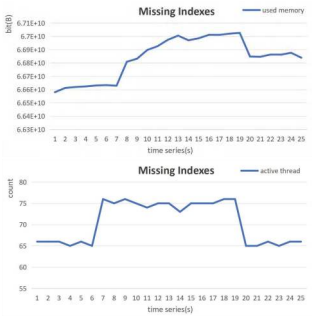

Figure 4: Used Memory and Active Thread of DBPA under missing indexes anomaly.

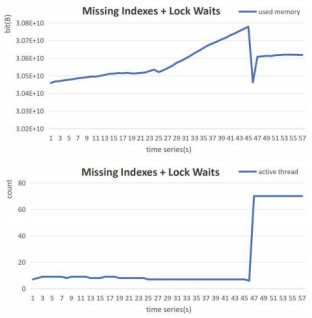

Figure 5: Used Memory and Active Thread of DBPA under multiple anomaly.

concurrency of injecting root cause to determine anomaly severity. Throughout anomaly injection, as shown in Figure 7, users can click the button of "View Metric Changes" and jump to the visualization page to monitor trends in operating system and database metrics. Post to anomaly injection, the system will trigger the tool, Prometheus, to collect and store metrics in user-specified paths.

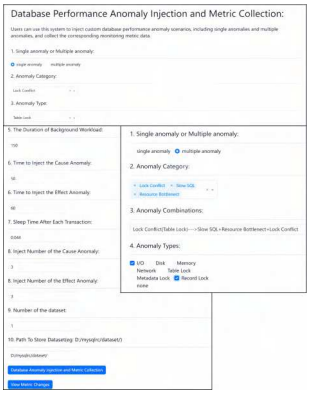

Figure 6: User Interface.

Figure 7: Metrics Dashboard.

ACKNOWLEDGMENTS

This work was supported by grants from the National Natural Science Foundation of China (U22B2020) and Meituan. Peng Cai is the corresponding author.

REFERENCES

[1] Chaoyu Chen, Hang Yu, Zhichao Lei, Jianguo Li, Shaokang Ren, Tingkai Zhang, Silin Hu, Jianchao Wang, and Wenhui Shi. 2023. BALANCE: Bayesian Linear Attribution for Root Cause Localization. *Proc. ACM Manag. Data* 1, 1 (2023), 95:1–95:26.

[2] Shiyue Huang, Ziwei Wang, Xinyi Zhang, Yaofeng Tu, Zhongliang Li, and Bin Cui. 2023. DBPA: A Benchmark for Transactional Database Performance Anomalies. *Proc. ACM Manag. Data* 1, 1 (2023), 72:1–72:26.

[3] Xiaoze Liu, Zheng Yin, Chao Zhao, Congcong Ge, Lu Chen, Yunjun Gao, Dimeng Li, Ziting Wang, Gaozhong Liang, Jian Tan, and Feifei Li. 2022. PinSQL: Pinpoint Root Cause SQLs to Resolve Performance Issues in Cloud Databases. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. 2549–2561.

[4] Dong Young Yoon, Ning Niu, and Barzan Mozafari. 2016. DBSherlock: A Performance Diagnostic Tool for Transactional Databases. In *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016*. ACM, 1599–1614.