TracEx: Understanding and Analyzing Database Traces

Dominik Durner
Technische Universität München
Germany
dominik.durner@tum.de

Lennart Espe
Technische Universität München
Germany
lennart.espe@tum.de

Jana Giceva
Technische Universität München
Germany
jana.giceva@in.tum.de

Anja Gruenheid
Microsoft GSL
Switzerland
anja.gruenheid@microsoft.com

Abstract

With the shift to databases-as-a-service, vendors are able to collect high-level database traces of executed workloads while retaining the privacy of their customers. In contrast to pure end-to-end latency statistics, traces contain enriched information that is useful for tasks such as workload monitoring and regression testing. Despite its importance, efficient analysis and exploration of traces and their rich feature space remains a challenge. In this paper, we introduce TracEx, an open-source **Trac**e **Ex**ploration tool that facilitates workload trace analysis and comparison for database systems. TracEx allows users to understand their workload by providing an intuitive, visual interface that explores the workload along different dimensions, e.g., resource utilization or database operator usage. Additionally, users are able to contrast and compare workloads that have been collected from different hardware configurations or even compare traces between database systems.

1 Introduction

The shift from on-premise database management systems (DBMS) to managed databases-as-a-service in cloud environments has enabled a wider variety of customers to use DBMSs on an everyday basis. At the same time, the way customers use database systems has diversified, creating new (types of) workloads in addition to established types like transactional and analytical processing, such as hybrid or graph-based workloads. Understanding how workloads are executed, optimizing resource utilization, and ensuring optimized execution of workloads in cloud database systems can be challenging for both providers and customers due to their inherent complexity. To address these challenges, managed database systems, e.g., Snowflake [3], Azure Synapse [1], and AWS Redshift [9], continuously monitor and record statistics on query executions in so-called *traces* that often contain workload characteristics such as performance data, resource metrics, and query execution statistics, while adhering to privacy regulations. In contrast to pure end-to-end latency statistics, traces can be used to enrich information collected for workload monitoring [16, 26] and regression testing. Despite its importance, efficient analysis and exploration of traces and their rich feature space remains a challenge. We identify that most challenges can be attributed to one of the following reasons:

(i) Extracting valuable features from raw data requires intricate knowledge of how the system operates because traces can include hundreds of features.
(ii) Mapping the features of interest into a single feature space is non-trivial, given a multitude of factors that interact with one another.

In our work, we investigate the use of DBMS traces to gain a more comprehensive understanding of a workload. This involves not only evaluating the performance of a workload, but also examining characteristics such as resource utilization, execution details (e.g., query plan), and the setup of a system's environment (e.g., whether caching is enabled).

Example 1.1 (Workload Analysis). A cloud provider's engineering team has completed a new release of its DBMS and is now looking to prioritize its next milestones, i.e., whether to improve the scan, join, or filter operator. To better understand what changes would provide the most value to their customers, they analyze the traces of their system. Figure 1 shows a visual analysis of the trace, where each query is assigned to a feature group (operator) if the majority of the execution time is spent on that operator. Based on this analysis, the engineers decide to improve the scan as most of the queries in the workload rely heavily on this operator.

Workload analyses like these can assist in developing a deeper understanding of the characteristics of a workload, allowing users to make decisions that guide and impact how workload execution and performance are improved in the long run. At the same time, performing easy-to-understand, visual analyses of multi-dimensional data is not trivial. Interpretation of these traces requires extensive domain knowledge to determine which features to display, as well as knowledge of how to map the feature values into a two- or three-dimensional space to facilitate intuitive understanding of the data. Thus, we argue that given the large amounts of data contained in DBMS traces (e.g., 7 GiB compressed data for 14 days of Snowset, a privacy-preserving trace from selected Snowflake customers [29]), reducing the search space and focusing on a variety of well-defined characteristics is a first step in understanding which parts of a single workload or trace should be examined more closely.

In addition to single-workload analysis, we also explore the problem of comparative workload understanding:

Example 1.2 (Workload Comparison). A customer wants to understand whether the benchmarks they are currently using mimic

 14th Annual Conference on Innovative Data Systems Research (CIDR '24). January 14-17, 2024, Chaminade, USA

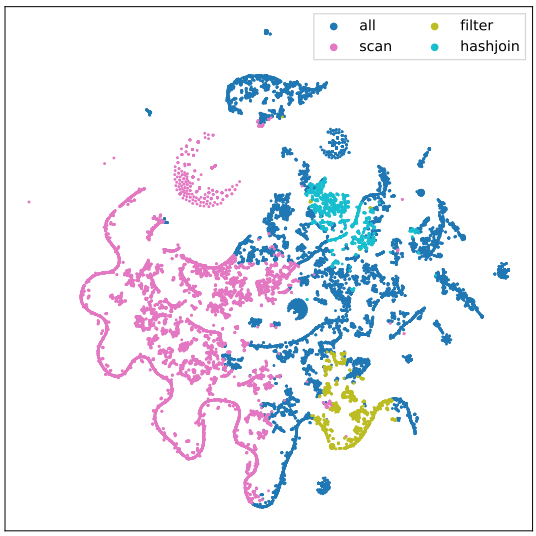


Figure 1: Query groups within the Snowset, clustered on operator shares.

their production workload. In particular, they want to ensure that the test suite has full coverage of all relevant DBMS operators and the query plans that are typically executed. Hence, they decide to capture the traces of the benchmark as well as the original workload and overlay them visually, to get a first impression of whether the benchmarks are sufficient. Figure 2 shows the resulting visualization. Looking at the difference between the observed benchmark and the original workload queries, they determine that the test coverage is insufficient and that they need to introduce additional test queries that will subsequently cover the remaining feature space.

The discussed examples focus on workload understanding and comparison. To address these and other challenges within the problem space of workload characterization and understanding, we introduce TracEx, a tool that allows users to visualize, compare, and contrast workloads. To the best of our knowledge, TracEx is the first workload analysis tool that allows users to evaluate their workload traces from one or more DBMS across a number of dimensions. It is designed in a modular way and currently supports trace exploration on SQL Server and Snowflake, but can easily be extended to other DBMS. TracEx enables users to answer research questions, like the ones above, by providing an easy-to-use graphical user interface and automatically generated visualizations. Thus, TracEx helps to overcome the identified challenges of using traces.

(i) TracEx assists in selecting relevant features by providing a close human-in-the-loop data exploration experience.
(ii) TracEx provides a set of visualizations that allow users to compare and contrast different workloads.

Furthermore, TracEx has been made publicly available as an open-source tool[1].

[1]https://github.com/lnsp/trace-explorer

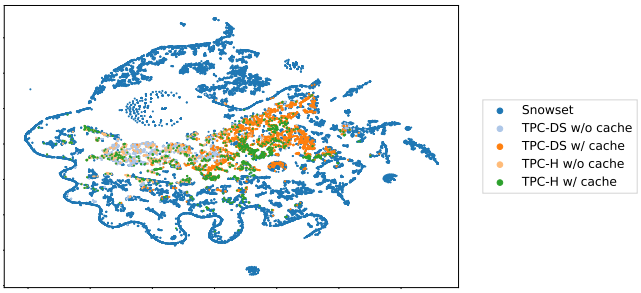


Figure 2: Snowset compared to the TPC-DS and TPC-H benchmarks (both SF10), clustered on operator shares.

2 Problem Statement & Design Considerations

In our work, we investigate the problem of workload visualization and interactive workload exploration. Specifically, we assume that there exists a set of log events E where each event $e \in E$ describes the execution of a database query. For example, the standardized TPC-H benchmark contains 22 queries. Thus, if the workload is executed once, E contains 22 events, where each event has a value v_i, numeric or categorical, for each recorded feature (attribute) a_i. Attributes $A = \sum_i a_i$ can be generic across or specific to a DBMS and describe information such as operator shares or resource utilization (e.g., compute resources or I/O operations).

2.1 Use case: Workload Analysis

In Example 1.1, we introduced the notion of workload analysis. More formally, we define *workload analysis* as the problem of understanding the characteristics of a single workload. The understanding of a workload can be based on a variety of dimensions. Specifically, workload traces contain a multitude of attributes, and determining which ones are important and should be examined more closely is non-trivial. For example, the Snowset [29] specifies query start and stop timestamps and execution duration by area of activity, such as query execution, control plane, and compilation time, as well as query profiling information. It also contains information about the warehouse, extensive information about I/O operations from S3, cache and local SSD, and network statistics. Given all these different types of attributes, we need to build a workload analysis tool that is generic enough to handle different types of statistics while allowing its users to efficiently explore the attribute space. Thus, the problem of workload analysis is to define an efficient and interactive framework that allows users to intuitively explore any of their attributes.

2.2 Use case: Workload Comparison

We refer to *workload comparison* as the ability to compare and contrast two (or more) different workloads along the same attributes. In Example 1.2, we discussed how workload comparison can help users understand the coverage of benchmarks w.r.t. a reference workload. Additionally, we envision that workload comparison techniques can be used to identify the evolution of a workload over time, i.e., the shift in query patterns or their resource consumption, or to contrast the execution of the same workload on different systems or hardware. The latter is particularly useful when cloud

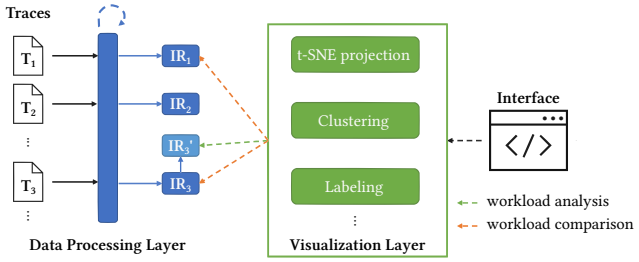


Figure 3: TracEx workflow: First, each dataset is transformed into our intermediate representation (IR). Additional aggregates and selections can be computed to generate derivatives of a dataset's IR. Afterward, TracEx visualizes the information on the desired dimensions. This is an iterative process facilitated by our easy-to-use interfaces.

service providers and their customers want to understand the impact of infrastructure changes and improve the resource utilization of a workload. Overall, we define the problem of workload comparison as providing an efficient and interactive framework that allows users to compare and contrast a (set of) workload(s).

2.3 TracEx Desiderata

Given these use cases, we observe several desiderata for TracEx:

Ease-of-Use. To make this tool feasible in practice, it needs to be intuitive to use, preferably through a graphical or web interface. We envision that this tool can be used by domain experts such as software engineers who want to dive deep into an issue and debug it, or data scientists who want to explore the attribute space for downstream machine learning applications. Thus, we want to provide an interface that allows users to explore their workload data independent of their domain knowledge.

Intuitive Visualizations. One of the biggest challenges in workload visualization is finding a good level of abstraction that highlights all the insights without overwhelming the user. Without a layer of abstraction, it is often not possible to visualize even a subset of a dataset.

Independence of Attributes. Different workload traces may contain a variety of generic and system-specific events and attributes. When implementing TracEx, we need to abstract from these attributes and build workflows that allow users to explore attributes independent of the exact variation of the attribute. In essence, we need to identify groups of attributes, such as numeric time-series attributes, categorical attributes, etc., around which we can then build our visualizations.

Extensibility. We developed TracEx as an open-source tool and, to further adoption, we envisioned TracEx as a modular library that allows developers to extend it with minimal overhead. For example, TracEx currently supports the extraction and processing of traces from two different database systems, Snowflake and SQL Server. New DBMSs can be added as new modules by leveraging the existing systems as examples. Similarly, we require that analysis and visualization techniques are themselves modules within TracEx so that they can, also, be extended in a straightforward manner.

3 TracEx Architecture

TracEx is designed for comprehensive trace analysis of execution traces produced by many commercial and open-source database systems. Initially, the generated database traces are converted into our TracEx intermediate representation. This intermediate representation (IR) serves as a data table where each cell value corresponds to the recorded value of an attribute for each given event. Users interacting with TracEx can then manipulate both the data in the IR as well as the visualizations interactively as shown in Figure 3. For example, the user can first examine the differences in traces T_1 and T_3 by clustering the operators used for executing these workloads. If they see discrepancies in the database I/O operator shares, the user can follow up and manipulate the IR of T_3, IR_3, to use only those events that describe I/O operations (IR_3'). They then decide to visualize a more in-depth analysis of the I/O resources used for this workload. This example workflow showcases the two main components of our tool, the data processing layer and the data visualization layer.

3.1 Data Processing Layer

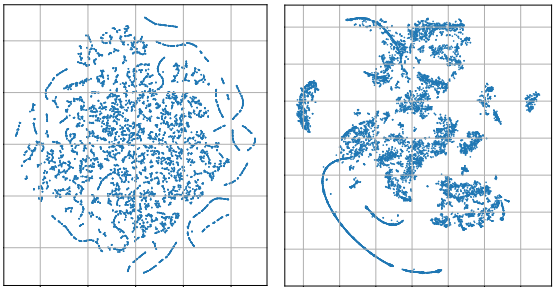
Preprocessing the data, i.e., loading and transforming the data into a common intermediate format, is crucial to streamline the pipeline of TracEx. Our input data can be of any type, but custom transformers into our row-wise intermediate language are required. Similar to industry pipelines [8], we first extract and clean the data, for example using outlier detection mechanisms, and then refine the dataset according to the attributes, selected by the TracEx user. For TracEx, we choose a tabular representation as IR and store the resulting data in Parquet [7]. Furthermore, we use DuckDB [21], as a query engine on top of the IR to enable complex processing steps. Using a SQL layer to efficiently slice and dice the data enables the creation of data cubes, which we then use as input to our visualization pipeline. This allows us to push filtering steps into the data processing layer which reduces the input size of the visualization data, increasing the interactivity of our tool. In addition to augmenting the dataset on the fly, we also allow users to perform outlier removal. For example, TracEx can compute the standard score or z-score as $z_i = \frac{x_i - \mu_i}{\sigma_i}$ for numerical attributes by calculating the mean and standard deviation of attribute a_i. By using an attribute threshold, we can exclude events from the resulting dataset.

3.2 Data Visualization Layer

To find meaningful insights in the trace data, TracEx allows us to explore and visualize the data in multiple ways. Our tool supports overall and individual group visualization, clustering, and automatic labeling of groups.

A challenge that all of these techniques face is to run efficiently in an interactive setting. To address this, we employ dimensionality reduction to minimize the amount of data processed without significant loss of information using a well-established technique called *Principal Component Analysis* (PCA) [31]. Our PCA algorithm chooses the number of components to retain at least 95% of the variance of the input data.

3.2.1 Group Visualizations While PCA reduces the dimensionality of the dataset significantly, it is not able to perform sufficient



(a) $p = 10$ **(b)** $p = 70$

Figure 4: t-SNE visualization of a smaller Snowset subset (10000 observations) with varying perplexity, clustered on operator shares.

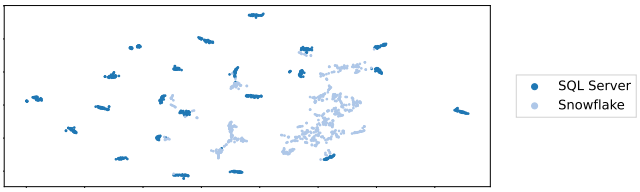
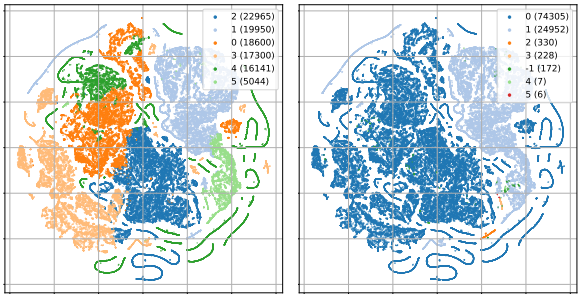


Figure 5: t-SNE visualization on TPC-H (SF10) for Snowflake and SQL Server ($p = 20, i = 2000$), clustered on operator shares.

transformations to allow for a two- or three-dimensional visualization. Therefore, we use t-Distributed Stochastic Neighbor Embedding (t-SNE) to visualize high-dimensional data in a two- or three-dimensional map [27]. While t-SNE is a powerful tool, a compact 2D representation of high-dimensional data can be misleading, as described by Wattenberg et al. [30]. The authors mention three key challenges in its deployment: a) A dependency on the chosen parameters of the algorithm, i.e., *perplexity* and the number of iterations, b) the normalization of cluster density, and c) the distance obfuscation between clusters. To address these concerns, we first take a closer look at the perplexity (p) parameter, which is used to change the trade-off between local and global features of a dataset. The clusters we see with t-SNE will typically become clearer as perplexity increases and need to be adjusted according to the dataset size. In addition to perplexity, t-SNE relies on the number of iterations (i) as a user-defined parameter. The literature recommends perplexity values between 5 and 200 (often less), and iterations of at least 250. Tools usually recommend $p = 30$ and $i = 1000$ as general starting points [6, 23].

Figure 4 shows the impact of perplexity on the performance of t-SNE on the Snowflake dataset. Specifically, we observe that setting perplexity $p = 10$ does not show any distinct clusters, but once we start increasing the parameter, we observe a clearer distinction between groups of queries in the visualization. Note that the distance between these groups is not representative. In particular, t-SNE obfuscates both the density of clusters and the distance between clusters through its dimensionality reduction. Therefore, it is not possible to visually compare t-SNE results from different datasets. To allow users to compare the same workload executed



(a) agglomerative **(b) DBSCAN**

Figure 6: Comparison of clustering methods on a subset of the Snowset.

on multiple different systems, we incorporated the possibility to combine datasets and visualize them as categories within a single t-SNE as shown in Figure 5. Here, the individual queries of the Snowflake and SQL Server are clearly different in their characteristics but consistent within themselves. For example, we observe distinct cluster groups for SQL Server indicating that the performance characteristics of almost all queries are different.

3.2.2 Clustering The visual clusters provided by t-SNE are an intuitive, but not absolute way to identify clusters and rely purely on user interpretation. To alleviate this problem, we implemented additional clustering techniques in TracEx.

Agglomerative hierarchical clustering. The key advantage of agglomerative hierarchical clustering is its independence of a predefined number of clusters, i.e., it is a threshold-based approach that minimizes the variance within clusters. Furthermore, it allows us to partition our datasets using a transparent strategy based on a defined linkage distance [22].

DBSCAN. Density-based spatial clustering of applications with noise (DBSCAN) is a clustering algorithm that discovers clusters of arbitrary shape [5]. The basic idea of DBSCAN is that the density within a cluster should exceed the density outside the cluster.

Comparing agglomerative hierarchical clustering and DBSCAN, we observe that the different clustering algorithms yield very different results, as shown in Figure 6. DBSCAN groups most of the data into two large clusters in the middle and keeps the outer data groups separate. The reason for this behavior is that the data has different density levels, making it difficult for the algorithm to identify clusters. In comparison, agglomerative clustering, as shown in Figure 6a, delivers a more nuanced view of the data. The separation between clusters is much clearer and more consistent with the visual t-SNE embedding. Note that both approaches use parameters that require human-in-the-loop tuning.

3.2.3 Workload Labeling Another important aspect of workload exploration is to label the visualized clusters so that users can directly understand why the visualized clusters exist. Our labeling algorithm summarizes the intuitive observation that a cluster is defined as a coherent group of events that have low intra-cluster variance for a set of attributes but large variance across clusters. To implement this idea, we first compute the mean μ_{C_i} and variance

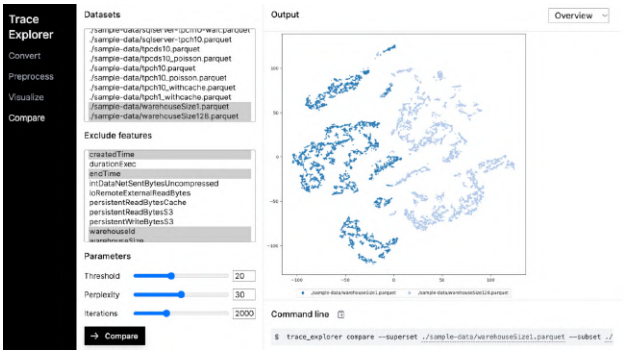


Figure 7: The Web UI of TracEx.

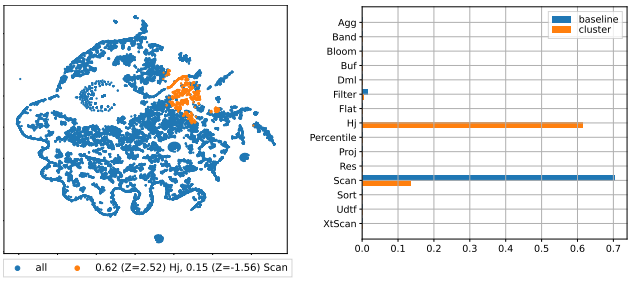


Figure 8: The query group analysis for a join-heavy cluster in the Snowset dataset (operator shares).

σ_{C_i} per attribute a_i for each cluster C. With these metrics, we compute a smoothed score $z_{C_i} = \left(\frac{\mu_{C_i} - \mu_i}{\sigma_i}\right) \cdot \left(\frac{1+\sigma_i}{1+\sigma_{C_i}}\right)$ per cluster and attribute. We use the cluster-wide (σ_{C_i}) and global (σ_i) standard deviations per attribute a_i for smoothing the score to mitigate the impact of large outliers on the mean value. In essence, this smoothed score identifies the difference per attribute between a selected cluster and the entire dataset.

We can sort the columns of each cluster by this smoothed score and choose the first x attributes to define the cluster label. A regular z-score greater than two shows that the attribute score is twice the standard deviation, and thus significantly different from the average within the dataset. Consequently, attributes with high scores define a cluster in a meaningful way, which we will demonstrate in Section 4.1. In contrast to using only principal components for t-SNE and the clustering algorithms, the labeling computation of mean, variance, and our smoothed score is relatively inexpensive and can be executed on the intermediate representation directly.

3.3 Human-In-The-Loop Exploration

In TracEx, we provide the user with two means of interacting with the framework, through a web and a command-line interface. The web UI is shown in Figure 7. Here, the user has the means to explore their database traces through the previously described exploration steps: By importing and pre-processing their data as well as subsequently visualizing it in various ways. We specifically designed TracEx to allow users to go back and forth between the different parts of the tool, which enables an iterative exploration process as follows: Users of TracEx essentially operate on an internal, intermediate representation as previously shown in Figure 3. This IR serves as the basis for scoping the data visualized in subsequent exploration steps. Thus, if a user wants to focus on a particular part of the data that they have identified as interesting during the data visualization, they can go back to the pre-processing step and re-scope the dataset as needed.

TracEx currently uses a selection of views that guides users to meaningful insights without overwhelming them with the details of visualization algorithms. The iterative nature of TracEx allows users to determine the most insightful visualizations or suggestions for which data to explore further with minimal guidance. The user can modify all parameters and datasets as they see fit, allowing for a fully customizable exploration process. As future work, we want to add the option to create templates that allow users to store their exploration workflows for easier reproducibility as well as providing some guidance for commonly deployed workflows specific to well-structured traces from known DBMS.

4 Demonstration

In our demonstration, we will focus on the user experience of interacting with the web UI, allowing them to interactively analyze the data as previously shown in Figure 7. We will provide trace data for exploration of experiments with Snowset, TPC-H, and TPC-DS from two different sources, SQL Server and Snowflake.

4.1 Walkthrough

As an example interaction with TracEx, consider the following demonstration scenario.

Step 1: Data Processing Layer. The user starts the demo by opening the web UI and uploading database traces to TracEx after which they can explore the IR, for example by filtering the input data.

Step 2: Data Exploration. Next, the user interacts with the dataset using SQL, for example by normalizing and scaling the output attributes or computing aggregates.

Step 3a: Data Visualization - Workload Analysis. After preprocessing, the data can be visualized using t-SNE, agglomerative clustering, and score-based labels, allowing the user to inspect clusters found in the workload. To showcase the generality of TracEx, we will guide the user through two different scenarios:

DBMS Operators. Figure 8 shows the individual cluster view of the join-heavy cluster from our running example. The smoothed z-score label indicates the importance of the hash-join, as the operator share is 62%, and the negative influence of the scan operator specific to this cluster. An in-depth visualization is shown on the right side of the figure, comparing the cluster's properties to those of the whole dataset. We can conclude that the hash-join is the deciding differentiating factor for this cluster. Returning to our running example, this suggests that engineers trying to identify future investments should focus on improving the hash-join here.

System Resources. In addition to query plan statistics, TracEx can be used to understand a workload's resource utilization. In our second scenario, we examine a cluster that has a strongly increased fraction of bytes written to S3 compared to the remaining

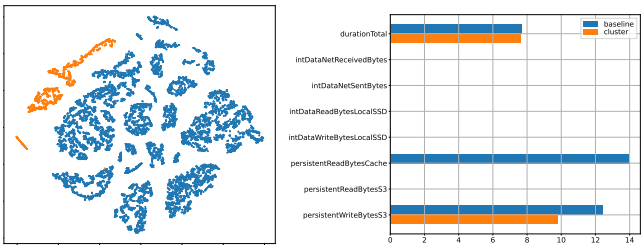


Figure 9: Write-heavy cluster of Snowset restricted to log-scaled resource features.

dataset. Figure 9 shows how this type of analysis can help optimize infrastructure for the user's workload.

Step 3b: Data Visualization - Workload Comparison. In this part of the demonstration, we want the user to compare different workloads. Figure 7 shows an example scenario when interacting with our web UI which allows the user to select the workloads as well as which attributes to compare. We will preload several different workloads, allowing users to mimic the comparison between Snowset and the TPC benchmarks, for example, as shown in Figure 2.

Overall, we envision an interactive demonstration that, after the initial walkthrough, will allow the users to switch between different parts of TracEx, for example refining workloads as described in Step 2 and then visualizing them using one of the provided techniques of the data visualization layer. The walkthrough is designed to cover the feature space of TracEx, but since the implementation provides a user interface with preloaded workloads and IRs, we will be able to dynamically adjust the demonstration scenario depending on user preferences.

5 Related Work

Although capturing the traces of workloads is common in database systems, to the best of our knowledge, TracEx is the first tool that allows to visualize, compare, and contrast workloads. Prior work has focused on tracing itself and its overhead, benchmarking of database systems, and visualizations for monitoring logs.

Visualization. In the DBMS space, visualizations have been used in various ways to facilitate user understanding: Many open-source and commercial software applications visualize log files from various (distributed) systems to monitor the health and performance of systems [11, 16, 19]. Database vendors also provide tools to examine query patterns and performance, such as AWS RDS Performance Insights, Snowflake Query Profile, or Azure Query Performance Insights [2, 18, 25]. These tools focus on individual query statistics or system-wide resource utilization. TracEx is an additional tool that helps understand query clusters within your workload, their characteristics, and allows users to compare different workloads. Unlike database-vendor-provided visualization tools, TracEx relies solely on privacy-preserving traces, allowing data to be shared across multiple entities. Similar to these tools, flame graphs help understand the resource utilization of different aspects of the database, typically at a per-query or system-wide granularity.

Prior research has found that enabling users to interactively explore data sets is important to facilitate their understanding of the properties of their data [15, 24]. In TracEx, we complement these approaches by using traces for interactive exploration using techniques such as t-SNE, which has been shown to work well for creating vector embeddings of queries [12].

General-purpose data visualization tools focus on various algorithms for representing the data. These tools often require stronger visualization knowledge (algorithms, etc.) and do not guide the user to useful results. TracEx's selection of easy-to-use tools helps to understand the key components (clusters) of the workload and allows for quick comparison of different data sets.

Database Tracing. Traces have been an object of interest for companies like Oracle, [20], or Microsoft, [17], which have focused on accurately capturing traces, replaying them, and helping users understand them as early as 1997. Most of these traces assume full access to the query and data space, thus allowing the DBMS provider to collect more detailed query information to replay the query. To resolve privacy issues, more recent work has furthermore added obfuscation of the resulting data [32]. In our setting, we assume that software-as-a-service database vendors collect anonymized query statistics of their customers, which is a common practice for cloud DBMS vendors. These statistics do not contain query-specific information but can be used to explore usage patterns of the users. One such example is the Snowset trace [29], provided by Snowflake, which we explore with TracEx as an instance of privacy-preserving traces.

Benchmarking. A core objective in benchmarking a DBMS-under-test is to evaluate the system under conditions that are as realistic as possible while at the same time avoiding the expense of replaying a whole workload. To address this challenge, prior work has focused on developing strategies for efficient benchmarking with a given set of workloads [10, 13], or automatically generating test workloads [14], but a core challenge remains that these workloads fail to represent real-world characteristics [4, 28]. With TracEx, we enhance a user's understanding of their *workload coverage* which can then be used to positively impact their benchmarking efforts by focusing more on the most important tests.

6 Conclusion

In this paper, we presented TracEx, an open-source **Trac**e **Ex**ploration tool that facilitates workload trace analysis and comparison for any database system that can collect traces. This allows users to understand their workload by providing an intuitive, visual interface that explores the workload along different dimensions, e.g., resource utilization or database operator usage. Additionally, users are able to contrast and compare workloads that have been collected from different hardware configurations or even compare traces between database systems.

Acknowledgments

We would like to thank Shaleen Deep for his helpful comments on the manuscript. Dominik Durner has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 725286).

References

[1] Josep Aguilar-Saborit and Raghu Ramakrishnan. 2020. POLARIS: The Distributed SQL Engine in Azure Synapse. *Proc. VLDB Endow.* 13, 12 (2020), 3204–3216.

[2] Amazon. 2023. Analyze and tune Amazon RDS database performance. https://aws.amazon.com/rds/performance-insights/. accessed: 2023-12-01.

[3] Benoît Dageville, Thierry Cruanes, Marcin Zukowski, Vadim Antonov, Artin Avanes, Jon Bock, Jonathan Claybaugh, Daniel Engovatov, Martin Hentschel, Jiansheng Huang, Allison W. Lee, Ashish Motivala, Abdul Q. Munir, Steven Pelley, Peter Povinec, Greg Rahn, Spyridon Triantafyllis, and Philipp Unterbrunner. 2016. The Snowflake Elastic Data Warehouse. In *SIGMOD Conference*. ACM, 215–226.

[4] Shaleen Deep, Anja Gruenheid, Paraschos Koutris, Jeffrey F. Naughton, and Stratis Viglas. 2020. Comprehensive and Efficient Workload Compression. *Proc. VLDB Endow.* 14, 3 (2020).

[5] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *KDD*. AAAI Press, 226–231.

[6] FlowJo. 2017. FlowJo tSNE. http://v9docs.flowjo.com/html/tsne.html. accessed: 2023-12-01.

[7] The Apache Foundation. 2023. *Apache Parquet*. https://parquet.apache.org/ accessed: 2023-12-01.

[8] Aurélien Géron. 2019. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, Inc.

[9] Anurag Gupta, Deepak Agarwal, Derek Tan, Jakub Kulesza, Rahul Pathak, Stefano Stefani, and Vidhya Srinivasan. 2015. Amazon Redshift and the Case for Simpler Data Warehouses. In *SIGMOD Conference*. ACM, 1917–1923.

[10] Florian Haftmann, Donald Kossmann, and Eric Lo. 2007. A framework for efficient regression tests on database applications. *VLDB J.* 16, 1 (2007), 145–164.

[11] Splunk Inc. 2023. Splunk: The Key to Enterprise Resilience. https://splunk.com/. accessed: 2023-07-20.

[12] Shrainik Jain and Bill Howe. 2018. Query2Vec: NLP Meets Databases for Generalized Workload Analytics. *CoRR* abs/1801.05613 (2018).

[13] Jinho Jung, Hong Hu, Joy Arulraj, Taesoo Kim, and Woon-Hak Kang. 2019. APOLLO: Automatic Detection and Diagnosis of Performance Regressions in Database Systems. *Proc. VLDB Endow.* 13, 1 (2019), 57–70.

[14] Martin L. Kersten, Stefan Manegold, Ying Zhang, and Panos Kuoutsourakis. 2019. SQALPEL: A database performance platform. In *CIDR*.

[15] Tim Kraska. 2018. Northstar: An Interactive Data Science System. *Proc. VLDB Endow.* 11, 12 (2018), 2150–2164.

[16] Grafana Labs. 2023. Grafana: The open observability platform. https://grafana.com/. accessed: 2023-07-20.

[17] Microsoft. 2022. Overview of the workload comparison process. https://learn.microsoft.com/en-us/sql/dea/database-experimentation-assistant-get-started?view=sql-server-ver16. accessed: 2022-10-25.

[18] Microsoft. 2023. Query Performance Insight. https://learn.microsoft.com/en-us/azure/postgresql/single-server/concepts-query-performance-insight. accessed: 2023-12-01.

[19] OpenTracing. 2023. The OpenTracing project. https://opentracing.io/. accessed: 2023-07-20.

[20] Oracle. 1997. Managing Workloads. https://docs.oracle.com/cd/A57673_01/DOC/sysman/doc/A55907_01/trace.htm. accessed: 2022-10-25.

[21] Mark Raasveldt and Hannes Mühleisen. 2019. DuckDB: an Embeddable Analytical Database. In *SIGMOD Conference*. ACM, 1981–1984.

[22] scikit-learn developers. 2023. AgglomerativeClustering. https://scikit-learn.org/stable/modules/generated/sklearn.cluster.AgglomerativeClustering.html. accessed: 2023-12-01.

[23] scikit-learn developers. 2023. sklearn.manifold.TSNE. https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html. accessed: 2023-12-01.

[24] Tarique Siddiqui, Paul Luh, Zesheng Wang, Karrie Karahalios, and Aditya G. Parameswaran. 2022. Expressive querying for accelerating visual analytics. *Commun. ACM* 65, 7 (2022), 85–94.

[25] Snowflake. 2023. Analyzing Queries Using Query Profile. https://docs.snowflake.com/en/user-guide/ui-query-profile. accessed: 2023-12-01.

[26] The OpenTelemetry Authors. 2023. OpenTelemetry. https://opentelemetry.io. accessed: 2023-12-01.

[27] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *JMLR* 9, 11 (2008).

[28] Adrian Vogelsgesang, Michael Haubenschild, Jan Finis, Alfons Kemper, Viktor Leis, Tobias Mühlbauer, Thomas Neumann, and Manuel Then. 2018. Get Real: How Benchmarks Fail to Represent the Real World. In *DBTest@SIGMOD*. ACM, 1:1–1:6.

[29] Midhul Vuppalapati, Justin Miron, Rachit Agarwal, Dan Truong, Ashish Motivala, and Thierry Cruanes. 2020. Building An Elastic Query Engine on Disaggregated Storage. In *NSDI*. USENIX Association, 449–462.

[30] Martin Wattenberg, Fernanda Viégas, and Ian Johnson. 2016. How to use t-SNE effectively. *Distill* 1, 10 (2016), e2.

[31] Svante Wold, Kim Esbensen, and Paul Geladi. 1987. Principal component analysis. *Chemometrics and Intelligent Laboratory Systems* 2, 1-3 (1987), 37–52.

[32] Jiaqi Yan, Qiuye Jin, Shrainik Jain, Stratis D. Viglas, and Allison W. Lee. 2018. Snowtrail: Testing with Production Queries on a Cloud Database. In *DBTest@SIGMOD*. ACM, 4:1–4:6.