

Database Benchmarking for Supporting Real-Time Interactive Querying of Large Data

Leilani Battle, Philipp Eichmann, Marco Angelini, Tiziana Catarci, Giuseppe Santucci, Yukun Zheng, Carsten Binnig, Jean-Daniel Fekete, Dominik Moritz

▶ **To cite this version:**

Leilani Battle, Philipp Eichmann, Marco Angelini, Tiziana Catarci, Giuseppe Santucci, et al.. Database Benchmarking for Supporting Real-Time Interactive Querying of Large Data. SIGMOD '20 - International Conference on Management of Data, Jun 2020, Portland, OR, United States. pp.1571-1587, 10.1145/3318464.3389732 . hal-02556400

HAL Id: hal-02556400
https://inria.hal.science/hal-02556400

Submitted on 28 Apr 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Database Benchmarking for Supporting Real-Time Interactive Querying of Large Data

Leilani Battle
University of Maryland*
College Park, Maryland
leilani@cs.umd.edu

Philipp Eichmann
Brown University
Providence, Rhode Island
philipp_eichmann@brown.edu

Marco Angelini
University of Rome "La Sapienza"**
Rome, Italy
angelini@diag.uniroma1.it

Tiziana Catarci
University of Rome "La Sapienza"**
catarci@diag.uniroma1.it

Giuseppe Santucci
University of Rome "La Sapienza"**
santucci@diag.uniroma1.it

Yukun Zheng
University of Maryland*
yzheng11@umd.edu

Carsten Binnig
Technical University of Darmstadt
Darmstadt, Germany
carsten.binnig@cs.tu-darmstadt.de

Jean-Daniel Fekete
Inria, Univ. Paris-Saclay, CNRS
Orsay, France
jean-daniel.fekete@inria.fr

Dominik Moritz
University of Washington
Seattle, Washington
domoritz@cs.washington.edu

ABSTRACT

In this paper, we present a new benchmark to validate the suitability of database systems for interactive visualization workloads. While there exist proposals for evaluating database systems on interactive data exploration workloads, none rely on real user traces for database benchmarking. To this end, our long term goal is to collect user traces that represent workloads with different exploration characteristics. In this paper, we present an initial benchmark that focuses on "crossfilter"-style applications, which are a popular interaction type for data exploration and a particularly demanding scenario for testing database system performance. We make our benchmark materials, including input datasets, interaction sequences, corresponding SQL queries, and analysis code, freely available as a community resource, to foster further research in this area: https://osf.io/9xerb/?view_only=81de1a3f99d04529b6b173a3bd5b4d23.

CCS CONCEPTS

• **Information systems** → **Data management systems**; **Data analytics**; • **Human-centered computing** → **Visualization systems and tools**.

1 INTRODUCTION

Motivation. The data science process often begins with users (i.e., analysts, data scientists) exploring possibly massive amounts of data through interactions with a graphical user interface, oftentimes a data visualization tool [4, 9, 12, 66]. However, each time a user interacts with a visualization interface, the underlying data must be processed (filtered, aggregated, etc.) such that the interface can *quickly provide a visual response* [39, 42, 56]. To meet this growing demand for *interactive* and *real-time* performance, the database and visualization communities have developed a variety of techniques, including approximate query processing [2, 14], online aggregation/progressive visualization [3, 19, 26], data cubes [8, 36, 38], spatial indexing [63], speculative query execution [8, 30], and lineage tracking [53].

However, we still lack adequate benchmarks to empirically assess which of these resulting systems provide satisfactory performance, and which systems are actually better than others for interactive, real-time querying scenarios. This issue is exacerbated for the most demanding yet popular visualization scenarios such as crossfilter [41, 58, 67], where one interaction may generate hundreds of queries per second, with an expectation of near-immediate results. Unfortunately, existing database benchmarks such as TPC-H [65], TPC-DS [64], or the Star Schema Benchmark (SSB) [47] are insufficient for making these comparisons. One main reason is that the workloads modeled in these benchmarks are not representative of *how* database queries are generated through user interactions, such as with tools like Tableau [59] or Spotfire [58]. In contrast, visualization benchmarks provide realistic scenarios, but lack the configurability, precision and automation afforded by database benchmarks when testing performance [7]. Furthermore, there exists no evaluation platform to test database systems under a range of interactive analysis conditions, such as different dataset characteristics, exploration scenarios, interaction paradigms, or user profiles. Therefore, our community is still woefully unequipped to answer the question "are database systems truly capable of supporting *real-time* interactive data exploration at scale?"

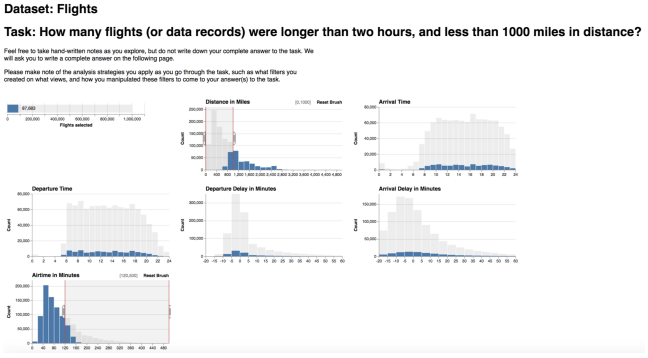


Figure 1: An example crossfilter application from our user study, visualizing the Flights dataset, composed of six numerical attributes. Each attribute is visualized as a histogram that can be filtered using a range-slider. When the user interacts with a range slider, the entire dataset is filtered, the bins in each histogram are recomputed, and the visualizations are updated.

Here, real-time means with a latency < 100 ms, or supporting at least ten frames per second [15, 41, 42]. Unfortunately, our study reveals more work is needed from the database community to answer positively.

Contributions. While there exist proposals [7, 18, 28, 29, 61] for evaluating database systems on interactive data exploration workloads, none of these approaches derive a benchmark from real user traces, introducing potential artifacts and possibly missing key behavioral patterns. To validate the suitability of database systems for interactive and visual data exploration workloads, this paper thus presents a benchmark constructed from real traces of users interacting with an information visualization system. In this scenario, every query corresponds to an actual interaction from a real user.

Given the innumerable analysis scenarios supported by visualization tools [23, 33], it would be unrealistic to have one benchmark represent all possible use cases. Instead, **we believe that a database benchmark ecosystem is required for evaluating interactive data exploration workloads**, similar to the TPC benchmarks for classical database workloads (OLTP, OLAP, etc.). Our long term goal is to collect user traces, representing workloads with different exploration characteristics. As a first step, we mainly use "crossfilter"-style applications to collect user traces and derive our benchmark (see Figure 1). The benchmark development process presented in this paper provides a blueprint for creating more interaction-focused benchmarks in the future.

Given how easy and intuitive they are to use when exploring complex datasets, Crossfilter interfaces are pervasive in information visualization tools including foundational systems like the Attribute Explorer [67] and Spotfire [58] to recent systems like Falcon [41]. Moreover, compared to other interaction designs (e.g., mouse hovers or check-box selection), **crossfilter-style applications are arguably the most demanding use case for database systems [41].** With one single interaction, a user can generate hundreds of database queries per second when moving a single crossfilter range-slider. For example, Figure 1 shows information about airline delays; changing the selection of the flight distance using the crossfilter range-slider of the histogram in the upper row (center), causes all other histograms to update at each mouse drag. Crossfilter interfaces are instances of *dynamic query* interfaces [57], which require live updates from the underlying database system as the slider moves in real time.

In the following, we discuss the contributions of this paper:

As a first contribution, we present the results of a user study conducted by the different international research groups who authored this article. In this user study, we asked 22 users with a broad range of data science experience to perform four representative interactive analysis tasks using a crossfilter visualization tool, across three different datasets, producing a total of 128 different traces for further study. Using the log data and video recordings collected from the study, we analyze and summarize our findings regarding the interaction patterns observed, and their effects on the resulting query workload. This characterization analysis provides useful insights into how interaction patterns can be taken into account when evaluating and optimizing queries on a database system.

As a second contribution, we present the results of our analysis of the 128 collected traces and characterize the behavior of typical users on interactive tasks. Although continuous interactions, such as dragging a slider, require a latency < 100 ms to provide continuous feedback, other events such as mouse clicks only require a 1 s latency, allowing for pre-computation strategies. Based on our analysis, we elaborate on design implications for database systems to support highly interactive use cases (e.g., crossfilter).

As a third contribution, we present a new benchmark based on our user traces. As mentioned before this benchmark aims to measure the ability of a DBMS to sustain the challenging workloads of interactive and real-time data exploration applications. Contrary to traditional database benchmarks focused on returning exact results without latency bounds, our benchmark focuses on other metrics. For example, when interactively dragging a slider, latency bounds are truly important whereas users typically tolerate approximate results or even some query results can be ignored to avoid accumulating lag (known as *debouncing*). We report the results of running our benchmark on five DBMSs, including MonetDB [10], VerdictDB [49] and DuckDB [55]. Surprisingly, we find that *none* of the DBMSs tested provide sufficient performance when visualizing 10M rows or more.

As a final contribution, we have made all of our study traces, benchmark code, and analysis code available online as a community resource to support future research.[1] As observed in prior work (e.g., [9, 20, 31, 40, 52]), experiment data is a highly-valuable source of insights not only for the database community but for other communities as well. Hence, we see this as an important step towards collaboration between the database and visualization communities.

2 BACKGROUND

In this section, we summarize key topics related to supporting and evaluating interactive exploration of large datasets.

2.1 Defining Interactive Data Exploration

Interactive data exploration has a specific set of characteristics that must be considered when evaluating performance. However, these characteristics are not well-defined in the database literature. Here, **we provide a concrete definition for interactive data exploration, and define the class of visualization interfaces (aka dynamic query interfaces) that we study in this paper**.

Interactive data exploration is considered in some ways synonymous with *information visualization* [9, 66], a core tenet of which is to facilitate "interactive, visual representations of abstract data to amplify cognition" [12, p. 7]. Thus the manipulation of visualizations through interactions is considered necessary to uncovering useful insights [4, 50]. To support interactions, state of the art visualization tools aim to provide *dynamic query interfaces* [57] that allow to quickly formulate new queries by interacting with a visual representation to manipulate the query. In general, dynamic query interfaces are implemented using interactive objects that appear on visualization applications either as widgets next to the main visualization area (e.g., sliders, radio buttons), or more deeply integrated with the visualization (e.g., brush filters, drag-based panning). To summarize, we define interactive data exploration as:

> *the process of extracting insights from data by creating and manipulating visualizations using dynamic query widgets that update in real-time.*

Interactive data exploration is useful when the user does not have a clear goal or question in mind and, as a consequence, cannot rely on a single query; typically, the user issues a series of queries to understand the data structure, discover insights, and pursue more specific investigations.

In this paper, we study crossfilter interfaces, since they are known to be among the most demanding use cases for DBMSs in terms of latency, number of user interactions, and resulting number of generated queries to the database [41]. We describe the interface used for this paper in §3.

[1] https://osf.io/9xerb/?view_only=81de1a3f99d04529b6b173a3bd5b4d23

2.2 Performance Considerations

In this section, **we highlight the primary performance concerns for interactive data exploration scenarios, and explain how these scenarios are fundamentally different from existing database evaluation contexts.**

2.2.1 Comparing with Existing Benchmarks. Interactive data exploration, particularly dynamic query approaches such as crossfilter-applications, stand out compared to traditional workloads in two ways. First, dynamic query approaches update visualizations *in real-time*, potentially generating hundreds of interaction events (and thus hundreds of queries) per second. Therefore, this use case emphasizes the need for *real-time results*. Unlike in traditional database scenarios (e.g., TPC-H [65], TPC-DS [64], or the Star Schema Benchmark (SSB) [47]), latencies of just one second can make interactive data exploration interfaces appear unresponsive [37].

Second, the significance of an interactive exploration workload is not the exact set of queries involved (which are simply a subset of OLAP), but the speed, cadence and somewhat unpredictable nature of the interactions observed. In this case, the DBMS is not periodically updating a fixed dashboard (i.e., traditional OLAP contexts), it is interacting in real time with *real people*, who often explore data in an ad-hoc manner. As a result, the characteristics of this type of workload are drastically different from workloads from other database scenarios (e.g., data warehousing). In addition, the metrics emphasized by these benchmarks fail to capture key performance considerations for interactive exploration contexts [7], such as whether real-time latency constraints are being violated, or whether approximate results are of sufficient quality to support effective data exploration.

In contrast, visualization benchmarks focus on collecting realistic use cases, where the extracted queries correspond to real user interactions [7, 52]. However, these benchmarks are generally folders of raw data, and lack the necessary details and infrastructure to transform this data into executable workloads for system performance evaluation.

Recent vision papers [7, 17] and tutorials [28, 29, 61] discuss these issues, and propose initial directions on how to design DBMSs to better support real-time interactive workloads (e.g., [17]). However, to the best of our knowledge, **this paper is the first to derive a database benchmark completely based on real traces of user interactions.** Our benchmark also provides a blueprint for deriving new benchmarks in the future, with the ultimate goal of forming an ecosystem of benchmarks to support a range of interactive analysis scenarios. Our design emphasizes two dimensions:

- **Ad-hoc workload**: how can we capture the flow and cadence of queries that users generate while performing dynamic query interactions?
- **System Latency**: how can we measure the ability of the DBMS to respond quickly during dynamic queries

to maintain the user's attention and help the user detect complex patterns in real time [57]?

2.2.2 Interactive Latency Thresholds. Several studies show that latency directly impacts user behavior [39, 56], and can negatively affect user analysis performance [37, 72]. Depending on the nature of the task, there are different kinds of latencies to consider [42]. When the user needs to perceive a visualization as a continuous "animation" (e.g., the result of continuously changing a slider), the latency requirement is less than 0.1s (*type1* latency). When the user just triggers one action (e.g., selecting an option on a user interface), the feedback should appear within 1s (*type2*). Finally, after issuing a command (e.g., writing a SQL query), the user will expect an answer within about 10s (*type3*). These "powers of ten" rules have been validated empirically [42] for interactive systems.

Type2 latency is often used as a threshold to evaluate data exploration systems such as Tableau [60], Spotfire [58], and Kyrix [63]. However, these applications do not challenge modern databases as much. Others argue that *type1* latencies need to be supported for dynamic queries [15, 41]. However, *type1* latency places an exceptionally heavy load on the underlying DBMS, and it is still unclear whether this level of latency support is truly necessary. **In this paper, we provide much-needed context for which latency levels are truly needed to support interactive data exploration** by collecting real user interaction logs, and testing how well current DBMSs support dynamic query workloads.

2.3 Optimizations for Interactive Contexts

In the following, **we summarize the strategies that have been devised across the HCI, visualization and database communities to sustain dynamic query workloads**.

Altering Perception of Latency. VizDom [15] provides progressive updates to brushing histograms to support *type2* latency. Li et al. [35] evaluated brushing histograms with click-based selections of histogram bars, supporting *type2* latency. However, in a controlled study comparing dynamic queries with brushing histograms, they show that some tasks are better achieved with sliders (i.e., with *type1* latency).

Pre-computing Results. Pre-computing histograms has been explored by Tanin et al. [62] to support *type1* latency. They argue that the number of required bin calculations is bounded by the number of pixels in the visual output, and pre-compute histogram updates accordingly. Moritz et al. [41] extend this idea by devising a histogram computation strategy that enables computing the results of all possible interactions in nearly the same time as for a single interaction. They use this technique to pre-compute data for all possible interactions with a single view, and re-computes when the user switches to a different view. All interaction requests are served using the pre-computed index, similar to other visualization systems that pre-compute similar data structures (e.g., data cubes [8, 36, 38]). However, pre-computation can be costly in both storage and offline computation time [7].

Modulating Event Rates. Throttling is implemented at the low-level of the interaction loop to avoid accumulating lag [43, p. 101]; it limits the number of events handled by the application. Current pointer devices generate 200 events per second, but the refresh rate of a screen is around 60-100Hz, so there is no need to process events faster than the refresh rate.

Debouncing is a related technique, where the visualization tool may drop any new interaction events, until the DBMS has finished processing the current batch of queries.

Early termination is another related technique that aborts ongoing computations as they become irrelevant (i.e., as they are supplanted by new interaction events) [51].

Approximate Computation. Many systems recognize that in exploratory contexts, users can navigate the data just as effectively using approximate results as with exact results. To this end, several DBMSs support *bounded queries*, which aim to answer queries within a specified amount of time by trading accuracy for response time [2]. A different approach, known by several names, such as *anytime query answering* [44], *online aggregation* [25, 26], and *progressive analytics* [5, 19], provides an approximate query result that improves over time, and allows the user to control the execution of this query (e.g., stop or steer the query) at any time before its completion. In similar spirit, several systems aim to reduce latency using offline approximation techniques, such as pre-computation of approximate aggregate values, typically in the OLAP context [48] or using different kinds of sampling [13]. However, what is gained in speed is lost in accuracy, which may not work for all exploration scenarios.

Our benchmark enables direct comparison of these techniques in a realistic yet controlled environment.

3 USER STUDY DESIGN

This section describes methodology used for designing our user study and collecting interaction traces. Our study design is based directly on prior studies of exploratory data analysis contexts [8, 37, 72], such as the study conducted by Battle & Heer to analyze exploration behavior with Tableau [9], and the studies by Liu & Heer [37] and Zgraggen et al. [72] to assess the effects of latency in exploratory data analysis.

3.1 Definitions and Terminology

This subsection defines key HCI and visualization terms related to the user study, summarized from prior work (e.g., [6, 9, 24, 34]): A *goal* is a high level concept, representing an end state or result that the user wishes to achieve. To accomplish a *goal*, the user executes a set of *tasks*, which may include solving a specific problem or selecting among different alternatives. A *task* is composed of a set of structured *actions*, representing the low level *interactions* of the user

Task ID	Task Prompt
T1	How many flights were longer than four and less than six hours?
T2	Which two-hour window (during time of day) contains more flights with longer arrival delays?
T3	Which factors appear to have the greatest effect on the length of departure delays?
T4	How do distance, departure delays, and both distance and departure delays together appear to affect arrival delays?

Table 1: List of all task prompts used in the study for the Flights dataset.

Task	Complexity Level	Exploration Level
T1	low	low
T2	low	low
T3	high	high
T4	high	low

Table 2: Complexity and exploration levels per task.

with the system. *Interactions* are executed through *gestures*, such as using a mouse or fingers on a touch screen.

In this paper, we study a specific type of dynamic query interface known as a crossfilter interface [41, 67]. As shown in Figure 1, a crossfilter application takes a database table, and visualizes each numerical attribute as a separate histogram.

Each histogram is overlaid with a *range slider* that filters the underlying tuples and triggers a re-rendering of the histograms; an active range selection over a given attribute is depicted in the crossfilter interface as a highlighted area in the corresponding histogram. These range filters are often referred to as a *brush* or *brush filter* throughout this paper. Ranges can be created or adjusted using left or right drag actions. Each range filter can also be removed by clicking the `Reset Brush` button in the corresponding histogram.

3.2 Datasets & Exploration Environments

For the study, we developed separate crossfilter-style exploration environments following known design strategies for cross-filtered views [69]. To ensure that our design is reasonably realistic, yet sufficiently controlled for a lab study, we worked directly with the Falcon team to align our design with the original goals for the Falcon system [41]. Note that crossfilter-applications generally involve only a small number of attributes [17, 41]. Typically, visualization datasets have less than 10 attributes, as reported in [27, Figure 2].

We used three real-world data sets selected across multiple domains, where each dataset was also selected for its use in evaluating previous systems: **(1) Flights (6 attributes)**: Performance metrics for flights tracked by the FAA [9] (example in Figure 1). **(2) Movies (8 attributes)**: Performance metrics for various films [70]. **(3) Weather (8 attributes)**: Daily weather measures for stations across the USA [9].

3.3 Study Participants

To collect traces, we recruited participants at three different universities, asked them to perform a series of realistic tasks, and recorded participants' interactions. Participants' experience ranged from undergraduate students across academic disciplines (e.g., computer science, engineering, business) developing their data science skills, to analysts and programmers returning briefly to academia to earn masters degrees, to advanced researchers with PhDs who incorporate data science practices in their everyday work. All participants had prior experience with data analysis tools or visualization tools. Each participant completed the study in roughly two hours, and received $20 compensation for their participation.

3.4 Study Protocol

Each participant first completed a demographic survey to collect general information (e.g., age, gender, etc.) and data science experience. As usual for human studies of this kind, participants were briefed on the functionality of the visualization interface, and provided with a 5-minute warm-up period to familiarize themselves with the interface, similar to prior work [37, 72]. The remainder of the study was then conducted in two analysis blocks. One dataset is analyzed per analysis block, and each block has a five-minute warmup with the dataset and five analysis tasks for participants to complete (one training task and four "real" tasks, see Table 1 for examples). Participants were asked to complete a survey after each task, where they provide their answer to the task and some additional information about their experience and understanding of the current task.

3.5 Task Design

Our task design is based on prior studies of exploratory data analysis [8, 9, 37, 70–72], and controls for the degree of complexity and exploration required to complete tasks (see Table 2). We incorporate existing measures for task complexity [9], which focus on: total attributes explored, the complexity of the data relationships analyzed, and task type (e.g., data assessment versus causality analysis). Exploration level is defined in prior work by the number of unique combinations of attributes explored to complete the task [9, 70, 71].

The tasks are further organized in 3 main classes. These classes are based on complexity and exploration levels, which should generally produce an increase in total operations (or interactions) executed by participants to solve a task, and thus result in a higher number of queries issued to the underlying database system. These classes are:

Quantitative Tasks. a quantitative task asks the user to retrieve a subset of data that respects specific quantitative conditions, made explicit in the text of the task. To solve this task, the user must apply the appropriate filters on the attributes of interest and then provide an answer. An example, relative to the Flights dataset, is "How many flights (or

data records) were longer than two hours, and less than 1000 miles in distance?" These tasks are of low complexity and low exploration (task T1 in our study).

Qualitative Tasks. a qualitative task requires the user to provide considerations about a specific aspect of the dataset, like the identification of changes with respect to multiple attributes of analysis, with usually only a broad indication from the task text (in contrast to the well defined conditions expressed for quantitative tasks). An example, relative to the Movies dataset, is "What changes, if any, do you observe in US DVD sales over time?" These tasks have varied complexity, but generally low exploration (tasks T2 and T4).

Exploratory Tasks. an exploratory task requires the user to explore all the dimensions of the dataset under analysis and the relations that could potentially exists among them in order to solve the task. An example, relative to the Weather dataset, is "Which data attributes appear to have the greatest effect on precipitation?" These tasks generally have high complexity and high exploration (task T3 in our study).

Tasks were administered in ascending order of complexity, such that participants complete the most difficult tasks last. This ordering was chosen through pilot tests of the study, where we found that participant performance (i.e., accuracy and speed) decreased when tasks were provided in a random order. None of the tasks were timed, but the tasks were designed to take 15 minutes or less to complete.

3.6 Data Collection

Each user session was video recorded (both with screen capture and camera recording) and logged through the visual environment, implementing a multi-trace analysis model well described in previous work [6, 9]. Participants' interactions are recorded in 3 main resources: application logs, containing information about the task itself (userID, taskID, dataset confidence, task confidence, answer, answer confidence, eventual comment); view logs, containing information about the configuration of the visualization environment for the given dataset; finally detailed log entries of participants' interactions with the visual environment during a task, collecting information like type of interaction (e.g., mousemove, brush filter, mouseout), timestamp, attribute on which the interaction happens, and mouse coordinates.

Each combination of participant, dataset, and task represents a separate analysis trace, resulting in 128 different traces: 44 for Flights, 36 for Movies, and 48 for Weather. These traces constitute an interaction baseline from which we reason about user behavior and performance (in §4), as well as database system performance (in §5). We share our traces as a community resource, so benchmark users can leverage this data in the future (e.g., for building new interaction models).

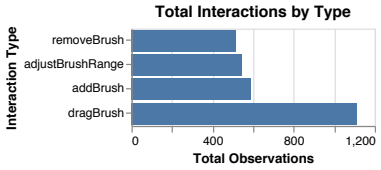


Figure 2: Total observations per interaction type (counting *interactions*, not interaction *events*).

4 CHARACTERIZING TRACES

Our goal is to develop a benchmark for database management systems, where we can execute real sequences of user interactions, and map them to a query workload for reproducible performance evaluation. As such, it is not the queries that we need to understand, but the interactions themselves. Therefore, to fully grasp the implications of the benchmark, we need to understand how user behavior and user performance ultimately impact system performance. To achieve this, we analyze the 128 collected traces along two key sets of metrics emphasized in the visualization and database literature:

(1) **Gestures & Filter Speeds [29, 41]:** to understand how users' physical movements influence interaction speeds and ultimately query generation rates, e.g., do users create or manipulate filters quickly or slowly?
(2) **Interaction Rates & Pacing [9, 21, 37]:** to understand the cadence of user analysis over time, e.g., do users perform interactions in rapid succession, or do we observe frequent pauses between interactions?

4.1 Gestures & Filter Speeds

The goal of this analysis is to provide insight into key characteristics and patterns found in real interaction sequences for crossfilter interfaces, across a range of tasks and datasets.

4.1.1 Interaction Types and Query Generation Rates. Figure 2 shows the occurrence of each interaction type in our user study. We see that drag brush interactions represent most interactions in the dataset (`dragBrush`), followed by adding a new brush (`addBrush`), and then adjusting existing brush ranges (`adjustBrushRange`) and removing brushes (`removeBrush`). Note that the records in this figure refer to interactions, and each interaction consists of multiple events that trigger an update to the connected visualizations (and thus multiple queries to the underlying DBMS).

Unlike interfaces that focus on click-based interactions (e.g., clicking on a buttons, menus, etc.), crossfilter interfaces allow users to perform long, continuous interactions by holding down the left button of the mouse. This interaction paradigm can lead to the user dragging brush filters in one direction, and then dragging them in the opposite direction, producing multiple "gestures" within a single interaction.

Crossfilter interactions are interesting because on average, they tend to generate queries at rates much faster than one query per second, which is significantly higher than other

Type	Avg. Total Queries	Avg. Duration (s)	Avg. Query Generation Rate (queries/s)
addBrush	40.3	1.4	35.8
adjustBrush--Range	47.8	1.8	29.1
dragBrush	174.3	9.3	25.0
removeBrush	1.0	-	-

Table 3: Average duration (in seconds) and total queries generated per interaction type. Average query generation rate (queries per second) is also calculated as total queries divided by duration.

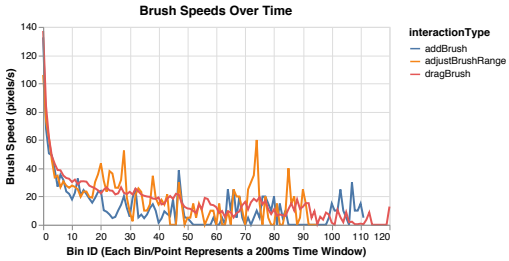


Figure 3: Average brush speed (in pixels/s) is calculated using windowed aggregation, where each window (or bin) represents a non-overlapping duration of 200 ms. Each bin (i.e., 200ms window) is a single point in the line chart. Insignificant bins in the long tail are hidden to save space (i.e., bin IDs greater than 120, or after a total duration of 24 seconds).

interactive visualization contexts, where roughly one query is produced per interaction (e.g., ForeCache, Kyrix, Tableau, etc.). Note that the reported query generation rates here refer to updating only a single view, thus the true query generation rate needs to be multiplied by the number of views in the interface minus one. For example, a rate of 25 queries per second actually translates to $25 \times (8 - 1) = 175$ total queries per second, if there are 8 views in the crossfilter interface (e.g., as we have for the weather dataset in our user study).

4.1.2 The Impact of Drag Interactions. Drag interactions stand out in how they can lead to more queries being executed on the underlying DBMS. However, these interactions are only meaningful if utilized in a certain way: as a user drags a brush, the connected visualizations update in real time. Thus, the more a user drags an existing brush, the more queries will be generated at a rapid pace.

We counted the number of gestures per interaction, and averaged them across interaction types. Note that as click-based interactions, removeBrush interactions contain zero gestures. We find that dragBrush interactions contain notably more gestures per interaction compared to other interaction types. This is due to specific dragging behavior we observed throughout the study, where participants would deliberately exploit the live update feature of Falcon through continuous interaction, consistent with foundational work on Dynamic Queries [57]. For example, participants would often add a brush filter and then drag it across the entire width of the view, to see how the other views changed as this filter shifted in real time. dragBrush seemed to be the only interaction for which users would pay attention to Falcon's live updates, while users seemed to ignore live updates for the other interactions. **These behaviors suggest that participants not only find the live update feature of Falcon useful, but also critical to an effective analysis session with crossfilter interfaces, providing motivation for optimizing DBMSs to support dynamic queries and live updates**.

4.1.3 Gestures Over Time. For each gesture (i.e., drag motion) within each interaction, we performed windowed aggregation over time and calculated statistics, with a time window of 200 ms. These time windows allow us to measure changes in brush speed during a gesture. For each time window, we calculated the brush speed (in pixels per second) by dividing the total change in pixels by the window size.

We find significant variation in brush speeds over time, where gestures that appear smooth to the human eye actually include frequent oscillations between faster and slower brush speeds. We also find that gestures tend to start with fast brush speeds, then slow down over time, shown in Figure 3. We can summarize this behavior as a "fast-slow" brush pattern, which suggests that individual gestures are likely to produce more queries towards the beginning of the gesture, and fewer queries towards the end. Given that drag-based interactions often include multiple gestures (especially `dragBrush` interactions), these results also suggest that larger patterns of oscillation will occur across entire interactions, producing variable query-generation rates over time.

4.2 Interaction Rates & Pacing

Here, we seek to understand what a "realistic" exploration pace looks like with dynamic queries, which is critical to understanding what resources may be available to DBMSs to anticipate and provision for future interactions.

4.2.1 Think Time Analysis. Battle & Heer measure what they call "think time" or the time between consecutive interactions when analysts explore data in Tableau, and find: "The means are notably high, ranging from 14 to 15 seconds, depending on the task. The median think time is 5-7 seconds, indicating skew" [9]. However, when calculating the same measures, we find that the median think time is actually fairly low for crossfilter use cases, compared to this prior

Dataset	Task	Avg. Think Time (s)	Median Think Time (s)	Avg. Seq. Length	Median Seq. Length	Avg. Total Sequences	Median Total Sequences
Flights	**T1**	4.58	2.18	4.45	3	4.29	3.0
	T2	6.29	2.18	3.41	3	14.50	12.5
	T3	3.81	1.04	3.31	3	22.71	15.0
	T4	5.41	2.23	2.54	2	21.36	21.5
Movies	**T1**	2.97	1.68	2.70	2	5.21	3.5
	T2	3.80	1.67	3.09	4	13.54	10.0
	T3	3.07	1.52	3.30	3	12.36	9.5
	T4	3.82	1.53	2.04	2	20.50	16.5
Weather	**T1**	6.32	3.34	3.36	2	11.31	10.5
	T2	6.19	2.64	3.62	3	10.56	7.5
	T3	2.47	1.40	3.31	3	38.06	33.5
	T4	3.51	1.70	2.16	1	24.44	19.0

Table 4: Average and median time between interactions, or "think time" [9], across dataset and task. Average and median # of interaction sequences (or consecutive interactions with a given attribute) are also reported.

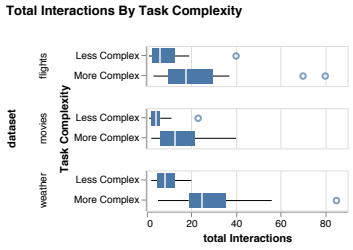


Figure 4: Distribution of total # of interactions according to task complexity, faceted by dataset.

work. Our observed mean think time ranges from 2.47 seconds and 6.32 seconds, and median think time ranges from 1.04 seconds to 3.34 seconds (see columns three and four of Table 4). Though we varied the complexity of our tasks and datasets in a way similar to the task design of Battle & Heer, these results suggest that users may spend less time "thinking" between crossfilter interactions, resulting in shorter gaps between interactions and thus less time for the system to predict and provision for future interactions. These results therefore support our argument that crossfilter use cases may pose a greater performance challenge for DBMSs, compared to other interactive contexts (e.g., Tableau).

The simplicity of the available interactions may influence the differences in think times. As observed in prior work [9, 32], interactions with higher cognitive load may result in longer think times. The more streamlined the interactions (and ultimately the user interface is as a whole), the greater the performance challenge posed for the underlying DBMS, in terms of maintaining interactivity.

Influence of Task Complexity and Open-Endedness. We grouped tasks by complexity, where tasks 1 and 2 are less complex, tasks 3 and 4 more complex (see Table 2). We find that more complex tasks involve performing more interactions, compared to less complex ones, shown in Figure 4. More complex tasks also have shorter think times. To test the relationship between task complexity and think time, we applied linear mixed-effects models, where task is a fixed effect, and participant and dataset are random effects. We find a statistically significant difference between high and low complexity tasks ($\chi^2(1, N = 2764) = 9.1193, p < .05$), where, to our surprise, high complexity tasks tend to have lower think times.

Tasks 3 and 4 are more complex, but 3 is more exploratory (i.e., more open-ended [9]) in nature. We can compare think times between these two tasks to see if think times seem to be different between exploratory and focused tasks. We find that the distribution of think times for task 3 does appear to be slightly lower than task 4, which may suggest that exploratory tasks may be performed at a faster pace than more focused tasks of similar complexity. We applied linear mixed effects models, in this case to assess the relationship between task (3 and 4 only) and think time duration; we find a statistically significant difference ($\chi^2(1, N = 1989) = 4.4859, p < 0.05$), with shorter think times for task 3.

When we analyze the distribution of interaction types per task (see Figure 5), we see a clear emphasis on brush dragging in more focused tasks (Task T4). More exploratory tasks (Task T3) are more varied in usage of interaction types. Thus participants seem to utilize a wider variety of interaction patterns during exploratory tasks, compared to focused ones.

Across all of our results, we find that **exploratory tasks tend to produce more interactions, faster interaction rates, and more variety in interaction usage, compared to simpler and more focused tasks.**

4.2.2 Interaction Sequences. Finally, we analyze the ebb and flow of queries generated over time. Across all participants, we find that many oscillate between performing interactions that generate many queries per second (e.g., `addBrush`, `dragBrush`) and interactions that do not (e.g., `removeBrush`). To further investigate the relationship between the interactions performed and the views they are performed on, we

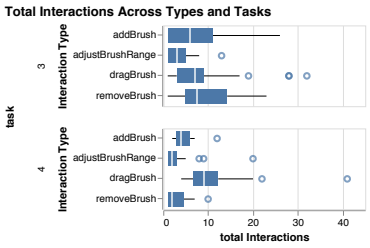


Figure 5: Distribution of total interactions across interaction types, faceted by task (complex tasks only).

analyzed interaction sub-sequences, where consecutive interactions on the same visualization (i.e., with the same data attribute) are grouped together. In general, only a few interactions are performed on any given brush filter before it is removed. Even when the visualization is revisited, a new brush filter is generally added, rather than reusing an existing filter. We calculated the average and median number of consecutive interactions for a given attribute (i.e., sequence length), and found them to be quite short, generally 2–3 interactions per sequence. Long sequences also occur, but somewhat infrequently, with the longest observed sequence involving 16 consecutive interactions. We also calculated the number of times each interaction type appears as the last interaction in a sequence. We find that the `removeBrush` and `dragBrush` interactions are by far the most prevalent interactions observed at the end of a sequence. Furthermore, the `removeBrush` interaction is the most common way that users end a particular sequence of interactions. These results suggest that soon after the user adds a new brush, the system should prepare for the user to remove this brush.

Participants also seem to alternate between different visualizations as they complete more complex tasks. We calculated the mean and median number of times users switch between views. We find that participants routinely switch between visualizations, with many sessions having on average 10 or more switches (see columns seven and eight of Table 4). These results suggest that users frequently interact with multiple attributes to complete analysis tasks, and may revisit the corresponding visualizations multiple times throughout the course of an analysis session.

4.3 Discussion of Results

In this section, we summarize the findings from our characterization analysis, and elaborate on the implications of these results for DBMSs and optimizations.

Crossfilter interfaces encourage patterns of rapid, continuous interaction. We find that exploratory analysis behavior tends to be faster paced and more variable for crossfilter use cases, however these variations follow certain behavioral patterns. For example, we find that users tend to follow a "fast-slow" brush pattern, where we see oscillation between very high and low brush speeds over time, even during individual interactions. Reserving more DBMS resources for the starting phases could improve these fast interactions. Users also tend to switch frequently between multiple views in exploratory tasks, and tend to remove brush filters soon (2–3 interactions) after making them. These results point to a "bursty" query workload, where rather than producing a steady flow of queries, we see oscillation between periods of extremely high query generation rates and low rates, which also span different data attributes over time.

The interaction paradigm for crossfilter UIs leads to high query generation rates. Crossfilter interfaces tend to favor drag-based interactions over click-based ones, which can lead to long, continuous interactions that generate hundreds of queries within seconds, significantly more queries than in other use cases. Users also seem to rely heavily on the drag-based interactions and live updates afforded by crossfilter interfaces, suggesting that this functionality should be supported in future DBMSs for interactive workloads.

Users behave differently when completing tasks with different complexity levels or exploratory levels. We find that more complex data analysis tasks tend to produce more interactions than simpler tasks. More focused tasks generally result in a heavy reliance on drag-interactions, whereas more exploratory tasks have a wider variation in the types of interactions used. Past work argues that users break data exploration goals down into more focused subtasks [8, 9], suggesting that these task-dependent patterns could all be observed at different times within the same exploration session. Thus, designing data processing optimizations that can reason about the user's current sub-task [8] could be a promising direction for crossfilter contexts.

5 BENCHMARKING FROM TRACES

The goal of our benchmark is to assess the performance and accuracy of DBMSs against typical workloads from real-time interactive visualization systems. Our design is based on existing database evaluation methods (e.g., [8, 9, 29, 30, 68]), which often decouple user data collection (i.e., extracting traces) from system evaluation (i.e., measuring DBMS query performance). Note that we do not want to model users' *reactions* to latency in our benchmark, rather the performance of the systems themselves. As done in prior work [8], we aim to illustrate how DBMSs perform in an ideal scenario where users interact naturally to explore the data.

A major challenge of this work is to translate raw interaction logs into usable query workloads that follow natural user behavior. With crossfilter applications, each user interaction may generate hundreds of interaction events per second, where each interaction event triggers a re-rendering of all views in the crossfilter interface. To re-render a single view, an aggregation query is executed, where 20-30 bins are

calculated per query. However, the user still *perceives* this sequence of interaction events as a single crossfilter interaction, and thus expects fast response rates (ideally frame rates of 50fps or higher [38]). This type of workload cannot be found in traditional database benchmarks.

To address this challenge, we constructed a benchmark pipeline consisting of several key components: mapping interaction events to database queries (§5.1 & §5.2), deriving meaningful metrics for query execution performance (§5.3), and selecting and testing a range of systems with the resulting queries and metrics (§5.4 & §5.5). In this section, we explain the details of our benchmark pipeline.

5.1 Setup

At its core, our benchmark consists of a series of *workflows*, where each workflow corresponds to a trace for a specific study participant, dataset and task. For each DBMS that we test, we implement a *driver* to allow workflow scripts to connect to the DBMS.

We execute workflows using a modified version of IDEBench developed in collaboration with the IDEBench authors [17]. IDEBench interprets, schedules, and delegates queries to individual database drivers. The drivers then translate user interactions from their (JSON) specifications to the query language (i.e., the SQL dialect) that is supported by the DBMS. Please see the IDEBench paper for more details [17].

5.2 Translating Workflows to Queries

The log events for each crossfilter interaction in a particular workflow can be mapped to a series of range queries to execute on a given DBMS. Any query language can be supported [17], but due to its popularity we focus on SQL.

Each entry in the workflow (i.e., logged interaction event) is translated into a target database query language using a straightforward approach. We take all of the filter statements from the current entry in the workflow (and all active filters from previous entries), and turn them into a set of filter predicates, and translate the bin calculations to grouping and aggregation statements. Please see [17] for more details. Benchmark users can also specify the workflow *ground truth*, or the expected results and performance of the DBMS.

Though multiple interaction events can be merged into a single group-by operation, Vartak et al. show that merging too many group-by operations can worsen rather than improve performance [68]. We usually have 20–30 bins per histogram, and the resulting combinatorics for merged queries would likely be cost prohibitive. Thus we maintain a one-to-one mapping of events to queries in our benchmark.

5.3 Benchmark Measures

Unlike existing database benchmarks, which expect 100% accurate responses and are mostly concerned with throughput and resource usage (e.g., memory), interactive systems are concerned with latency, and are able to handle some inaccuracy in query results. Therefore, our benchmark measures:

- **Throughput**: number of transactions per second
- **Latency**: time between issuing a request and receiving its response
- **Accuracy**: distance between the exact results and the returned results

Our benchmark records detailed measures for each query q in a workflow script, including completion time $t(q)$, exact results $e(q)$, and actual results $a(q)$. We define a set of primary and secondary measures related to responsiveness and accuracy, calculated using the reports generated by executing the workflow scripts. These reports can then be aggregated across participants to produce the final benchmark measures.

We assume that once bin counts are retrieved for a given query, the rendering and network transfer time is negligible. Therefore, we omit this timing data from our metrics.

5.3.1 Responsiveness. Responsiveness is influenced by response time, or whether the DBMS returns results fast enough for the crossfilter interface to display them within a specified time threshold. Latency thresholds (and latency violations) are measured frequently to evaluate data exploration systems (e.g., [8, 9, 16, 37, 38, 68]). Ideally, DBMSs should be tested against standard frame rates (i.e., 50–60 frames per second), similar to experiments run by Liu & Heer [38]. However, this threshold may be unrealistic for datasets larger than one million points, so we instead set a more modest worst-case threshold of 10 frames per second (or 100 ms), consistent with the type1 latency constraint discussed in §2.2. Note that this threshold is five times lower than most thresholds reported in previous visualization work [8, 16, 53, 63]. As mentioned in §4, live updates are a critical feature for crossfilter use cases, and 500 ms is simply too slow to maintain even our 10 frames per second worst-case threshold.

Furthermore, when events come at a rate faster than the DBMS can handle, it is important to focus on *the most recent events first*, and drop the rest (i.e., debouncing), or the DBMS may become bogged down in backlogged query requests. Thus, we also measure query drop (and cancellation) rates.

We can calculate responsiveness with respect to the number of queries *issued* (i.e., the fraction of responsive queries). Let Q_i represent the set of queries issued. These queries can be either successfully *answered on time* by the DBMS (represented by the set Q_a), *answered too late* (represented by Q_l), *dropped* by the driver (Q_d) to avoid accumulating lag, or *canceled* by the driver because the user terminated the interaction (Q_c). Here late queries mean the DBMS returned results outside the interactivity threshold $t(q) > 100, q \in Q_l$(i.e., a latency violation). Therefore, we have:

$$Q_i = Q_a \cup Q_l \cup Q_d \cup Q_c \quad (1)$$

$$|Q_i| = |Q_a| + |Q_l| + |Q_d| + |Q_c| \quad (2)$$

We further define the following measures, where response rate is our primary measure:

Response Rate The fraction of queries answered on time, out of all queries issued (captures latency violations):

$$\frac{|Q_a|}{|Q_i|} \tag{3}$$

Average Query Duration The average latency of all the queries answered:

$$\frac{\sum_{q \in Q_a} t(i)}{|Q_a|} \tag{4}$$

5.3.2 Accuracy. Approximate systems can produce results faster than "blocking" DBMSs, potentially improving user analysis performance [72]. However, approximate results deviate from the ground-truth, introducing error into the resulting visualizations and possibly misleading users. To capture how much an approximate result deviates from the ground-truth, we infer the accuracy of results from the measured error between an exact and actual result: $\|e(q) - a(q)\|$. This error is the distance between the histograms. To compute it, we use the bins $a(q)_i$ and $e(q)_i$ over m bins:

$$\|e(q) - a(q)\| = \sqrt{\sum_{i \in [1,m]} (e(q)_i - a(q)_i)^2} \tag{5}$$

Mean Relative Error (MRE) We can measure the error of multiple results of an aggregate query and its ground-truth specification by computing the relative error, i.e., the ratio between the absolute error (as in Equation 5) and the magnitude of the exact result:

$$\sum_{q \in Q_a} \frac{\|e(q) - a(q)\|}{\|e(q)\| \cdot |Q_a|} \tag{6}$$

5.3.3 Discussion. A perfect workflow report would have zero latency violations (i.e., $|Q_l| = 0$), 100% accuracy (i.e., average error of 0), and zero missed queries (i.e., $|Q_d| = |Q_c| = 0$), producing a response rate of 100%. **The two main measures to report for our benchmark are the response rate (Equation 3) and the mean relative error for accuracy (Equation 6).** A good-enough database should report 0 for the former, and a minimized number for the latter.

5.4 DBMSs & Drivers

We have implemented an IDEBench driver for each DBMS tested[2]. Upon execution, each driver reads the given workflow script, translates it to queries, sends the queries to the DBMS, retrieves the results, and computes the final quality measures. The driver can be further optimized using the strategies described in §2.3, such as debouncing or aborting long transactions, according to the capability of the DBMS.

Robust commercial systems: In this group, we test two particular DBMSs: MonetDB and PostgreSQL. These systems are commonly used to evaluate a variety of transactional or analytical processing scenarios. Thus these systems are a good starting point to develop a performance baseline.

Lightweight analytics systems: DBMSs like SQLite and DuckDB are designed specifically for easy integration with well-known, programming-focused data science use cases, representing an alternative form of "interactivity" that lies between traditional visualization and database interfaces.

Approximate Systems: Given that accuracy is one of our primary measures, we selected one approximate query engine, VerdictDB for evaluation [49], which can connect to a variety of SQL-based systems. In all of our experiments, we pair VerdictDB with PostgreSQL, and we configure VerdictDB to build a 10% sample for every dataset.

5.5 Running the Benchmark

All of our interaction logs, workflows, driver code, and analysis code are provided as part of our public benchmark. We have provided a Docker image file for easy setup, and two scripts that will execute all of our benchmark workflows and compute our primary measures across all generated workflow reports. These resources, and specific details for running our benchmark are available online (see footnote 1).

6 RESULTS

6.1 Environment Setups

We tested three setups to evaluate the DBMSs in §5.4. We focus on our Server setup here. The second setup (Laptop) is discussed in our technical report [1], the third (Docker) is provided to support replication of our methods.

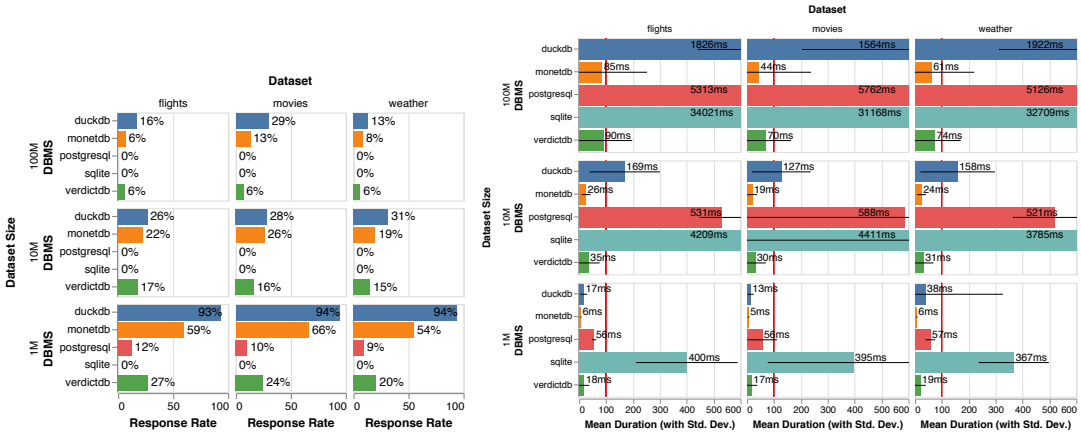
- **Server**: a single server (Red Hat 7 operating system), with 20 GB of memory, 12 cores (Intel(R) Xeon(R) Silver 4116 CPU @ 2.10GHz), and over 30 TB of disk space.
- **Laptop**: a Macbook Pro (10.14 Mojave), six cores (Intel Core i7 @ 2.2GHz), 16 GB of RAM and 250GB SSD.
- **Docker**: a Docker image (Ubuntu 18.04), provided with our benchmark (see footnote 1).

Default settings were used for all DBMSs and environment setups. Connection pooling was used for all drivers that supported it (MonetDB, PostgreSQL, VerdictDB).

6.1.1 Dataset Generation. Our goal was to capture natural exploration behavior in our benchmark (see §3.2 for details). As such, we initially curated small datasets to ensure participants' explorations were unimpeded by latency, similar to prior work [8, 73]. However, we need the ability to scale the data up to measure effects on DBMS performance.

We generated synthetic versions of the same datasets in §3 (similar to [73]). We construct statistical models capturing attribute relationships and patterns (e.g., distributions, correlations, etc.). We then generate new samples using these statistical models. Rather than testing on a single massive dataset, we test on three increasingly large dataset sizes, illustrating broader performance patterns as the data is scaled up: one million rows, 10 million rows, and 100 million rows.

[2] IDEBench drivers are straightforward to implement for many systems, please see our code for five examples.



(a) Response Rate (in Percent). **(b) Mean Query Duration (in Milliseconds).**

Figure 6: Server results, displaying response rate and mean query duration, faceted by dataset size and name.

In total, we evaluate *nine dataset cases*: (synthetic flights, movies, weather data) × (1M, 10M, 100M rows). Uncompressed, dataset sizes range from 54MB (flights-1M) to 7.8GB (movies-100M). Here, "large" is relative: though not large in an absolute sense, these scale-up scenarios still pose a serious performance challenge, as we will show in our results.

6.1.2 Workflow Evaluation. Each of our 22 study participants completed four exploration tasks for two datasets, resulting in 128 traces (see §3.6), and ultimately 128 workflows for our performance evaluations.

Using IDEBench, we executed each workflow for each dataset size (1M, 10M, 100M) and each DBMS (MonetDB, PostgreSQL, VerdictDB with PostgreSQL, SQLite, DuckDB), resulting in $128 \times 3 \times 5 = 1920$ individual experiments, which we aggregated by dataset and size to form 9 larger experiments (flights-1M, movies-10M, etc.).

6.2 Server Results

Here, we discuss our findings from our nine aggregated experiments using the measures described in §5.3. Each aggregated experiment is represented as one visualization in the small multiples visualizations in Figure 6a and Figure 6b.

6.2.1 Response Rate (Equation 3). Here, we measure the fraction of queries that were successfully answered by each DBMS within the specified time threshold of 100 ms (i.e., *type1* latency). As discussed in §2.2, real-time systems must support *type1* latency to maintain basic interactivity. Figure 6a shows our results for calculating response rate for all workflows. The best possible response rate is 100% (i.e., no queries were dropped, answered late, or canceled).

We found wide variations in performance even for our "easiest" scenario, the 1M rows case (bottom row of Figure 6a). In this case, SQLite never returned query results within 100ms. As observed in prior work [55], SQLite's design prioritizes transactional workloads, which results in poor analytical performance. PostgreSQL had slightly better performance, with response rates of 9% to 12%. VerdictDB has better performance than PostgreSQL with response rates of 20% to 27%, which is expected since VerdictDB is operating over a sample of the data. However, VerdictDB is surprisingly slow overall. We observe that VerdictDB seems to be designed for interactive *programming* environments, and incurs usage overhead that is invisible in a programming environment (e.g., a Jupyter Notebook), but still problematic in real-time visualization environments.

The two highest-performing DBMSs in the 1M rows case were MonetDB (54% to 59%) and DuckDB (93% to 94%). DuckDB's superior performance may be explained in part by its embeddable design, which eliminates some communication overhead, as well as its optimizations for both OLAP and OLTP workloads. However, DuckDB also assumes an interactive programming environment is being used. We see that DuckDB's performance degrades as we increase the dataset size from 1M to 10M rows, going from a response rate of 94% to 31%. From §4, we know that most interactions produce roughly 175 queries per second. These results suggest that in the 10M case, most DBMSs would support roughly 54 queries per second, making the system appear about 3X slower in terms of responsiveness (assuming the worst-case time threshold of 100ms). In fact, all of the DBMSs we tested had very low response rates in the 10M case, showing that scale-up is a serious problem for interactive use cases.

In the 100M rows case, we found that no DBMS provided better than a 29% response rate, and DuckDB was again the best performing DBMS. But DuckDB had 16% response rate or less in 2 out of 3 datasets for the 100M case. These other scenarios translate to about roughly a 7X slowdown in the interface. Thus, as the data scales up, no DBMS seems to be able to consistently support our latency requirements.

6.2.2 Query Duration (Equation 4). Here, we analyze the mean duration (in milliseconds) for each query successfully answered by each DBMS. Note that in this case, we include all

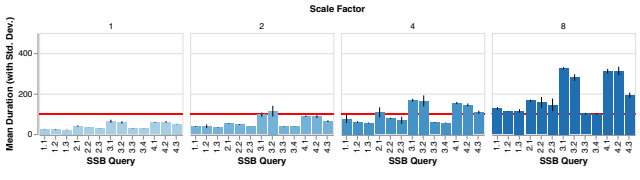


Figure 7: Mean query duration and standard deviation (in milliseconds) for the 13 queries in the SSB. The mean is calculated across five runs executed for MonetDB. MonetDB performs well (i.e., within our 100ms threshold, drawn in red) up to scale factor four.

answered queries in our calculation of the mean, even if they did not meet the time threshold of 100ms. Our results are displayed in Figure 6b, with standard deviation represented in the error bars. We find that in the 1M rows case, most DBMSs are able to meet our 100ms threshold, highlighted in red. The only exception is SQLite, which returned results in about 390-400ms on average. Note that most of VerdictDB's queries are dropped (i.e., low response rate) but the few remaining queries execute quickly (i.e., fast query duration).

As dataset size increases, we see an order of magnitude increase in query duration for PostgreSQL, SQLite and DuckDB. Furthermore, PostgreSQL and DuckDB switch to violating the 100ms threshold. This linear scale-up behavior suggests that visualizing larger sizes (e.g., billions of rows) may be impractical. We find that mean query duration does not increase as rapidly for MonetDB and VerdictDB, and their mean query durations stay within the 100ms threshold for all three dataset sizes. However, this time threshold is clearly difficult to maintain, as demonstrated by the wide error bars in the 100M rows case for MonetDB and VerdictDB.

6.2.3 Accuracy (Equation 6). Besides the response rate and query duration, we also evaluated the accuracy of the results returned by VerdictDB, the approximate system we used for our experiments. We configured VerdictDB to use a scramble (sample) table of 10% of the original data sizes (1M and 10M for this experiment). For all 128 workflows we then computed the ground-truth for a total number of 732,723 returned results, using MonetDB. On the 1M dataset, we measure a MRE of 0.0412, and a similarly low percentage (4.29%, σ = 12.99%) of missing bins. Interestingly, on the 10M datasets we measure a lower MRE (0.0108%), and fewer missing bins 1.46% (σ = 7.42%), indicating that the larger sample (10 x as large as the one used for the 1M dataset) leads to better quality results and smaller error variance.

6.3 Comparing with SSB & TPC-H

6.3.1 Performance on SSB. We also compare our selected DBMSs using the Star Schema Benchmark (SSB) [46], an extension of TPC-H. As there is no standard query generator for SSB, we created one for the community, using the query specifications of O'Neil et al. [46]. Our code is available in our OSF repository. The SSB with scale factor one is most similar to our 10M use case in terms of table sizes (roughly 600MB and millions of rows). Therefore, we compare our benchmark results with SSB scale factor one. Due to space constraints, only the results for MonetDB are displayed in Figure 7 (see our Technical Report for more details [1]).

MonetDB provides blazing fast mean query duration (i.e., within our threshold of 100ms) for nearly all SSB queries until scale factor four, but even then MonetDB maintains its performance for seven out of thirteen queries (response rate of 53%). In contrast, all other DBMSs were significantly slower than our latency threshold (our technical report has more details [1]). However, as discussed in §6.2, none of the DBMSs provide satisfactory response rates for our 10M use case, with the highest response rate being 31% (from DuckDB). These differences in performance are due largely to the differences in the query workloads of the two benchmarks. Slow and periodic reports-based workloads observed in SSB (and TPC-H) differ significantly from the rapid and ad-hoc query workloads generated by real people in crossfilter use cases, leading to different performance profiles.

6.3.2 DuckDB on TPC-H. DuckDB is tested regularly[3], including with TPC-H with scale factor one, which we report here for comparison. Note that our 10M datasets are of similar magnitude to the TPC-H tables (533-794MB). While DuckDB has competitive (subsecond) latencies on most TPC-H queries, it has poor performance on our visualization benchmark, also due to differences in queries: TPC-H emphasizes slow, reports-based analytics, compared to the rapid and ad-hoc queries from real user interactions in our benchmark.

7 DISCUSSION

In this work, we present initial steps towards an end-to-end performance benchmark for interactive, real-time querying scenarios, derived from user study data that we collected for crossfilter contexts (a representative of dynamic queries). Our benchmark design is inspired by recent proposals [6, 17] and tutorials [28, 29, 61] across multiple communities, and incorporates methodology from HCI, visualization, and databases. We ran our benchmark with 128 workflows and five different DBMSs, and found that none of these systems could adequately support our benchmark workload for datasets considered far from large in the database community.

Our benchmark will enable database designers to test whether their systems are suitable for interactive and real-time scenarios, as well as provide opportunities to leverage knowledge of user interaction behavior to develop novel optimization strategies. We provide the user study, evaluation, and analysis materials as a community resource, to support and encourage further research in this area (see footnote 1).

[3] https://www.duckdb.org/benchmarks/index.html

7.1 Interaction and Performance

The key factor that differentiates our performance benchmark from others is its ability to capture the cadence and flow of data exploration interactions made by real people in real time, and as a result the dynamic and ad-hoc nature of the resulting database queries. As such, a clear understanding of user interaction behavior is critical to discerning the impact of DBMS performance on end users. To this end, we analyzed the interaction behaviors and patterns observed in our user study data. We find that when given the opportunity to use crossfilter interactions, people take full advantage of their capabilities. These results suggest that by ignoring the crossfilter use case, DBMSs are missing out on a tremendous opportunity to have a strong positive impact on interactive data analytics. Furthermore, we find that user interactions generally produce hundreds of queries per second, potentially placing an intense load on the underlying DBMS.

7.2 Optimization Opportunities

Here, we discuss opportunities to improve DBMS performance for dynamic queries, given our results.

Approximate query engines show promise for interactive use cases [22], however these systems have not been designed with *type1* latency in mind. In particular, these systems assume they are being executed in a *programming environment* (e.g., Jupyter Notebook, etc.), where execution overhead is hidden by the naturally slow cadence of writing and testing code. More work remains to remove this overhead to enable real-time interactive data exploration.

Most existing database (or visualization) lineage capture techniques would be at best wasteful, and at worst overwhelmed, given the rapid (e.g., 175 queries per second) updates produced in crossfilter use cases. Rather than logging and responding to every event, it would be more effective to strategically throw non-essential events away, a promising new direction explored in recent work [54], and could be investigated further to support *type1* latency.

Event modulation (e.g., throttling, debouncing, etc.) is a form of admission control, which has been studied in other database contexts. Existing techniques could be adapted to better handle high-load scenarios for crossfilter use cases. It can also be advantageous to merge queries to reduce total queries executed, but not in every case [68]. Identifying optimal merges could be interesting future work.

Another optimization opportunity lies in predicting what interactions users will perform next [8, 11, 45]. Our collected data provides an opportunity to construct new user interaction models, which could in turn be used to speculatively execute queries ahead of users as they explore.

7.3 Limitations & Future Work

One concern could be that only one interface design is represented in the current benchmark (crossfilter). However, as stated in §1 and §2, not all interactive scenarios are worth exploring from a performance perspective. We specifically sought out use cases that would push the limits of what current DBMSs are capable of. Furthermore, we present not only the data we collected for this use case, but a full pipeline for developing new interactive benchmarks for the future. Anyone can take our code and data, and adapt them to other interface and study designs to produce new benchmarks.

We believe that a single benchmark cannot (and should not) cover all visualization scenarios. For example, the TPC has separated classic database workloads into multiple benchmarks (OLTP, OLAP, etc.). Hence, we argue for a similar approach: developing an *ecosystem* of exploration benchmarks. We envision a cross-community effort to develop benchmarks for a wide range of interactive use cases. Inspired by previous proposals (e.g., [6]), we are working towards a more general benchmark *repository and platform*, to execute data-driven benchmark development on a larger scale. Then anyone in the community could contribute new inputs (user logs, input datasets, interface designs), and the platform could derive benchmarks accordingly.

ACKNOWLEDGMENTS

This work was supported in part by NSF Award IIS-1850115, and the Moore Foundation Data Driven Discovery Investigator program. We thank Schloss Dagstuhl—Leibniz Center for Informatics for their support in organizing "Dagstuhl Seminar 17461 - Connecting Visualization and Data Management Research". We thank Yinan Chen, Pooja Gada, and Simrat Bhandari for their help in conducting the user study. We also thank Daniel Abadi, Amol Desphande, and the anonymous referees for their valuable comments and suggestions.

REFERENCES

[1] 2020. *Technical Report: Database Benchmarking for Supporting Real-Time InteractiveQuerying of Large Data.* Technical Report. https://osf.io/ymndj/?view_only=81de1a3f99d04529b6b173a3bd5b4d23

[2] Sameer Agarwal, Barzan Mozafari, Aurojit Panda, Henry Milner, Samuel Madden, and Ion Stoica. 2013. BlinkDB: queries with bounded errors and bounded response times on very large data. In *Proceedings of the 8th ACM European Conference on Computer Systems*. ACM, 29–42.

[3] Susanne Albers. 2003. Online algorithms: a survey. *Mathematical Programming* 97, 1-2 (2003), 3–26.

[4] Sara Alspaugh, Nava Zokaei, Andrea Liu, Cindy Jin, and Marti A Hearst. 2018. Futzing and moseying: Interviews with professional data analysts on exploration practices. *IEEE Trans. Vis. Comput. Graphics* 25, 1 (2018), 22–31.

[5] Sriram Karthik Badam, Niklas Elmqvist, and Jean-Daniel Fekete. 2017. Steering the craft: UI elements and visualizations for supporting progressive visual analytics. *Computer Graphics Forum* 36, 3 (2017), 491–502.

[6] Leilani Battle, Marco Angelini, Carsten Binnig, Tiziana Catarci, Philipp Eichmann, Jean-Daniel Fekete, Giuseppe Santucci, Michael Sedlmair, and Wesley Willett. 2018. Evaluating Visual Data Analysis Systems: A Discussion Report. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics (HILDA'18)*. ACM, New York, NY, USA, Article 4, 6 pages. https://doi.org/10.1145/3209900.3209901

[7] Leilani Battle, Remco Chang, Jeffrey Heer, and Mike Stonebraker. 2017. Position statement: The case for a visualization performance benchmark. In *Data Systems for Interactive Analysis (DSIA), 2017 IEEE Workshop on*. IEEE, 1–5.

[8] Leilani Battle, Remco Chang, and Michael Stonebraker. 2016. Dynamic Prefetching of Data Tiles for Interactive Visualization. In *Proceedings of the 2016 International Conference on Management of Data (SIGMOD '16)*. ACM, New York, NY, USA, 1363–1375. https://doi.org/10.1145/2882903.2882919

[9] Leilani Battle and Jeffrey Heer. 2019. Characterizing Exploratory Visual Analysis: A Literature Review and Evaluation of Analytic Provenance in Tableau. *Computer Graphics Forum* 38, 3 (2019), 145–159.

[10] Peter A Boncz, Marcin Zukowski, and Niels Nes. 2005. MonetDB/X100: Hyper-Pipelining Query Execution.. In *Cidr*, Vol. 5. 225–237.

[11] Eli T Brown, Alvitta Ottley, Helen Zhao, Quan Lin, Richard Souvenir, Alex Endert, and Remco Chang. 2014. Finding waldo: Learning about users from their interactions. *IEEE Trans. Vis. Comput. Graphics* 20, 12 (2014), 1663–1672.

[12] Stuart K. Card, Jock D. Mackinlay, and Ben Shneiderman (Eds.). 1999. *Readings in Information Visualization: Using Vision to Think*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[13] Surajit Chaudhuri, Gautam Das, and Vivek Narasayya. 2007. Optimized stratified sampling for approximate query processing. *ACM Transactions on Database Systems (TODS)* 32, 2 (2007), 9.

[14] Surajit Chaudhuri, Bolin Ding, and Srikanth Kandula. 2017. Approximate Query Processing: No Silver Bullet. In *Proceedings of the 2017 ACM International Conference on Management of Data (SIGMOD '17)*. ACM, New York, NY, USA, 511–519. https://doi.org/10.1145/3035918.3056097

[15] Andrew Crotty, Alex Galakatos, Emanuel Zgraggen, Carsten Binnig, and Tim Kraska. 2015. Vizdom: Interactive Analytics Through Pen and Touch. *Proc. VLDB Endow.* 8, 12 (Aug. 2015), 2024–2027. https://doi.org/10.14778/2824032.2824127

[16] Andrew Crotty, Alex Galakatos, Emanuel Zgraggen, Carsten Binnig, and Tim Kraska. 2016. The Case for Interactive Data Exploration Accelerators (IDEAs). In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics (HILDA '16)*. ACM, New York, NY, USA, Article 11, 6 pages. https://doi.org/10.1145/2939502.2939513

[17] Philipp Eichmann, Carsten Binnig, Tim Kraska, and Emanuel Zgraggen. 2018. IDEBench: A Benchmark for Interactive Data Exploration. *CoRR* abs/1804.02593 (2018). https://arxiv.org/abs/1804.02593

[18] Philipp Eichmann, Emanuel Zgraggen, Zheguang Zhao, Carsten Binnig, and Tim Kraska. 2016. Towards a Benchmark for Interactive Data Exploration. *IEEE Data Eng. Bull.* 39, 4 (2016), 50–61.

[19] Jean-Daniel Fekete, Danyel Fisher, Arnab Nandi, and Michael Sedlmair. 2019. Progressive Data Analysis and Visualization (Dagstuhl Seminar 18411). *Dagstuhl Reports* 8, 10 (2019), 1–40. https://doi.org/10.4230/DagRep.8.10.1

[20] Mi Feng, Evan Peck, and Lane Harrison. 2018. Patterns and pace: Quantifying diverse exploration behavior with visualizations on the web. *IEEE Trans. Vis. Comput. Graphics* 25, 1 (2018), 501–511.

[21] M. Feng, E. Peck, and L. Harrison. 2019. Patterns and Pace: Quantifying Diverse Exploration Behavior with Visualizations on the Web. *IEEE Trans. Vis. Comput. Graphics* 25, 1 (Jan 2019), 501–511. https://doi.org/10.1109/TVCG.2018.2865117

[22] Alex Galakatos, Andrew Crotty, Emanuel Zgraggen, Carsten Binnig, and Tim Kraska. 2017. Revisiting reuse for approximate query processing. *Proceedings of the VLDB Endowment* 10, 10 (2017), 1142–1153.

[23] Parke Godfrey, Jarek Gryz, and Piotr Lasek. 2016. Interactive visualization of large data sets. *IEEE Transactions on Knowledge and Data Engineering* 28, 8 (2016), 2142–2157.

[24] David Gotz and Michelle X Zhou. 2009. Characterizing users' visual analytic activity for insight provenance. *Information Visualization* 8, 1 (2009), 42–55.

[25] Joseph M Hellerstein, Ron Avnur, Andy Chou, Christian Hidber, Chris Olston, Vijayshankar Raman, Tali Roth, and Peter J Haas. 1999. Interactive data analysis: The control project. *Computer* 32, 8 (1999), 51–59.

[26] Joseph M Hellerstein, Peter J Haas, and Helen J Wang. 1997. Online aggregation. *Acm Sigmod Record* 26, 2 (1997), 171–182.

[27] Kevin Hu, Neil Gaikwad, Michiel Bakker, Madelon Hulsebos, Emanuel Zgraggen, César Hidalgo, Tim Kraska, Guoliang Li, Arvind Satyanarayan, and Çağatay Demiralp. 2019. VizNet: Towards a large-scale visualization learning and benchmarking repository. In *Proceedings of the 2019 Conference on Human Factors in Computing Systems (CHI) (CHI '19)*. ACM, New York, NY, USA, Article Paper 662, 12 pages. https://doi.org/10.1145/3290605.3300892

[28] Stratos Idreos, Olga Papaemmanouil, and Surajit Chaudhuri. 2015. Overview of data exploration techniques. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. ACM, 277–281.

[29] Lilong Jiang, Protiva Rahman, and Arnab Nandi. 2018. Evaluating interactive data systems: Workloads, metrics, and guidelines. In *Proceedings of the 2018 International Conference on Management of Data*. ACM, 1637–1644.

[30] Niranjan Kamat, Prasanth Jayachandran, Karthik Tunga, and Arnab Nandi. 2014. Distributed and interactive cube exploration. In *2014 IEEE 30th International Conference on Data Engineering*. IEEE, 472–483.

[31] Matthew Kay and Jeffrey Heer. 2015. Beyond weber's law: A second look at ranking visualizations of correlation. *IEEE Trans. Vis. Comput. Graphics* 22, 1 (2015), 469–478.

[32] Heidi Lam. 2008. A Framework of Interaction Costs in Information Visualization. *IEEE Trans. Vis. Comput. Graphics* 14, 6 (Nov 2008), 1149–1156. https://doi.org/10.1109/TVCG.2008.109

[33] Heidi Lam, Enrico Bertini, Petra Isenberg, Catherine Plaisant, and Sheelagh Carpendale. 2011. Empirical studies in information visualization: Seven scenarios. *IEEE Trans. Vis. Comput. Graphics* 18, 9 (2011), 1520–1536.

[34] Heidi Lam, Melanie Tory, and Tamara Munzner. 2017. Bridging from goals to tasks with design study analysis reports. *IEEE Trans. Vis. Comput. Graphics* 24, 1 (2017), 435–445.

[35] Qing Li, Xiaofeng Bao, Chen Song, Jinfei Zhang, and Chris North. 2003. Dynamic Query Sliders vs. Brushing Histograms. In *CHI '03 Extended Abstracts on Human Factors in Computing Systems (CHI EA '03)*. ACM, New York, NY, USA, 834–835. https://doi.org/10.1145/765891.766020

[36] Lauro Lins, James T Klosowski, and Carlos Scheidegger. 2013. Nanocubes for real-time exploration of spatiotemporal datasets. *IEEE Trans. Vis. Comput. Graphics* 19, 12 (2013), 2456–2465.

[37] Zhicheng Liu and Jeffrey Heer. 2014. The Effects of Interactive Latency on Exploratory Visual Analysis. *IEEE Trans. Vis. Comput. Graphics* 20, 12 (Dec 2014), 2122–2131. https://doi.org/10.1109/TVCG.2014.2346452

[38] Zhicheng Liu, Biye Jiang, and Jeffrey Heer. 2013. imMens: Real-time Visual Querying of Big Data. *Computer Graphics Forum* 32, 3 (2013).

[39] Robert B. Miller. 1968. Response Time in Man-computer Conversational Transactions. In *Proc. of the Fall Joint Computer Conference, Part I*. ACM, 267–277. https://doi.org/10.1145/1476589.1476628

[40] Tova Milo and Amit Somech. 2018. Next-step suggestions for modern interactive data analysis platforms. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 576–585.

[41] Dominik Moritz, Bill Howe, and Jeffrey Heer. 2019. Falcon: Balancing Interactive Latency and Resolution Sensitivity for Scalable Linked Visualizations. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '19)*. ACM, New York, NY, USA, 11. https://doi.org/10.1145/3290605.3300924

[42] Jakob Nielsen. 2016. Response Times: The 3 Important Limits. https://www.nngroup.com/articles/response-times-3-important-limits/. Accessed: 2019-07-20.

[43] Dan R. Olsen, Jr. 1998. *Developing User Interfaces*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[44] Eyal Oren, Christophe Guéret, and Stefan Schlobach. 2008. Anytime query answering in RDF through evolutionary algorithms. In *International Semantic Web Conference*. Springer, 98–113.

[45] Alvitta Ottley, Roman Garnett, and Ran Wan. 2019. Follow The Clicks: Learning and Anticipating Mouse Interactions During Exploratory Data Analysis. In *Computer Graphics Forum*, Vol. 38. Wiley Online Library, 41–52.

[46] Patrick O'Neil, Elizabeth O'Neil, Xuedong Chen, and Stephen Revilak. 2009. The star schema benchmark and augmented fact table indexing. In *Technology Conference on Performance Evaluation and Benchmarking*. Springer, 237–252.

[47] Patrick E O'Neil, Elizabeth J O'Neil, and Xuedong Chen. 2007. The star schema benchmark (SSB). *Pat* 200, 0 (2007), 50.

[48] Themis Palpanas, Nick Koudas, and Alberto Mendelzon. 2005. Using datacube aggregates for approximate querying and deviation detection. *IEEE transactions on knowledge and data engineering* 17, 11 (2005).

[49] Yongjoo Park, Barzan Mozafari, Joseph Sorenson, and Junhao Wang. 2018. VerdictDB: Universalizing Approximate Query Processing. In *Proceedings of the 2018 International Conference on Management of Data (SIGMOD '18)*. ACM, New York, NY, USA, 1461–1476.

[50] William A Pike, John Stasko, Remco Chang, and Theresa A O'connell. 2009. The science of interaction. *Information Visualization* 8, 4 (2009).

[51] Harald Piringer, Christian Tominski, Philipp Muigg, and Wolfgang Berger. 2009. A Multi-Threading Architecture to Support Interactive Visual Exploration. *IEEE Trans. Vis. Comput. Graphics* 15, 6 (Nov 2009), 1113–1120. https://doi.org/10.1109/TVCG.2009.110

[52] Catherine Plaisant, Jean-Daniel Fekete, and Georges Grinstein. 2007. Promoting insight-based evaluation of visualizations: From contest to benchmark repository. *IEEE Trans. Vis. Comput. Graphics* 14, 1 (2007), 120–134.

[53] Fotis Psallidas and Eugene Wu. 2018. Provenance for Interactive Visualizations. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics (HILDA'18)*. ACM, New York, NY, USA, Article 9, 8 pages. https://doi.org/10.1145/3209900.3209904

[54] Fotis Psallidas and Eugene Wu. 2018. Smoke: Fine-grained lineage at interactive speed. *Proceedings of the VLDB Endowment* 11, 6 (2018).

[55] Mark Raasveldt and Hannes Mühleisen. 2019. DuckDB: An Embeddable Analytical Database. In *Proceedings of the 2019 International Conference on Management of Data (SIGMOD '19)*. ACM, New York, NY, USA, 1981–1984. https://doi.org/10.1145/3299869.3320212

[56] Ben Shneiderman. 1984. Response Time and Display Rate in Human Performance with Computers. *ACM Comput. Surv.* 16, 3 (Sept. 1984), 265–285. https://doi.org/10.1145/2514.2517

[57] Ben Shneiderman. 1994. Dynamic Queries for Visual Information Seeking. *IEEE Softw.* 11, 6 (Nov. 1994), 70–77. https://doi.org/10.1109/52.329404

[58] Spotfire 1995. TICO Spotfire. https://www.tibco.com/products/tibco-spotfire. Accessed: 2019-07-20.

[59] Chris Stolte, Diane Tang, and Pat Hanrahan. 2002. Polaris: A system for query, analysis, and visualization of multidimensional relational databases. *IEEE Trans. Vis. Comput. Graphics* 8, 1 (2002), 52–65.

[60] Tableau 2003. Tableau Software. https://www.tableau.com/. Accessed: 2019-07-20.

[61] Nan Tang, Eugene Wu, and Guoliang Li. 2019. Towards Democratizing Relational Data Visualization. In *Proceedings of the 2019 International Conference on Management of Data*. ACM, 2025–2030.

[62] Egemen Tanin, Richard Beigel, and Ben Shneiderman. 1996. Incremental Data Structures and Algorithms for Dynamic Query Interfaces. *SIGMOD Rec.* 25, 4 (Dec. 1996), 21–24. https://doi.org/10.1145/245882.245891

[63] Wenbo Tao, Xiaoyu Liu, Yedi Wang, Leilani Battle, Çağatay Demiralp, Remco Chang, and Michael Stonebraker. 2019. Kyrix: Interactive Pan/Zoom Visualizations at Scale. *Computer Graphics Forum* 38, 3 (2019), 529–540.

[64] TPC-DS 2016. TPC-DS. http://www.tpc.org/tpcds/. Accessed: 2017-11-02.

[65] TPC-H 2016. TPC-H. http://www.tpc.org/tpch/. Accessed: 2017-11-02.

[66] John W Tukey. 1977. *Exploratory data analysis*. Vol. 2. Reading, Mass.

[67] Lisa Tweedie, Bob Spence, David Williams, and Ravinder Bhogal. 1994. The Attribute Explorer. In *Conference Companion on Human Factors in Computing Systems (CHI '94)*. ACM, New York, NY, USA, 435–436. https://doi.org/10.1145/259963.260433

[68] Manasi Vartak, Sajjadur Rahman, Samuel Madden, Aditya Parameswaran, and Neoklis Polyzotis. 2015. Seedb: Efficient data-driven visualization recommendations to support visual analytics. In *Proceedings of the VLDB Endowment International Conference on Very Large Data Bases*, Vol. 8. 2182.

[69] C. Weaver. 2008. Multidimensional visual analysis using cross-filtered views. In *2008 IEEE Symposium on Visual Analytics Science and Technology*. 163–170. https://doi.org/10.1109/VAST.2008.4677370

[70] K. Wongsuphasawat, D. Moritz, A. Anand, J. Mackinlay, B. Howe, and J. Heer. 2016. Voyager: Exploratory Analysis via Faceted Browsing of Visualization Recommendations. *IEEE Trans. Vis. Comput. Graphics* 22, 1 (Jan. 2016), 649–658. https://doi.org/10.1109/TVCG.2015.2467191

[71] Kanit Wongsuphasawat, Zening Qu, Dominik Moritz, Riley Chang, Felix Ouk, Anushka Anand, Jock Mackinlay, Bill Howe, and Jeffrey Heer. 2017. Voyager 2: Augmenting Visual Analysis with Partial View Specifications. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '17)*. ACM, New York, NY, USA, 2648–2659. https://doi.org/10.1145/3025453.3025768

[72] Emanuel Zgraggen, Alex Galakatos, Andrew Crotty, Jean-Daniel Fekete, and Tim Kraska. 2017. How Progressive Visualizations Affect Exploratory Analysis. *IEEE Trans. Vis. Comput. Graphics* 23, 8 (2017), 1977–1987. https://doi.org/10.1109/TVCG.2016.2607714

[73] Emanuel Zgraggen, Zheguang Zhao, Robert Zeleznik, and Tim Kraska. 2018. Investigating the Effect of the Multiple Comparisons Problem in Visual Analysis. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI '18)*. ACM, New York, NY, USA, Article 479, 12 pages. https://doi.org/10.1145/3173574.3174053