Why TPC Is Not Enough: An Analysis of the Amazon Redshift Fleet

Alexander van Renen*† — UTN — alexander.van.renen@utn.de

Dominik Horn* — Amazon Web Services — domhorn@amazon.de

Pascal Pfeil* — Amazon Web Services — pfeip@amazon.de

Kapil Vaidya — Amazon Web Services — kapivaid@amazon.com

Wenjian Dong — Amazon Web Services — wjdong@amazon.fr

Murali Narayanaswamy — Amazon Web Services — muralibn@amazon.com

Zhengchun Liu — Amazon Web Services — zcl@amazon.com

Gaurav Saxena — Amazon Web Services — gssaxena@amazon.com

Andreas Kipf † — UTN — andreas.kipf@utn.de

Tim Kraska — Amazon Web Services — timkrask@amazon.com

ABSTRACT

Database research and development is heavily influenced by benchmarks, such as the industry-standard TPC-H and TPC-DS for analytical systems. However, these twenty-year-old benchmarks neither capture how databases are deployed nor what workloads modern cloud data warehouse systems face these days.

In this paper, we summarize well-known, confirm suspected, and unearth novel discrepancies between TPC-H/DS and actual workloads using empirical data. We base our analysis on telemetrics from Amazon Redshift – one of the largest cloud data warehouse deployments. Among others, we show how write-heavy data pipelines are prominent, workloads vary over time (in both load and type), queries are repetitive, and how most properties of queries or workloads experience very long tailed distributions. We conclude that data warehouse benchmarks, just like database systems, need to become more holistic and stop focusing solely on query engine performance. Finally, we publish a dataset containing query statistics of 200 randomly selected Redshift serverless and provisioned instances (each) over a three-month period, as a basis for building more realistic benchmarks.

PVLDB Reference Format:
Alexander van Renen, Dominik Horn, Pascal Pfeil, Kapil Vaidya, Wenjian Dong, Murali Narayanaswamy, Zhengchun Liu, Gaurav Saxena, Andreas Kipf, and Tim Kraska. Why TPC Is Not Enough: An Analysis of the Amazon Redshift Fleet. PVLDB, 17(11): 3694 - 3706, 2024.
doi:10.14778/3681954.3682031

*Contributed equally.
†Work performed while at Amazon Web Services.

Proceedings of the VLDB Endowment, Vol. 17, No. 11 ISSN 2150-8097.
doi:10.14778/3681954.3682031

Table 1: Redshift vs. TPC-H/DS: *Key differences between real workloads and synthetic benchmarks.*

	Name (Section)	TPC-H/DS	Redshift Fleet
QUERY	Types (3.2)	read-mostly	write/read
	Complexity (3.4)	narrow	large/long tails
	Operators (3.4)	mostly-join	varied
	Scans Filters (3.5)	full scan	function calls
	CTAS (3.3)	none	freq./repeating
WORKLOAD	Weekly pattern (4.1)	N/A	yes
	Daily pattern (4.2)	N/A	no
	Distribution (4.3)	narrow	large/long tails
	Repeating (4.4)	per-run	daily
	↪ Queries	↪ none	↪ high
	↪ Templates	↪ all	↪ high
	↪ Filters	↪ high	↪ high
DATA	Types (5.1)	fixed text int/date	varchar timestamp
	Table growth (5.2)	all stable	mostly stable
	Skew (5.3)	low	high
	External formats (5.4)	only CSV	mostly CSV

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

1 INTRODUCTION

TPC-H/DS. The TPC-H [32] and TPC-DS [31] benchmarks have been the de-facto standard (heavily skewed towards TPC-H) to evaluate and compare database systems in industry [2, 9, 15, 29] as well as in academia. In fact, we searched through all VLDB papers from the previous five years and found that 14 % of them mention

TPC-H/DS. Both benchmarks have been created to stress test the query throughput of analytical data warehouse applications, faced with a decision support workload. While rarely used in most evaluations, both benchmarks include periodic data ingestion tasks in the form of bulk deletes and inserts (with some ELT in the case of TPC-DS). The benchmarks assume a closed system, i.e., a fixed number of active user sessions, which each issue queries back to back. These properties make for a highly valuable test for the performance of a query engine. However, typical queries in Redshift's fleet don't query as many tables as TPC-DS does, indicating that TPC-DS only represents the tail end of analytical workloads. More importantly, as we will outline in the following sections, a cloud data warehouse consists of more than just a query engine (e.g., workload management, adaptive scaling) and we need to evaluate these additional components as well.

Cloud Data Warehouses. Databases have, as they always do, rapidly evolved over the previous decade to adopt to a new landscape, namely: the cloud. Most databases' software is no longer shipped as an on-premise deployment to run on dedicated machines for a fixed number of users. Instead, databases are sold as a (serverless) services, integrated into an intricate cloud ecosystem: Customers expect an elastic data processing platform that can be used for a wide range of workloads ranging from data scientist tasks, over ETL pipelines and ML, to dashboarding. These database offerings are often used by a varying number of users (open system) and can interconnect with many systems. Hence, with databases encompassing more and more features, benchmarks like TPC-H/DS only evaluate a narrow portion of the overall system.

Redshift Fleet Statistics. Amazon Redshift is a managed database service fully adopted to the previously outlined cloud landscape. As part of Amazon Web Services' (AWS) offerings, it is one of the most widely used cloud data warehouses on the market. In this paper, we publish insights from analyzing its telemetry, which permits us to empirically evaluate what customers are doing with their database systems in this novel landscape. Our findings confirm some previously suspected usage patterns and uncover novel insights. The goal of this work is to highlight potential avenues of research, close down some dead ends, and connect database research better with actual customer use cases.

Contributions. We categorize our findings into three sections relating to (I) individual queries (Section 3), (II) entire workloads (Section 4), and (III) dataset characteristics (Section 5). All our findings are summarized in Table 1, which gives an overview of how actual workloads differ from TPC-H/DS. One surprising finding in this work is the repetitiveness of queries: We find that in 50 % of database clusters 80 % of queries are 1-to-1 repetitions of previously seen queries (details in Section 4.4). This finding presents huge opportunities for result caching and (automatic) Materialized Views, which are extensively used in Redshift. We break down how well our caching implementation is currently working and show open opportunities in Section 4.6. Next to our analysis of fleet data, we open source a dataset of query logs ("Redset", described in Section 6) containing statistics over all queries that ran on a sample of Redshift instances over the span of three months.

2 EXPERIMENTAL METHODOLOGY

Dataset. The presented data is based on statistics over database clusters from the Redshift fleet. If not otherwise mentioned, it relates to the provisioned offering of Redshift and data over one month. Please note that some statistics cannot be published because they contain sensitive information.

Privacy. Given that all statistics are based on customer clusters, we cannot access data directly due to Amazon regulations for preventing exposure of any sensitive information or any potential impact on normal operations of customer clusters. All statistics are therefore based on externalized, anonymized telemetry data.

TPC-H/DS. All experiments relating to TPC-H/DS were conducted on an off-the-shelf ten node `ra3.4xlarge` provisioned Redshift cluster. We chose a TPC-H/DS scale factor of 3 TB for both benchmarks. We use all queries of the respective benchmarks, including the standardized, but lesser known insert/delete queries.

3 HOW DO THE QUERIES DIFFER?

In this section, we give a general overview over the types of queries and their resource demands being run on Redshift. After that, we dive deep into the distribution of plan operator nodes, the complexity of filter conditions, and the large amount of CTAS queries.

3.1 A qualitative impression

Combining these fleet observations with our experience from talking to hundreds of customers, we note a few interesting trends. While there are certainly more, all Redshift directors considered these as most notable.

Live dashboards. Customers are running live, latency sensitive dashboard workloads at high concurrency. The p95 response time is often < 3 seconds as some user is waiting on the dashboard to render. Often, new data is being ingested every few minutes or sometimes continuously, limiting the impact of caching whole query results.

ETL to ELT. We observe workload patterns are shifting from ETL (extract-transform-load) to ELT (extract-load-transform). Customers are landing raw data, both structured and semi-structured, and curating data on Redshift for downstream consumption. Data freshness SLAs require ingestion pipelines to be simplified and reduce the number of hops. The transformation is done either within Redshift with SQL or with tight read-transform-write back loops. Features like Zero-ETL are just accelerating this trend.

Lazy SQL. A combination of machine-generated SQL and the non-SQL-expert persona means that queries received by the engine may not be efficiently written. Tools which generate SQL, often try to make the generation process easier, rather than trying to focus on easier to optimize queries (e.g., they use subqueries, temp tables for parameters, etc.). Query rewriters and optimizers need to adapt to handle this efficiently to maintain good query performance.

Repetition. There is a high rate of query repetition encouraging analytics engines to go beyond simple caching techniques to approaches that learn or even memorize from the initial runs.

Highly variable workloads. While some customers separate distinct workloads onto different clusters connected with data sharing, not all do. The result is that some workloads vary significantly over

Table 2: Query Types: *The different types of queries executed in Redshift, their cumulative runtime, and the size distribution of query statements.*

		Number of Queries			Query Runtimes			Query Text Sizes [byte]: Fleet					Mean	
		Fleet	TPC-H	TPC-DS	Fleet	TPC-H	TPC-DS	Median	Mean	P90	P99	Max	TPC-H	-DS
RO	Select	48.9%	75.9%	79.2%	**28.6%**	**91.6%**	**86.7%**	260	4396	1542	16413	16 776 K	823	2793
RW	Insert	19.5%	6.9%	5.6%	7.3%	.4%	.5%	378	9972	1401	141941	17 307 K	52	570
	Copy	7.4%	10.0%	9.6%	**38.3%**	**1.1%**	**10.8%**	337	431	541	2445	98 K	135	311
	Delete	6.6%	6.9%	5.6%	2.6%	7.0%	2.0%	204	340	376	950	15 962 K	73	577
	Update	3.8%	.0%	.0%	2.2%	.0%	.0%	239	807	789	11761	11 700 K	0	0
	CTAS	1.9%	.0%	.0%	**16.4%**	**.0%**	**.0%**	481	3595	3244	30987	13 817 K	0	0
Sys	Maint.	8.2%	.0%	.0%	3.3%	.0%	.0%	165	271	619	969	4 K	0	0
	Other	3.7%	.0%	.0%	1.3%	.0%	.0%	53	311	336	3942	7294 K	0	0

time and consist of diverse queries (dashboards, ETL, and large ad-hoc queries). This trend motivated the development of Redshift's AI-driven scaling [23].

3.2 Analytics = Updates

Updates are Frequent. To give an overview of the workload, we investigate which types of queries are being run on Redshift (cf. Table 2). Despite Redshift being an analytic data warehouse, we can see a large number of read/write queries (≈40 %). Most frequently, TPC-H/DS are executed as a read-only benchmark in industry and academia. But even if the two data refresh function of TPC-H (consisting of three `copies`, two `inserts`, and two `deletes`) and three maintenance functions of TPC-DS (consisting of twelve `copies`, seven `inserts`, and seven `deletes`), respectively, are taken into account the ratio of updates is still less than in Redshift (≈24 % for TPC-H and ≈21 % for TPC-DS).

Updates are Heavy. The "Query Runtimes" column of Table 2 show what percentage of the overall query runtime is spent on the various query types. It shows that the data manipulation queries are not only numerous, but also account for more runtime than read-only queries. This is in stark contrast to TPC-H/DS where read-only queries account for ≈90 % of time. This shows that the notion of a read-only data warehouse is outdated. Even analytical systems need to support updates and deal with their implications: staleness of statistics, reclaiming of freed space, maintenance of indexes, etc. We suspect that the influx in write queries could be partially due to people moving from ETL to ELT pipelines, especially with Redshift's new Zero-ETL feature that lets users continuously load data from transactional engines into Redshift [4].

System Maintenance. Next to user-related queries, we can also observe that ≈10 % of queries fall into the system category (≈5 % of runtime). This category is made up of all work that Redshift needs to perform internally to keep the system running. While these queries are certainly necessary to run TPC-H, they are usually not reported and, therefore, not a focus in cost/performance optimization.

3.3 CTAS

Having observed a high overall runtime impact by `CREATE TABLE AS` (`CTAS`) statements in customer workloads, we want to investigate those queries in this section, analyze how these tables are being used, and outline possible strategies on how to optimize databases for this usage pattern. As previously shown, around ≈1.9% of queries are `CTAS` queries, but those account for ≈16.4% of the load on Redshift. Those tables created by `CTAS` statements are heavily used; roughly ≈40% of queries overall utilize them.

When looking at how often the same `CTAS` table is recreated, we find that most tables (≈66%) are created only once within the span of a week. This leaves ≈34% of `CTAS` tables that are recreated multiple times and we can analyze if they are also used similarly. By looking at subsequent queries on a `CTAS` table before it is being recreated, we find that ≈95% of them are being used in the exact same way each time. This would indicate that `CTAS` tables are often used in prepared pipelines, and, most often, to precompute a result that can then be retrieved quick and easy. The latter is evidenced by the fact that around ≈80% of `CTAS` tables are only used by one query each.

Table 3: Query Runtimes: *Number of queries and their total runtimes grouped into various runtime buckets.*

Time bucket	% of queries			% of sum(runtime)		
	Fleet	TPC-H	-DS	Fleet	TPC-H	-DS
(0s, 10ms]	13.7	0	0	0.01	0	0
(10ms, 100ms]	48.3	0	0	0.4	0	0
(100ms, 1s]	24.9	0	22	2.3	0	2
(1s, 10s]	9.9	27	59	7.3	3	19
(10s, 1min]	2.2	55	13	13.3	30	25
(1min, 10min]	0.86	8	5	35.7	66	55
(10min, 1h]	8e-2	0	0	25.2	0	0
(1h, 10h]	8e-3	0	0	14.3	0	0
>=10h	9e-5	0	0	1.6	0	0

3.4 Query Complexity

Due to obvious privacy concerns, Redshift operators have no facilities to analyze customer queries directly. Therefore, we only report several indicator metrics to approximate the structure and size of queries being run in real world workloads compared to TPC-H/DS. We can see that there is a heavy tail in query text length (which serves as an indicator for query complexity): while most queries are rather simple, there are some heavy hitters.

Table 4: Query Plan Nodes: *Shows the average number of plan operators per query in the Redshift fleet compared to TPC-H/DS. The first row tells us that queries with a runtime of up to 1s have an average of 1.63 table scans.*

	Redshift Fleet				TPC	
	(0s, 1s)	[1s, 60s)	[60s, ∞)	Total	-H	-DS
Scan	1.63	4.15	6.51	2.03	4.00	9.17
HashJoin	0.33	1.30	2.19	0.48	2.54	5.46
MergeJoin	0.04	0.02	0.02	0.04	0.00	0.00
NestLoop	0.01	0.03	0.07	0.01	0.00	0.12
Agg	0.52	0.50	0.72	0.52	1.15	2.11
Sort	0.10	0.23	0.34	0.12	0.73	0.95
Merge	0.08	0.09	0.12	0.08	0.69	0.68
Window	0.03	0.08	0.20	0.04	0.00	0.23
Network	0.15	0.37	0.57	0.18	0.69	1.15

SQL Code Size. First, we analyze the SQL text length of submitted queries in bytes in Table 2. We found, in accordance with earlier studies [35], that customer queries are much longer than the ones from standard benchmarks. Especially the higher percentiles show an extreme long-tailed distribution where Redshift encounters queries of several mega bytes posing significant stress to any database frontend. For instance, the largest `select` queries contain over ≈16 MB of SQL code, which is also the current limit of Redshift [3]. As a comparison, the entire Lord of the Rings trilogy contains about 580 thousand words, which (assuming an average word length of 4.7 in the English language) amounts to only 2.7 MB.

Query Runtimes. Second, we investigate runtimes of queries. Table 3 groups all executed queries on the fleet by their runtime. While most queries are very short running (i.e., less than 100 ms), the vast majority of resources is spent on longer running queries. For instance, while less than 0.1 % of queries run longer than an 1 h, they take up ≈25 % of all resources. Similar to query text, we observe a long tailed distribution for runtimes. In contrast, the query runtime in TPC-H/DS is a narrow spectrum: depending on the scale factor, all queries roughly run in the same order of magnitude of time. Outliers, in either direction, are less skewed. The long running queries stress the query scheduler and the concurrency control of the database. In addition, Redshift's MVCC implementation needs to keep increasingly old snapshots around for the long running query, while data is constantly being updated in the system.

Query Plan Nodes. Third, we compare the average number of used query operator types through the fleet with TPC-H/DS (cf. Table 4). For short running queries, we can see a much higher ratio of scan and join operators in TPC-H/DS compared to the fleet, because most short running queries only touch a limited amount of tables. Similar trends can observed for most operators, showing that the TPC-H/DS benchmarks do not capture the small, but constant and ever-present load of short running queries in actual data warehouses. These short running queries need to co-exist with long running queries in the database, posing significant challenges to scheduling and concurrency control mechanisms. For the buckets with longer running queries in the fleet (one second to a minute and upward), the usages of operators match more closely.

Table 5: Scan Types: *Number of occurrences of different filter types for base table scans.*

Category	Fleet	TPC-H	TPC-DS
subquery	2.1%	1.1%	0.6%
string matching	2.1%	8.0%	0.1%
function call	4.9%	2.3%	0.1%
conjunction	32.7%	21.6%	37.2%
disjunction	8.4%	9.0%	9.4%
no predicate	20.9%	51.1%	35.7%
simple	41.4%	20.5%	21.9%

3.5 Scan Types

We classified base table scans run in the Redshift fleet into different categories based on the filters applied in Table 5. Scans categorized as *subquery* use the result of a subquery in their filter. *String matching* scans perform string pattern matching such as `SIMILAR TO`, `LIKE`, or regex. *Function call* scans contain a call of a system or user-defined function. *conjunction* and *disjunction* scans contain `AND` or `OR` respectively. *No predicate* scans are full scans without any filters, and *simple* scans are those that match none of the previous categories. Note that the percentages do not sum up to 100% as the categories are non-exclusive. Overall, the distribution of scan categories in the Redshift fleet is not terribly different from the benchmarks. TPC-H does far more string matching than TPC-DS and the Redshift fleet. Redshift customers use functions more often than the benchmarks. Interestingly, the *no predicate* and *simple* scans make up 60-70% which shows the importance of optimized scan code.

4 HOW DO THE WORKLOADS DIFFER?

In this section, we go beyond the scope of individual queries and investigate how queries are combined into workloads as well as the characteristics of those workloads. We find that workloads vary a lot within individual days, but are rather stable, in general, over longer periods of time. Another interesting finding is how often individual filters, query structures, and even the exact same queries repeat. These findings present huge opportunities for workload prediction and caching.

4.1 Query Elasticity (daily)

Individual Clusters. In this section, we analyze how workloads vary over time. Figure 1 shows the daily load aggregated over all clusters (labeled: `All`) as well as three individual, hand-picked clusters (labeled: `C1` to `C3`) with different load patterns over a one-month period (November 2023). The first depicted cluster `C1` shows a very stable workload, with almost no variation between days. The second cluster `C2` shows a very common workload pattern where the cluster has a high amount of work on weekdays and lower one on the weekend (the Saturdays of the depicted month are labeled on the horizontal axis). The third cluster `C3` shows a more chaotic workload (those are less common). Lastly, when looking at the combined load of all clusters `All`, we observe that most small fluctuations average out and result in a rather stable load, with slight dips on the weekends.

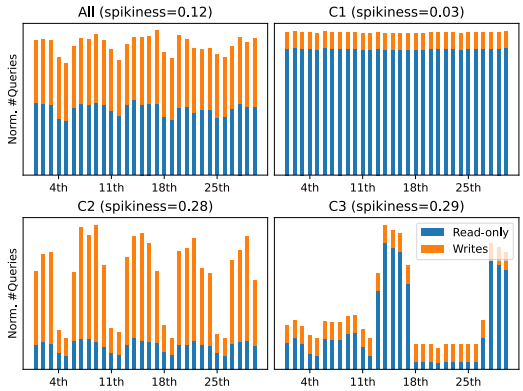


Figure 1: Query Elasticity (daily): *Shows the daily load (approximated by query count) for the entire fleet (All) and three representative clusters (C1-C3). Each plot's title shows the spikiness of the workload, measured as the sum of the root mean square errors between the current and previous day.*

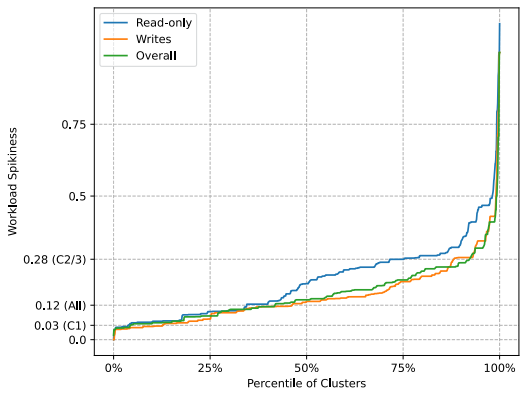


Figure 2: CDF Workload Spikiness (daily): *Spikiness (cf. Figure 1) distribution over all clusters in the Redshift fleet. The vertical axis marks the hand-picked clusters from Figure 1 as a point of reference.*

Spikiness Factor. In order to extend this analysis from individual clusters to the entire Redshift fleet, we introduce a workload *spikiness* factor: a single number to capture how much a workload fluctuates over time. It is calculated as the sum of the root mean square errors between the load of a particular day and its preceding day. Each depicted workload in Figure 1 shows its spikiness in its title for comparison: The higher the spikiness factor, the more the load varies throughout the month.

Fleet Behavior. In the next figure (Figure 2), we calculate the spikiness for all clusters in the fleet and plot the CDF. We marked the clusters `C1`-`C3` from the previous figure (Figure 1) as points of reference on the vertical axis. We can thus observe that most clusters in the fleet (75 %) of workloads are more stable than `C2`/`C3`. This demonstrates the potential of serverless approaches such as Redshift Serverless [5] or Google BigQuery [14]. However, to minimize database administration overhead, more automated ad-hoc scaling mechanisms [23] are needed.

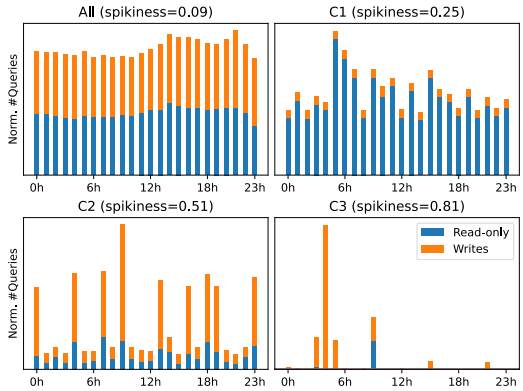


Figure 3: Workload Elasticity (hourly): *Show the average load per hour for the entire fleet (All) and three representative clusters (C1-C3). Each plot's title shows the spikiness of the workload, measured as the sum of the root mean square errors between the current and previous hour.*

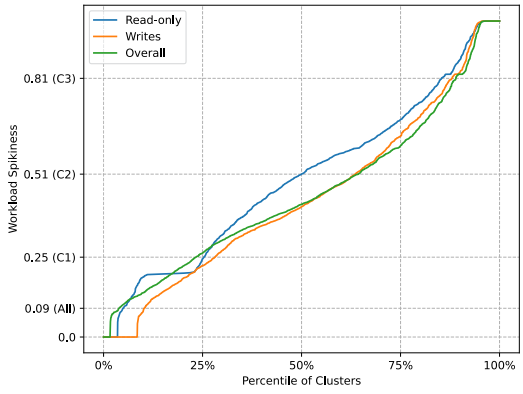


Figure 4: CDF Workload Spikiness (hourly): *Spikiness (cf. Figure 3) distribution over the Redshift fleet. The vertical axis marks the hand-picked clusters from Figure 3 as a point of reference.*

4.2 Query Elasticity (hourly)

Individual Spikiness. As shown in the previous section, the load on a day to day basis is rather stable, with predictable drops on the weekend. In Figure 3 we show the intra-day load for all clusters combined (labeled: `All`) and three individual clusters (labeled: `C1` to `C3`, different clusters as in Section 4.1) with different load patterns. The load per hour is calculated as the number of queries that were run in this particular hour (other metrics like memory used or runtime show a similar distribution). In contrast to the day to day load, we find that the intra-day load can vary significantly in individual clusters. Even so, it is relatively stable and constant across the fleet (`All`). Observe how the last depicted cluster (`C3`) has a huge spike around 4am, but relatively low load for the remainder of the day. The first cluster (`C1`) has a comparatively constant load throughout the day.

Fleet Spikiness. In Figure 4, we show this spikiness factor (as introduced in Section 4.1 but for hours instead of days) across all clusters throughout the Redshift fleet. As a point of reference, the

Table 6: Query Runtime and Memory Distribution

	p50	p90	p99	p99.9
Runtime [s]	0.1	1.8	26.1	203.0
Memory [GB]	0.6	3.6	21.4	250.6

spikiness of the aggregated cluster load (`All`) and the three hand-picked clusters (`C1` - `C3`) from Figure 3 are marked on the y-axis of Figure 4. We can see that the load patterns vary significantly across the fleet:

While ≈50 % are less spiky than `C2`, there are more than ≈20 % that are even more spiky than `C3`. Therefore, we conclude that some workloads can be well and easily served by a static cluster, but many require the dynamic, scalable deployment of the cloud. For cost optimality, it is imperative that the compute resource of a cluster in a cloud analytics platform can be dynamically adjusted to the actual load of the cluster, as in [23].

TPC Spikiness. Neither of the standard TPC-H/DS workloads have a varying load pattern over the day: The work is constant throughout the test period and usually only lasts a few hours at most. However, the automatic elasticity through concurrency scaling [5, 14, 37] or more advanced AI-driven scaling methods [23] became central to all cloud data warehouses to provide the best performance at low cost and reporting the best throughput as demanded by TPC benchmarks is no longer adequate. The recently introduced Cloud Analytics Workload (CAB) [34] simulates such a scenario.

4.3 Workload Tail-Distribution

As a general pattern throughout our experiments, we have seen very large outliers in distributions in essentially all dimensions. In this section, we show those for workloads including query runtime, memory consumption, and daily number of queries.

Table 6 shows the distribution for query runtime and memory. In the median, we observe a runtime of 100 ms with a maximum execution time of several days, and a 2030x difference between the median and the p99.9 value. The memory distribution is similarly skewed with a 418x difference between the median and the p99.9 value, and a maximum value in the hundreds of TB.

Such strong tail-skewed distributions confirm an open system and a lot of variety in the complexity of individual queries, as opposed to a closed system such as TPC-H/DS benchmarks. Interestingly, the most important customers often operate in the tail as they tend to also be the largest customers, which push the system.

4.4 Repeating Queries

Repetitiveness Definition. To characterize how repetitive a workload is, we investigate three similarity metrics: how often do scan filters repeat, how often does the entire query repeat, and how often does the structure of the query, i.e., the query template repeat. Each time, repetitiveness is defined as the number of repeats divided by the total number of observed filters / queries / templates. Suppose, for example, there are 5 distinct queries $q1, \ldots, q5$. $q1$ was issued three times, $q2$ was issued 2 times and the rest were each issued once. This means we saw a total of 8 queries with two repetitions for $q1$ and one repetition for $q2$. The repetition rate therefore works out to $\frac{3}{8} = 37.5\%$.

We calculate repetitiveness rates on top of cryptographically secure one-way hashed filter/query text strings. Note that semantically irrelevant elements of the text, e.g., newlines, multiple whitespaces and comments, are removed before hashing in all three metrics. Both repeating filters and queries count *exact* repeats, i.e., all literals are the same, while repeating query structures also remove literals before hashing. For example, the following queries would count as repeats for query structure but not for queries or filters.

```
SELECT SUM((table_a.amount > 3)::int)
FROM table_a JOIN table_b USING (id)
WHERE table_a.column_date BETWEEN '20230101' AND '20230131'
  AND table_b.name = 'gollum'

SELECT SUM((table_a.amount > 42)::int)
FROM table_a JOIN table_b USING (id)
WHERE table_a.column_date BETWEEN '20230107' AND '20230111'
  AND table_b.name = 'gollum'
```

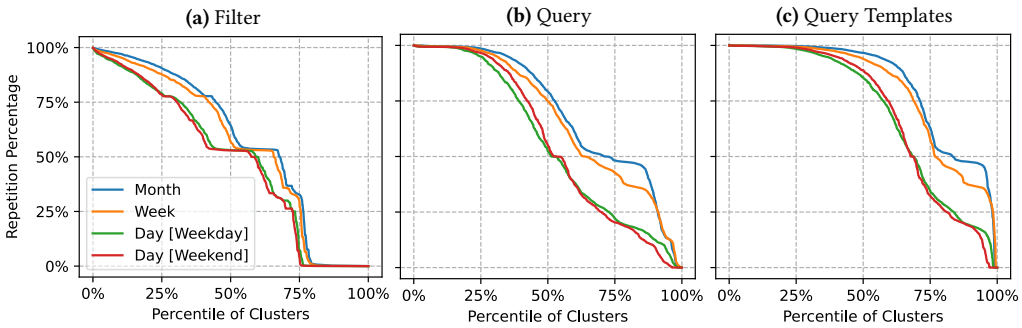
Each query has two table scans. `table_b` is predicated with `name = 'gollum'` in both cases, hence this filter would count as repeating. The filter on `column_date` differs, i.e., is not considered repeating.

Repetitiveness. We show the repetitiveness of scan filters (Figure 5a), entire queries (Figure 5b), and query structures (Figure 5c) against percentile of clusters in Figure 5. The CDF plots show one line per time window, ranging from individual days to an entire month. The close resemblance between the three graphs shows that query and filter repetitions are not just a weekly or monthly pattern. For instance, we can see that on ≈60 % of Redshift clusters up to ≈50 % of scan filters repeat over an entire month or week, while the same is true for ≈50 % of clusters on a single day (Figure 5a). Further, almost ≈25 % of clusters show almost no filter repetition while queries themselves virtually always repeat to some degree.

This artifact of customer workloads is not well represented in TPC-H and TPC-DS, as zero queries are repeating in a single run. On the flip side, query structures always repeat across runs since both workloads comprise of the same queries with randomly instantiated filter condition parameters (once per run).

We therefore also measured filter repetition rates over a number of consecutive runs for both benchmarks. Results are presented in Figure 5d. For TPC-H, 10 to 100 runs are required to reach the ≈50 % filter repetition rate seen for more than 60 % of real world clusters (2 to 5 runs for TPC-DS).

Repeating Query Runtimes. Thus far, these results indicate an opportunity for employing caching on various levels, and we will discuss some ideas and already deployed techniques within Redshift in Section 4.6. However, to benefit from caching, we must first consider the runtimes of the repeating queries, as caching longer-running queries is likely more lucrative. Therefore, we grouped repeating queries by their maximum observed runtime across repetitions and plot the respective repetition rates over the percentile of clusters in Figure 6. We chose the maximum observed runtime per query to eliminate caching effects and to ensure that all repetitions of a particular query end up in the same group. This yields a couple of interesting insights. First, both long and short running



(d) TPC Filters

Runs	TPC-H	TPC-DS
1	7.7%	22.8%
2	20.5%	39.6%
5	30.7%	54.9%
10	38.5%	64.1%
100	65.9%	81.3%

Figure 5: Repetition: *CDF plots and stats about filter, query and query template repetition in the real world vs. filter repetition after a consecutive number of runs in TPC.*

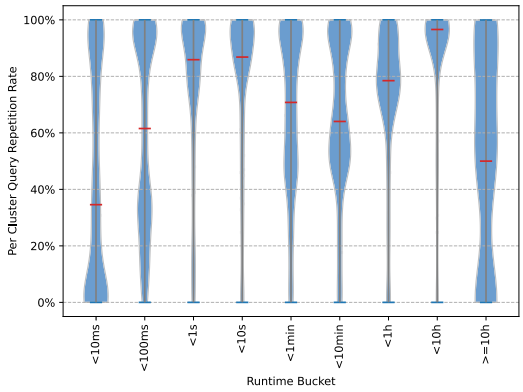


Figure 6: Repeated Queries by Runtime: *PDF of the percentage of repeating queries over clusters, grouped into different runtime buckets. The maximum encountered runtime of a query is used to assign repeats to their buckets.*

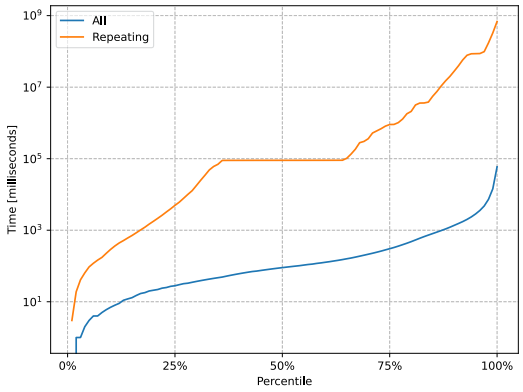


Figure 7: Median Downtime Distribution: *Downtime between all select statements and repeats of the same select statement.*

queries are repeating. While ultra-short ones (<100 ms) seem to either all repeat or never repeat at all, short queries (<10 s) heavily repeat on most clusters. Long running queries (>1 h) almost always repeat on all clusters. Those might be regular transformation or analytical tasks and are likely picked more carefully due to their resource intensiveness. There are only a limited amount of sample points in the last bucket (>10 h) (cf. Table 3), which leads to varied results. Overall, we conclude that longer-running queries do indeed repeat and, therefore, we can establish that caching strategies have a significant performance potential.

Caching Requirements. The timing between repeating queries is important for caching, since long intervals between repeats increase the likelihood of cache staleness or eviction. In Figure 7 we show the distribution of median downtime between a) all select queries and b) repeats of the same query. For more than 50 % of repeating queries, the median time between repeats is just under 100 s. The longest median time between repeats on a cluster we observed is just below 12 days.

This indicates that we should try to avoid invalidating cached results until a (rather short) timeout has passed. Since we rarely observe more median downtime than a second between arbitrary selects, i.e., there is a high cache pollution potential, a standard least-recently-used (LRU) policy with a small cache size might evict results before their query repeats shortly thereafter.

4.5 Common Table Sets

Having analyzed the repetition rate of query templates, we go one step further and investigate common table sets. While workloads often contain thousands of queries over hundreds of tables, we find that often the number of sets of distinct tables used in the queries is very limited. We analyzed this in ten randomly picked clusters and found that within more than 50 000 queries there are only an average of 23.5 distinct sets of tables being used (max=108). This result indicates that it might be beneficial to pre-calculate join indexes for these few important joins [11].

4.6 Dealing with Repeating Queries

Redshift already deploys a result cache that is capable of capitalizing on the high query repetition rate. While highly effective, we analyze some outstanding opportunities to further improve the hit rate and why certain queries are hard to cache. Overall the cache has a hit rate of ≈20 %. There are two main reasons for cache misses:

Table 7: Column Data Type Distribution: *Shows the proportion of columns that use a particular data type and the proportion of columns marked as Predicate Columns by Redshift.*

Data Type	Stored Columns			Predicate Columns		
	Fleet	TPC-H	TPC-DS	Fleet	TPC-H	TPC-DS
varchar	52.1%	21.3%	11.2%	53.8%	15.6%	9.8%
numeric(P, S)	10.2%	14.8%	18.8%	7.0%	11.1%	8.8%
integer	9.1%	19.7%	44.5%	11.6%	22.2%	60.3%
bigint	7.0%	11.5%	-	9.4%	15.6%	-
timestamp w/o tz	6.2%	-	-	5.8%	-	-
double	4.5%	-	-	2.2%	-	-
boolean	3.9%	-	-	1.5%	-	-
date	2.2%	6.5%	2.6%	3.2%	8.9%	0.5%
smallint	2.1%	-	-	2.3%	-	-
char(N)	1.7%	26.2%	22.8%	2.4%	26.7%	20.6%
float	0.4%	-	-	0.2%	-	-
timestamp w/ tz	0.4%	-	-	0.4%	-	-

Table 8: Table Size Distribution: *Shows the distribution of an estimate of the number of rows stored in non-empty tables across the fleet and for the TPC benchmarks.*

Rowcount Bucket	Fleet	TPC-H	TPC-DS
$(10^0, 10^1]$	12.3%	12.5%	-
$(10^1, 10^2]$	10.8%	12.5%	25.0%
$(10^2, 10^3]$	12.0%	-	-
$(10^3, 10^4]$	12.9%	-	16.6%
$(10^4, 10^5]$	14.2%	-	12.5%
$(10^5, 10^6]$	13.5%	-	4.2%
$(10^6, 10^7]$	11.4%	-	4.2%
$(10^7, 10^8]$	7.6%	12.5%	8.3%
$(10^8, 10^9]$	3.5%	25.0%	12.5%
$\geq 10^9$	1.8%	37.5%	16.7%

compulsory misses and inability of queries to be cached. Compulsory misses constitute ≈20 % of all queries because only ≈80 % of queries do repeat and thus can be cached. Out of the repeating queries, ≈60 % are cache misses because when such a query arrived last, it was not cached. ≈30 % of queries in this category are not populated because of a concurrent transaction which updated one or more tables used in the read query. ≈24 % could not be cached because their transactions were read/write. If a write to a table comes after read query, then its results cannot be cached. ≈24 % used an non-immutable functions like `CURRENT_DATE`. ≈14 % access system tables to read metrics and catalog. ≈4 % could not be cached as they read an external table for which modifications cannot be tracked. ≈2 % could not be cached because its result set was too large. Lastly, ≈2 % could not be cached because another query's result was concurrently being cached.

5 HOW DOES THE DATA DIFFER?

Having looked at queries and workloads, we use this section to investigate statistics about data: What data types are being used, how skewed is customer data, and what storage formats are used on external storage.

5.1 Data Types

Fleet Data Types. Table 7 compares how frequently different data types are used in Redshift compared to TPC-H/DS. Notably, time zones appear to be a rarely used features occurring only in ≈0.4 % of all columns. Even more advanced time-related types (namely, `times` and `intervals`) only account for ≈0.02 % in total. Further, we see occasional usage of complex data types, such as `json` (≈0.1 %) or `geography` (≈0.02 %). Redshift stores its `numerics` in either 64 or 128 bit registers, depending on the defined precision. Here, we find that ≈50 % are stored as 64 bits.

TPC-H Data Types. When comparing to TPC-H and TPC-DS, it stands out that the benchmarks do not use any of the temporal data types except `date`. Instead, TPC-DS is emulating a timestamp with the columns `t_time`, `t_hour`, `t_minute`, `t_second`, and `t_am_pm` in a `time_dim` dimension table, where each row represents one second. This table also stores additional information that could be directly inferred, such as `t_shift`, `t_sub_shift`, and `t_meal_time`. In contrast, Redshift customers use the `timestamp w/o tz` type quite frequently (6.2%). More generally, TPC-H and TPC-DS use only a few basic types, notably many fixed sized `char` columns. Redshift customers, on the other hand, use a more diverse set of data types. As previously shown [35], variable-sized text columns are most prominent in the fleet. Notably, the TPC-H/DS do not use the `boolean` data type, which makes up 3.9% of the customers' columns. Instead, they emulate it with a `char(1)` (e.g., `l_linestatus`).

Predicate Types. Table 7 also shows (on the right) how often columns of a certain data types are marked as *predicate* columns by Redshift. Predicate columns are columns that have been used for comparisons, i.e., in a filter, join or group condition in previous queries or as (part of) a sort or distribution key [1]. Those columns make up 10% of columns in the fleet and are prioritized by Redshift when it comes to collecting statistics. Analyzing the types of these predicate columns shows that the previously mentioned more complex data types are used even less often as comparison inputs. Otherwise, the predicate columns are fairly similar to the stored data distribution.

5.2 Table Sizes and Change

Table Size. Table 8 shows how many rows are stored in tables throughout the fleet and in the TPC benchmarking datasets with our chosen scale factor (3000). Continuing the trend of long-tailed distributions, there is a small number of tables in Redshift with trillions of rows, while the majority is much more reasonably sized with only millions of rows. In fact, most tables have less than a million rows and the vast majority (98 %) has less than a billion rows. Much of this data is small enough such that it can be cached or replicated, opening up opportunities for optimized data layouts and distribution schemes. The tables in the TPC datasets have a few small statically-sized tables and bigger tables that grow with the scale factor. Obviously, there are many orders of magnitude

Table 9: Table Growth Trends: *We group tables into the categories shrinking, stable (i.e. number of inserted and deleted rows is roughly equal), and growing tables.*

Growth Behavior	Fraction of Fleet Tables
No insert/delete	86.6%
Yes insert/delete	13.4%
↪ Stable	76.70%
↪ Growing	23.24%
↪ Shrinking	0.06%

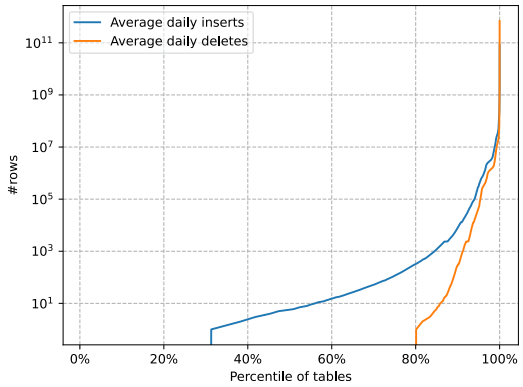


Figure 8: Insert Quantity.: *CDF over daily inserts and deletes.*

less tables in the benchmark data sets than in the Redshift fleet. We observe that TPC-DS's table size distribution is closer to the Redshift fleet's than TPC-H. By using multiple scale factors in a single benchmark one could likely get close to the fleet distribution, but a benchmark with a configurable table size distribution would be desirable.

Table Growth. Table 9 classifies tables into a number of categories depending on their growth behavior. We first distinguish the ones not receiving any updates (first row in table) and the ones receiving changes (second row). Next, we break down the latter category into stable (where the number of inserts and deletes roughly neutralize each other), growing, and shrinking tables. There are only some outlier tables that were actually shrinking during the monitored time frame. Overall the vast majority of tables are untouched and a potential easy candidate for optimizations such as indexing, sorting, replication, or compression. These results are consistent with TPC-H, where 25 % of tables receive changes. Both of these tables (`lineitem` and `orders`) are stable. There are no growing tables in TPC-H (similar for TPC-DS).

Daily Updates. Figure 8 shows the percentage of tables (horizontal axis) that insert/delete a certain amount of rows per day (vertical axis). This plot excludes tables which do not experience any deletes or inserts. We can see that there are far more inserts than deletes. ≈80% of tables receiving inserts did not receive deletes the same day, and ≈30% of tables receiving deletes did not receive inserts the same day.

5.3 Data Distribution

In this section, we analyze the distribution of table data by gathering information through the column statistics that Redshift's query planner is using. It uses a combination of Most Frequent Values (MCVs), compressed histogram (*compressed* meaning it does not contain the MCVs' values), null fraction, and Number of Distinct Values (NDVs).

Most Common Value Skew. First, we calculate the fraction of rows that are occupied by the (up to) twenty MCVs, the single most frequent value, and null values per column for the fleet, TPC-H, and TPC-DS. We investigate the null fraction in this paragraph as we observe that `null` is often a "hidden" extra MCV. We show the results of this experiment aggregated over all tables in Figure 9. The plot is over the percentile of columns (horizontal axis) and shows the CDF of the respective fraction (vertical axis). For instance, in the MCV plot (left most figure), we can see that at most 25 % of rows consist of MCVs for 40 % of fleet columns. Overall, TPC-H has the least amount of skew, followed by TPC-DS, and then the fleet for all three metrics. In addition, the MCV curves are a lot steeper for TPC-H/DS than the fleet. This indicates that the columns in the synthetic benchmarks, especially TPC-H, are more extreme: either consisting of uniformly distributed values or a small set of unique values (e.g., `n_name` or `l_shipmode`). The amount of skew in the fleet data is more even. Skew is a double edged sword for data processing systems, as it can be beneficial (e.g., caching, distributed join processing), but when not expected or dealt with it is also capable of causing large bottlenecks (e.g., multi-threading, partitioning, cardinality estimation). It is also remarkable that the frequency of the single most frequent value (center plot) often makes up a large portion of the sum of the MCV frequencies for both the benchmarking data sets as well as the fleet.

Null Fraction. Null fractions are depicted in the right most plot. Similar to MCVs, there are generally more in the fleet than in the synthetic benchmarks. TPC-H has no `null`s at all and TPC-DS only has a limited amount. Interestingly, TPC-DS appears to have a high number of columns with a low frequency of null values, something that we do not see in the fleet. In contrast, the fleet has (1) more columns with null values and (2) those columns also have higher fraction of null values (some with >99% of `null` values).

Histogram Skew. Next, we analyze the skew of the remaining data (excluding `null` values and MCVs) by looking at the histograms present in Redshifts' statistics. We measure the mean Q-error [21] over columns between the actual histogram bucket width and the histogram bucket width if the data was distributed uniformly ($\frac{\text{histogram max}-\text{histogram min}}{\#\text{buckets}}$). Q-Error is the ratio between uniform and true histogram bucket width. The lowest Q-Error value is 1, which would imply a uniform distribution. The greater the Q-Error, the more skewed the data represented by the histogram is. The resulting data is shown in Figure 10. The trend of the fleet data being significantly more skewed than the two benchmark's data continues here. Observe how TPC-H is very uniformly distributed with a maximum Q-Error of two. 50 % of TPC-DS tables are uniformly distributed, and the maximum Q-Error here is 500. This is in stark contrast to the tables in the fleet where the Q-Errors are

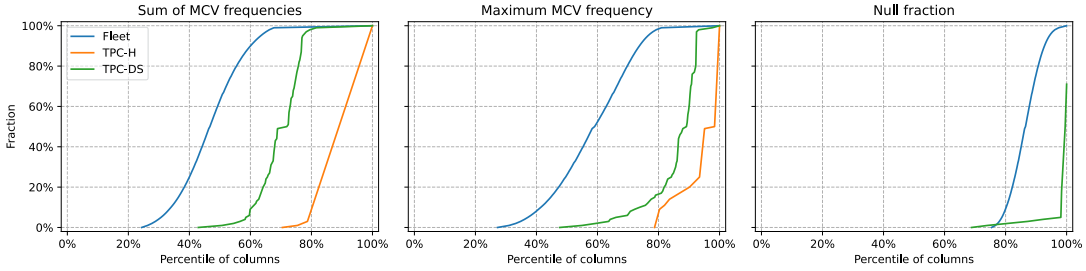


Figure 9: Most Common Value Skew & Null Fraction: *Shows the fraction of the (up to) twenty most common values (left), the fraction of the most common value (middle), the fraction of null values (right) over all columns in the fleet, TPC-H, and TPC-DS.*

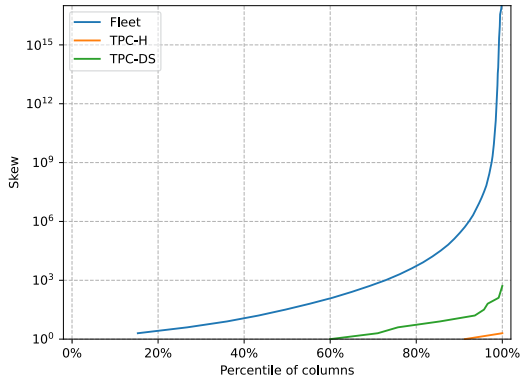


Figure 10: Histogram Skew: *Shows the mean Q-Error between a uniform distribution and a histogram of a column for all columns.*

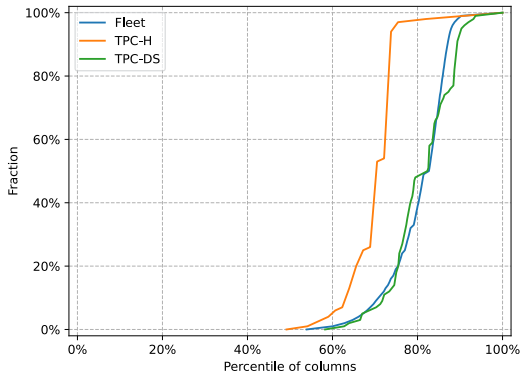


Figure 11: Distinct Value Ratio: *Shows the distribution of the distinct value ratio for all columns.*

much higher, reaching up to 4.2×10^{239}. These experiments confirm the widely assumed fact that actual data is seldom uniformly distributed.

Table Uniqueness. Figure 11 shows the distribution of the distinct value fraction ($\frac{NDV}{\#rows}$) for all columns excluding `null` values. 0 % means that all values of the depicted column hold the same value, while 100 % means that all values in the column are different. We notice an overall trend shared by TPC-H/DS and the fleet that most columns are not very unique. Further, only ≈10 % of columns have

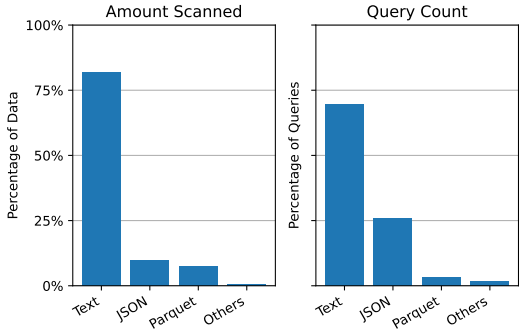


Figure 12: Storage Format on S3: *The format of external table data, either read by* `COPY` *or within an external table scan.*

extremely high cardinality (distinct value fraction close to 100 %) indicating that the column is entirely or mostly unique.

5.4 External Storage Formats

Lastly, we analyze external data sources for `COPY` and scans over S3 data. The results are depicted in Figure 12. We show both relative bytes scanned (left) and relative query count (right) for different external data formats. While column-optimized formats such as parquet have been around for a while and are well integrated into Redshift, they appear to be seldom used compared to the more established and human-readable formats, such as CSV and JSON. It appears that, here, the ease of use and interoperability of CSV and JSON outweighs the better performance of parquet for customers. Half of the queries reading from external sources target compressed data, 99 % of which is stored in `gzip` format.

6 REDSET

To facilitate future research, we publish a new dataset of anonymized query logs: the *Redset*[1]. It contains metadata for user queries from 200 serverless and provisioned clusters (each) over a three-month period in 2024. We sampled clusters from our fleet such that the published dataset contains representatives for the various levels of observed busyness. This metric is computed as an equally weighted sum of the number of user queries on the cluster n_c and their total execution time e_c, normalized w.r.t. the maximum number of user queries / total execution time out of all cluster in the fleet:

[1] https://github.com/amazon-science/redset

$$\text{busyness}(c) = \frac{n_c}{\max N} + \frac{e_c}{\max E}$$

Each row in Redset represents one query execution. For obvious customer privacy reasons, the dataset only comprises statistics. The columns published in this release contain general information (unique cluster-, user-, database-, and query id), timing information (arrival time, compilation-, queuing-, and execution- duration), query details (abort status, command type like SELECT, COPY, ..., number of [permanent, external, and system-] tables accessed, and the ids of permanent tables written to and read from), I/O information (number of bytes scanned/spilled), operator information (number of joins, scans, and aggregations in the plan), and cluster information for provisioned (number of nodes in the cluster).

With Redset we want to specifically encourage drawing further conclusions about real world workloads, conducting research on ML-based techniques such as query prediction models and workload forecasting and creating novel benchmarks that closer mimic the observed real-world workloads. We believe this dataset can help with the latter, e.g., because it contains arrival timestamps for each query. Compared to CAB [34], this permits creating more realistic benchmarks which exhibit similar spikes in activity compared to real workloads instead of randomly approximating arrival times. We plan to refresh and expand the dataset's features over time.

7 CONCLUSIONS

As we have shown in this paper real analytical workloads differ vastly from our current TPC-H/DS benchmarks. As systems start to adapt more aggressively to the workload and data, standardized benchmarks like TPC-H/DS become less and less suited for evaluating the actually performance data warehouses provide to their users. For example, Redshift's Multidimensional Data Layouts [10] leverage the fact that many scans and predicates repeat by organizing the storage layout based on the usage patterns. Compared to traditional techniques (e.g., single column sort-keys or z-order encoding) it can be orders of magnitudes faster for real workload, but on TPC-H/DS the impact is much more limited. Similarly, Redshift's Automated Materialized Views (Auto-MV) [5] takes advantage of the fact that many queries are repeating beyond what is observed in TPC-H/DS. Hence, the database which is the fasted on TPC-H/DS is not necessarily the fastest for actual customer workloads. Similarly, if TPC-H/DS are used to prioritize what techniques to implement, it will not necessarily be the best prioritization to improve actual customer workloads. In the following, we summarize our findings and differences to TPC-H/DS in more detail.

7.1 On Queries

Analytics = Updates. We showed that a significant portion of work in Redshift is spent on updates. This shows that even data in analytical warehouses can not be considered static or unchanging. Therefore, it is important that this aspect is better reflected in benchmarks, and proposed data structures or algorithms for warehouses should consider updates. For benchmarking, TPC-H/DS should be used with its data freshness functions, at the very least. Unfortunately, updates in TPC-H are difficult to use: Each insert/delete requires a unique set of delete keys/new tuples (usually loaded from a CSV file). Further, performing the TPC-H's data freshness functions requires to reload significant portions of the database before the next benchmark can be executed. Future benchmarks should consider writes an essential part and make them easy to use to ensure they will actually be used and [34] already represents a step in the right directions.

Heavy Tails. As a general trend throughout this paper we have shown that the distribution on query runtime, SQL statement length, and most distributions over aspects of data processing have very long and heavy tails. This means that only reporting the 99th percentile, would often leave out critical edge cases (1 in 100). Instead, an industry-ready system needs to support more than the 99th (or even 99.999th) percentile, as those points in a distribution are often a critical use case. For instance, we showed that most queries in Redshift have a very short run time (in fact, more than 50 % of queries finish in less than 100 ms). In fact, the p99 percentile of queries finish within 1 min. Those only account for roughly a fifth of the overall query runtime, however.

7.2 On Workloads

Elasticity. We analyzed the varying load over time and found that there often is a weekly pattern: Within a cluster, the load on weekdays is somewhat stable with a drop on the weekend. However, within a single day there are often severe spikes in the load from hour to hour. Using these patterns can be very beneficial in a serverless setting to automatically adjust resources to customer demands [23]. Using the dataset published as part of this paper, researchers have the opportunity to experiment with these patterns of varying load over time. The large presence of short query workload (Section 3.4) and spikiness (Section 4.2) motivates AutoWLM's [27] policy to prioritize short query execution over long queries using techniques like SQA in order to keep their latency low. This policy of AutoWLM cannot be motivated by TPC-H/DS.

Repeating Queries. We showed that queries are often repeating within a cluster (80 % of queries repeat in 50 % of clusters). This presents great opportunities for caching, but requires accurate tuning of cache size, retention time, and fine grained invalidation in case of updates. Future work should also investigate how writes interact with these repeating queries. Repeating scans can be exploited via indexing [28] or storage techniques like Redshift Multidimensional Data Layouts [10]. Note that TPC-H/DS does not capture these patterns when executed once. Multiple reruns can approximate similar repetition rates, however, it is questionable how representative this would really be.

7.3 On Data

Types. Variable length strings are the most dominant datatype in Redshift, begging the question how many of those strings are true strings versus other types (numbers or timestamps) stored as strings. According to [35], string columns often contain boolean or non-existent enumeration types such as gender or airplane codes. Either way, efficient string storage [7, 22] and processing (such as late materialization) is needed.

Skew. We showed that skewed data distributions, especially ones with a small number of heavy hitters are prominent in customer tables. This can be beneficial for column encodings (e.g. run-length or dictionary encoding), distributed query execution [26], or storage

layouts [10, 12]. Nevertheless, skew can also present a challenge for join ordering or multi-threading if not detected or dealt with.
External Data. For external data sources, **CSV** is clearly the dominant format in Redshift. Other column-oriented and/or binary formats have not replaced this simple text-based format yet. Consequently, techniques for automatic schema inference, on-the-fly statistics [8, 25], and fast, distributed CSV [16] parsing are still vital to efficiently support real world data lake queries.

7.4 On Benchmarking

We draw two conclusions with regards to benchmarking from the presented data: (1) A cloud data warehouses consists of a large number of components, such as scaling, data ingestion and migration, multi-tenancy, query execution, privacy, and more. Given this high dimensional feature space, we argue for the need of dedicated benchmarks for the individual features. We summarize other proposed benchmarks in related work (cf. Section 8). (2) Assuming we build benchmarks for all aspects of a cloud data warehouse, we still need a TPC-*-like benchmark for the database's core component – the query execution engine.

Given the large spectrum of clusters with heavily tailed distributions over essentially all attributes (cluster size, data size, query count, query complexity, etc.), a single benchmark can only evaluate the system against one point in this distribution. Therefore, we propose to work towards a benchmark generator capable of instantiating benchmarks for a number of different points in the distribution to be able to evaluate different usage scenarios.

8 RELATED WORK

In an ever evolving field such as databases, we always need to adjust the benchmarks to accurately reflect the latest workload demands [24, 30]. As benchmarking and datasets are essential in database research, there has been a lot of work on this topic as summarized by an earlier study [20]. With the move to the cloud, new demands [6] have emerged. In this section, we highlight some recent and influential work in this area and compare it to ours.
Snowset. In 2020 Snowflake published the "Snowset" [36] – a dataset containing information about ≈69 million queries that ran on their system throughout a two week period. Alongside they published an in-depth analysis on of the Snowset [37]. While the study was done with a systems/network perspective, they also provided valuable insights for the database community into cloud data warehouse workloads. Most notably, they reported the high ratio of write queries in analytical workloads and the oscillating pattern of query arrival times. Interestingly, they found this pattern to be most prominent for read-only queries, while write queries were much more constant over time. They further reported a high fluctuation in the workload demands of individual customers, while the overall resource demand of all customers together did not vary more than 2x over a day in various dimensions, such as network, CPU, and memory. In contrast to this paper, their analysis was limited to the data that they published in the Snowset. While this allows for full reproducibility of their results, we go a step further by digging into more details of our internal the telemetry. Doing so allows us to confirm many of their results, and add additional insights (e.g., query repetition rate, filter expression complexity or data skew).

Cloud Analytics Benchmark. Later, in 2022, the Snowset was picked up by the database community and analyzed through a database lens [34]. The authors compared the published workload in the Snowset to TPC-H and found many differences. First and foremost, they reported on the long-tails when it comes to query resource consumption (in terms of runtime, memory, CPU, database size etc.), the multi-tenancy of cloud databases with vastly varying customers, and the elasticity of workloads over time. In addition, they used those insights to design a new benchmark for analytical cloud data warehouses – the Cloud Analytics Benchmark (CAB). The benchmark builds on TPC-H and models multiple tenants that issue queries with commonly observed arrival times over multiple hours. The goal of the benchmark is to minimize query latencies for each individual customer, while minimizing the overall cost of the system under test (i.e., the cloud data warehouse). Thus, the benchmark captures how well a cloud data warehouse can adopt to elastic workloads of multiple different tenants at the same time. In contrast, our paper builds on internal Redshift fleet data that is more detailed and captures more aspects than the publicly available Snowset. We are again able to confirm many results of this study while adding more insights into cloud data warehouse workloads like the adoption of data lakes, rates at which tables grow time, and statistics about table data (skew, uniqueness, most common values).
Other Cloud Analysis Works. Most recently, beyond the Snowset and CAB, Leis et al. [19] proposed a model to dynamically determine the optimal environment for a given workload. Further, the move of many workloads to the cloud has changed the workloads' demands on a data warehouse system significantly, as described by Binnig et al. [6]. They set forward a list of demands for cloud systems such as scalability and varying load patterns in 2009, many of which are still valid and could be confirmed in our analysis. In another body of work, Tableau [35] published statistics on their datasets in comparison with TPC-H/DS. We confirm their findings in our experiments and provide further insights. They highlight the long-tailed distribution of query text length, number of joined relations, and the omnipresence of text-based column types. Further, in [18] the authors performed a multi-year study where they offered a database-as-a-service front end to other scientists. They found that even for non SQL experts the complexity of queries quickly reached a high level and that the life time of datasets was shorter than in classical workloads. We observed a similar trend in the usage of `CTAS` expressions, which suggests an interactive usage of the data warehouse with intermediate results. Further, there have been several proposals for new benchmarks. Some focus on the ability of the system to be scaled [13, 17, 33], which is crucial to adjust to changing workloads within a day, but less of an issue for longer time periods (Section 4.2). Going one step further, Poggi et al. [24] propose a benchmark that tests how well systems adopt to varying workloads, which requires scalability as a building block.

ACKNOWLEDGMENTS

We extend our thanks to the Redshift team members Mohammed Al-Kateb, Matt Abrams, Naresh Chainani, George Erickson, Bruce McGaughy, Davide Pagano, Ippokratis Pandis, and Rahul Pathak, all of whom played a part in making this paper possible.

REFERENCES

[1] [n.d.]. ANALYZE - Amazon Redshift — docs.aws.amazon.com. https://docs.aws.amazon.com/redshift/latest/dg/r_ANALYZE.html. [Accessed 2023-11-27].

[2] Amazon. [n.d.]. Amazon Redshift continues its price-performance leadership. https://aws.amazon.com/blogs/big-data/amazon-redshift-continues-its-price-performance-leadership/. [Accessed 2023-12-14].

[3] Amazon. [n.d.]. Amazon Redshift SQL. https://docs.aws.amazon.com/redshift/latest/dg/c_redshift-sql.html. [Accessed 2023-12-18].

[4] Amazon. [n.d.]. What is Zero ETL? https://aws.amazon.com/what-is/zero-etl/. [Accessed 2024-1-26].

[5] Nikos Armenatzoglou, Sanuj Basu, Naga Bhanoori, Mengchu Cai, Naresh Chainani, Kiran Chinta, Venkatraman Govindaraju, TJ Green, Monish Gupta, Sebastian Hillig, Eric Hotinger, Yan Leshinksy, Jintian Liang, Michael McCreedy, Fabian Nagel, Ippokratis Pandis, Panos Parchas, Rahul Pathak, Orestis Polychroniou, Foyzur Rahman, Gaurav Saxena, Gokul Soundararajan, Sriram Subramanian, and Doug Terry. 2022. Amazon Redshift re-invented. In *SIGMOD/PODS 2022*. https://www.amazon.science/publications/amazon-redshift-re-invented

[6] Carsten Binnig, Donald Kossmann, Tim Kraska, and Simon Loesing. 2009. How is the weather tomorrow?: towards a benchmark for the cloud. In *DBTest*.

[7] Peter A. Boncz, Thomas Neumann, and Viktor Leis. 2020. FSST: Fast Random Access String Compression. *VLDB* (2020).

[8] Graham Cormode and Ke Yi. 2020. *Small summaries for big data*.

[9] Databricks. [n.d.]. Databricks Sets Official Data Warehousing Performance Record. https://www.databricks.com/blog/2021/11/02/databricks-sets-official-data-warehousing-performance-record.html. [Accessed 2023-12-14].

[10] Jialin Ding, Matt Abrams, Sanghita Bandyopadhyay, Luciano Di Palma, Yanzhu Ji, Davide Pagano, Gopal Paliwal, Panos Parchas, Pascal Pfeil, Orestis Polychroniou, Gaurav Saxena, Aamer Shah, Amina Voloder, Sherry Xiao, Davis Zhang, and Tim Kraska. 2024. Automated multidimensional data layouts in Amazon Redshift. In *SIGMOD/PODS 2024*. https://www.amazon.science/publications/automated-multidimensional-data-layouts-in-amazon-redshift

[11] Jialin Ding, Ryan Marcus, Andreas Kipf, Vikram Nathan, Aniruddha Nrusimha, Kapil Vaidya, Alexander van Renen, and Tim Kraska. 2022. SageDB: An Instance-Optimized Data Analytics System. *VLDB* (2022).

[12] Jialin Ding, Vikram Nathan, Mohammad Alizadeh, and Tim Kraska. 2020. Tsunami: A Learned Multi-dimensional Index for Correlated Data and Skewed Workloads. *VLDB* (2020).

[13] Michael Ferdman, Almutaz Adileh, Yusuf Onur Koçberber, Stavros Volos, Mohammad Alisafaee, Djordje Jevdjic, Cansu Kaynak, Adrian Daniel Popescu, Anastasia Ailamaki, and Babak Falsafi. 2012. Clearing the clouds: a study of emerging scale-out workloads on modern hardware. In *ASPLOS*.

[14] Sérgio Fernandes and Jorge Bernardino. 2015. What is BigQuery?. In *International Database Engineering & Applications Symposium*.

[15] Fivetran. [n.d.]. Cloud Data Warehouse Benchmark. https://www.fivetran.com/blog/warehouse-benchmark. [Accessed 2023-12-14].

[16] Chang Ge, Yinan Li, Eric Eilebrecht, Badrish Chandramouli, and Donald Kossmann. 2019. Speculative Distributed CSV Data Parsing for Big Data Analytics. In *SIGMOD*.

[17] Kai Hwang, Xiaoying Bai, Yue Shi, Muyang Li, Wen-Guang Chen, and Yongwei Wu. 2016. Cloud Performance Modeling with Benchmark Evaluation of Elastic Scaling Strategies. *IEEE Trans. Parallel Distributed Syst.* (2016).

[18] Shrainik Jain, Dominik Moritz, Daniel Halperin, Bill Howe, and Ed Lazowska. 2016. SQLShare: Results from a Multi-Year SQL-as-a-Service Experiment. In *SIGMOD*.

[19] Viktor Leis and Maximilian Kuschewski. 2021. Towards Cost-Optimal Query Processing in the Cloud. *VLDB* (2021).

[20] Zheng Li, Liam O'Brien, He Zhang, and Rainbow Cai. 2012. On a Catalogue of Metrics for Evaluating Commercial Cloud Services. In *GRID*.

[21] Guido Moerkotte, Thomas Neumann, and Gabriele Steidl. 2009. Preventing bad plans by bounding the impact of cardinality estimation errors. *VLDB* (2009).

[22] Ingo Müller, Cornelius Ratsch, and Franz Färber. 2014. Adaptive String Dictionary Compression in In-Memory Column-Store Database Systems. In *EDBT*.

[23] Vikram Nathan, Vikramank Singh, Zhengchun Liu, Mohammad Rahman, Andreas Kipf, Dominik Horn, Davide Pagano, Gaurav Saxena, Balakrishnan (Murali) Narayanaswamy, and Tim Kraska. 2024. Intelligent scaling in Amazon Redshift. In *SIGMOD/PODS 2024*. https://www.amazon.science/publications/intelligent-scaling-in-amazon-redshift

[24] Nicolás Poggi, Víctor Cuevas-Vicenttín, Josep Lluis Berral, Thomas Fenech, Gonzalo Gómez, Davide Brini, Alejandro Montero, David Carrera, Umar Farooq Minhas, José A. Blakeley, Donald Kossmann, Raghu Ramakrishnan, and Clemens A. Szyperski. 2019. Benchmarking Elastic Cloud Big Data Services Under SLA Constraints. In *TPCTC*.

[25] Alice Rey, Michael Freitag, and Thomas Neumann. 2023. Seamless Integration of Parquet Files into Data Processing. In *BTW*.

[26] Wolf Rödiger, Sam Idicula, Alfons Kemper, and Thomas Neumann. 2016. Flow-Join: Adaptive skew handling for distributed joins over high-speed networks. In *ICDE*.

[27] Gaurav Saxena, Mohammad Arifur Rahman, Naresh Chainani, Chunbin Lin, George Caragea, Fahim Chowdhury, Ryan Marcus, Tim Kraska, Ippokratis Pandis, and Balakrishnan (Murali) Narayanaswamy. 2023. Auto-WLM: Machine learning enhanced workload management in Amazon Redshift. In *SIGMOD/PODS 2023*. https://www.amazon.science/publications/auto-wlm-machine-learning-enhanced-workload-management-in-amazon-redshift

[28] Tobias Schmidt, Andreas Kipf, Dominik Horn, Gaurav Saxena, and Tim Kraska. 2024. Predicate caching: Query-driven secondary indexing for cloud data warehouses. In *SIGMOD/PODS 2024*. https://www.amazon.science/publications/predicate-caching-query-driven-secondary-indexing-for-cloud-data-warehouses

[29] Snowflake. [n.d.]. Industry Benchmarks and Competing with Integrity. https://www.snowflake.com/blog/industry-benchmarks-and-competing-with-integrity. [Accessed 2023-12-14].

[30] Junjay Tan, Thanaa Ghanem, Matthew Perron, Xiangyao Yu, Michael Stonebraker, David J. DeWitt, Marco Serafini, Ashraf Aboulnaga, and Tim Kraska. 2019. Choosing A Cloud DBMS: Architectures and Tradeoffs. *PVLDB* (2019).

[31] Transaction Processing Performance Council (TPC). 2021. TPC BENCHMARK™ DS Standard Specification Version 3.2.0. https://www.tpc.org/TPC_Documents_Current_Versions/pdf/TPC-DS_v3.2.0.pdf. [Accessed 2023-11-28].

[32] Transaction Processing Performance Council (TPC). 2022. TPC BENCHMARK™ H Standard Specification Revision 3.0.1. https://www.tpc.org/TPC_Documents_Current_Versions/pdf/TPC-H_v3.0.1.pdf. [Accessed 2023-11-28].

[33] Wei-Tek Tsai, Yu Huang, and Qihong Shao. 2011. Testing the scalability of SaaS applications. In *SOCA*.

[34] Alexander van Renen and Viktor Leis. 2023. Cloud Analytics Benchmark. *VLDB* (2023).

[35] 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*.

[36] Midhul Vuppalapati. [n.d.]. Snowflake dataset containing statistics for 70 million queries over 14 day period. https://github.com/resource-disaggregation/snowset. [Accessed 2022-04-15].

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