Cloud Analytics Benchmark

Alexander van Renen
Friedrich-Alexander-Universität Erlangen-Nürnberg
alexander.van.renen@fau.de

Viktor Leis
Technische Universität München
leis@in.tum.de

ABSTRACT

The cloud facilitates the transition to a service-oriented perspective. This affects cloud-native data management in general, and data analytics in particular. Instead of managing a multi-node database cluster on-premise, end users simply send queries to a managed cloud data warehouse and receive results. While this is obviously very attractive for end users, database system architects still have to engineer systems for this new service model. There are currently many competing architectures ranging from self-hosted (Presto, PostgreSQL), over managed (Snowflake, Amazon Redshift) to query-as-a-service (Amazon Athena, Google BigQuery) offerings. Benchmarking these architectural approaches is currently difficult, and it is not even clear what the metrics for a comparison should be.

To overcome these challenges, we first analyze a real-world query trace from Snowflake and compare its properties to that of TPC-H and TPC-DS. Doing so, we identify important differences that distinguish traditional benchmarks from real-world cloud data warehouse workloads. Based on this analysis, we propose the Cloud Analytics Benchmark (CAB). By incorporating workload fluctuations and multi-tenancy, CAB allows evaluating different designs in terms of user-centered metrics such as cost and performance.

PVLDB Reference Format:
Alexander van Renen and Viktor Leis. Cloud Analytics Benchmark. PVLDB, 16(6): 1413 - 1425, 2023.
doi:10.14778/3583140.3583156

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/alexandervanrenen/cab.

1 INTRODUCTION

Cloud is Taking Over. Over the previous decade, the cloud [4] has transitioned from a promising niche topic to the standard way of deploying systems. In particular, cloud-native data warehousing and analytics has become a major growth area, with systems such as Snowflake, Amazon Redshift, Databricks, and Google BigQuery enjoying great success. However, it is probably fair to say that most of the research and development behind these cloud-native systems occurred in industry, and most academic database systems research still does not consider the implications of the epochal transition to the cloud.

Cloud is Different. The cloud environment differs from on premise deployments in some important respects. Most cloud-native data warehouse services run on virtual compute instances hosted by public cloud providers such as Amazon EC2, Microsoft Azure virtual machines, or Google Compute Engine (GCE). Systems like Lambada [25] or Starling [26] build on top of function-as-a-service offerings. Either way, these resources can be dynamically scaled to intermittent workloads in very short time periods (minutes or even seconds). In principle, a perfectly-scalable and elastic database system could use twice the compute resources to cut the query time in half – thus escaping the traditional computer science tradeoff between hardware cost and runtime. Obviously, such a perfectly-scaling data warehouse is theoretical, but it illustrates the point that in the cloud we have to rethink our understanding of the relation between performance, compute resources, and cost [12]. The traditional goal of optimizing runtime while assuming a fixed, given hardware is not sufficient anymore.

Cloud Architectures. The cloud landscape has led to a variety of different data warehouse architectures [31] and pricing models. These approaches can be classified along the following spectrum: (1) Customers can rent compute resources (i.e., virtual machines) themselves and host their database systems of choice. (2) Companies are offering hosted data warehouse services, such as Snowflake or Amazon Redshift. (3) There are fully-managed services like Google BigQuery and Amazon Athena that fully embrace the query-as-a-service model. Generally speaking, there is a clear trend towards more integrated "serverless" services.

Benchmarking Challenges. Despite the significant differences between the on-premise world and the cloud, academics [31] as well as practitioners usually rely on pre-cloud era benchmarks like TPC-H or TPC-DS when comparing cloud analytics systems [6, 14, 36]. While those benchmarks have served us well in the past, they fall short in capturing the opportunities and challenges of cloud data warehouses [2, 13, 24]. They evaluate the performance of a query engine for a particular workload on a given hardware, enabling us to determine which cloud data warehouse has the superior query engine. While a fast query engine is definitely desirable, it hardly provides any direct customer value in an environment where hardware resource can virtually be scaled indefinitely. Thus, we need to view the query engine as one component of a larger system: Unlike with an on-premise deployment, a cloud data warehouse service should manage the hardware – ideally automatically adapting to the current workload. Moreover, cloud data warehouses are not used by a single customer, but by multiple customers at the same time (multi-tenancy).

Snowflake Dataset to the Rescue. Cloud-native database systems research in academia remains difficult and therefore still relatively rare. One reason is that it is not clear how to operationalize the abstract cloud challenges suggested above into a concrete research program. In other words, since most academics do not have access to real-world cloud workloads, research remains stuck in the on-premise world. The goal of this paper is to overcome this problem.

Proceedings of the VLDB Endowment, Vol. 16, No. 6 ISSN 2150-8097.
doi:10.14778/3583140.3583156

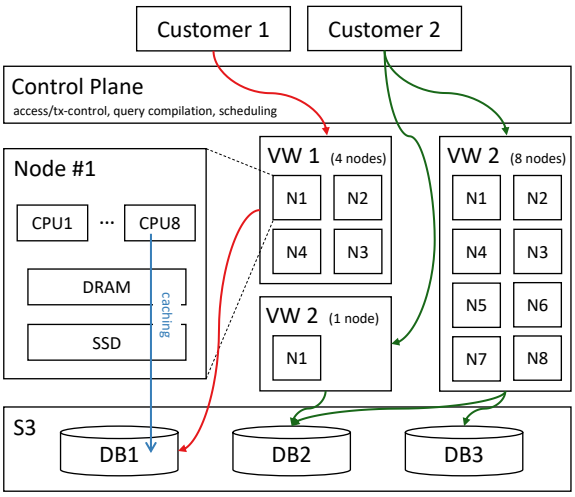

Figure 1: Snowflake Architecture: *Customers can rent any number of virtual warehouses (VWs) to process databases stored in S3.*

To do this, we rely on real-world telemetry dataset released by Snowflake, one of the most prominent cloud data warehouses. The workload contains performance statistics for all Snowflake queries over a 2-week period in 2018.

Contributions. The first contribution of the paper is to analyze the Snowflake dataset and compare it with well-known OLAP benchmarks from the TPC. We identify important workload differences, including a higher ratio of data manipulation language (DML) statements and a much higher variance of table and query sizes. Based on this analysis, our second contribution is a new benchmark for a cloud data warehousing called *Cloud Analytics Benchmark (CAB)*. For pragmatical reasons, CAB is derived from TPC-H, but adds crucial dimensions such as elasticity, multi-tenancy, and the cloud-specific query characteristics. We propose CAB as a new cloud data warehouse benchmark focusing on monetary workload cost and query latency. From an academic perspective, we argue that such a benchmark is essential to (1) enable a fair and realistic comparison of existing cloud data warehouses and (2) provide a cloud-specific optimization goal for designing new systems and architectures.

2 BACKGROUND: SNOWFLAKE WORKLOAD

In this section, we describe the architecture of the Snowflake data warehouse platform and give an overview of the Snowset – a dataset with detailed profiling information of the Snowflake platform.

2.1 Snowflake

Snowflake is a data warehousing product offering analytical data management as a service [7]. Customers can rent *virtual warehouses* (VWs), which are dedicated compute clusters that run on public cloud instances. Snowflake is able to run on three of the major public cloud vendors (Amazon's AWS, Google's GCP, Microsoft's Azure). The number of nodes used in a VW can be configured in powers of two by the customer and earlier work [20] suggests that on AWS VWs currently consist of the relatively small `c5d.2xlarge` instances (8 vCPUs, 16 GB DRAM, and one 200 GB NVMe SSD).

Snowflake deploys a disaggregated storage architecture (cf. Figure 1) where all persistent data is stored on Amazon's Simple Storage Service (S3), a scalable object store accessed through a REST-like web API, and loaded into VWs on demand for query processing. The persistent data is organized in logical databases and Snowflake allows for an *n*:*m* mapping between VWs and databases (in fact, database access can be shared between customers and is even sold on an internal marketplace). To improve query performance, each node uses its local DRAM and SSD as a write-through cache for persistent data. This ephemeral cache is also used for intermediate data and can spill over to S3 (data is never written to other nodes). To further reduce the amount of data that needs to be fetched from S3, Snowflake offers zone map-based pruning for table scans. In addition, a partitioning key can be specified for each table, which can reduce network communication during distributed joins and aggregations. While partitioning can improve query performance, we opted to forego manual optimization in our experiments. In Snowflake, all meta information about databases, partitions, and queries are stored in an OLTP database (FoundationDB), which is also drives transaction isolation (read committed) and atomicity [8].

2.2 Snowset

In 2020, Snowflake released the so called *Snowset* dataset [34], which contains statistics about ≈ 69 million queries that were run during a two-week period on the Snowflake platform on AWS in February/March 2018. To the best of our knowledge, Snowset is the first publicly available dataset on real-world, large-scale data warehousing. It also offers profound insights into a major cloud-native software-as-a-service offering. Due to obvious privacy concerns, the Snowset does not contain any user data or SQL code, but many per-query runtime metrics as summarized in following table:

Meta	queryId, warehouseId, databaseId
Platform	warehouseSize, coresPerServer, usedServers
Timing	start, end, duration, scheduling compilation, controlPlane, execution
Data	{read, write} × {persistent, interm.} × {bytes, requests} × {S3, cache}
CPU Time	user, system
Profiling	{read, write} × {persistent, interm.} × {s3, cache}
Profiling	CPU × {IDL, busy}, mutex, setup, teardown, …
Profiling	Scan, Join, Agg., Sort, DML, …

A 2020 NSDI paper by Vuppalapati et al. [35] analyzed the Snowset from a networking and systems perspective, deriving interesting resource utilization and management insights. The findings by Vuppalapati et al. relevant for our research are:

► In terms of query counts, the Snowset workload consists of many read/write queries (≈59 %) compared to read-only (≈28 %) and write-only ones (≈13 %). They have classified this surprisingly high amount of read/write queries as ETL-style data transformation jobs that are performed within the data warehouse, due to many of those queries reading roughly as many bytes as they write (i.e., read_bytes ÷ write_bytes = 1±0.1 for ≈61.2 % of read/write queries).

► When looking at query arrival rates over time, the Snowset often exhibits an oscillating pattern, where more read-only queries arrive during working hours on weekdays. This effect is much

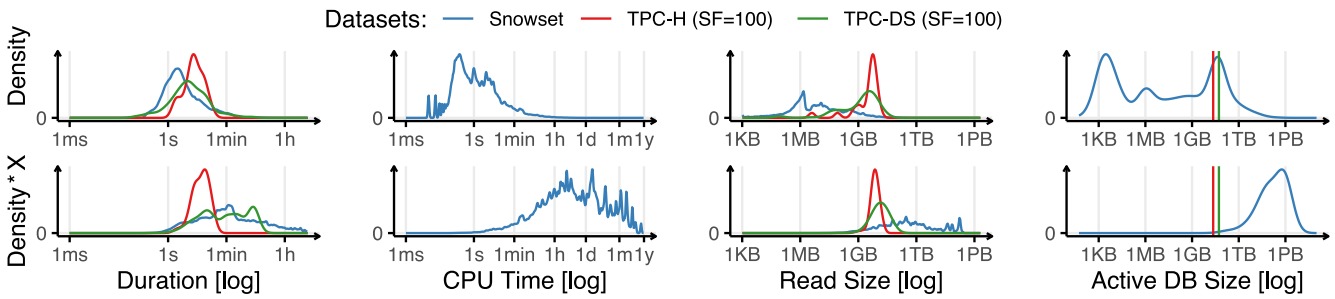

Figure 2: Workload Profile: *First row: density function of the duration, CPU time, and read bytes per query as well as the database size per customer. Second row: weight density function by query importance (its value form the x-axis). Example: in the CPU time plot, we multiply each query by its consumed CPU time (i.e., a query that takes* 100 s *of CPU time is* 100 *times more important than one that only takes* 1 s*).*

less pronounced for read/write queries and non-existent for write-only queries; their arrival rate remain at a constant (noisy) level throughout the week. This suggests that the ETL process is mostly automated while analytical queries are more user driven (manual inputs and tool generated). While this holds for the workload as a whole, our results suggest that all three query types vary over time for individual customers (cf. Section 4).

► They observe a similar behavior for resource utilization: The fluctuating resource demands of individual customers mostly average out in a system-wide view for CPU, Memory, Network TX, and Network RX to an average of 51 %, 19 %, 11 %, and 32 % respectively.

► The average cache-hit ratios is between 60 % and 80 %, even though the cache is much smaller than the database (0.1 % on average), suggesting temporal and spatial skew in the workloads.

In addition to these points, the Snowset has many insights for database researchers that have not yet been mined and discussed in the literature. In the next to sections, we therefore expand this investigation of the Snowset focusing on SQL workload characteristics, multi-tendency aspects, and elasticity concerns.

3 REAL-WORLD DATA WAREHOUSING

The TPC-H and TPC-DS benchmarks are the de facto standard for measuring data warehouse performance [10]. The Snowset gives us the opportunity to compare these artificial workloads with a real-world trace of actual customer workloads. To this end, we measured the TPC-H (sf = 100, size = 100 GB) and TPC-DS (sf = 100, size = 100 GB) benchmark on a commercially available Snowflake instance ("STANDARD" edition) in late 2021. We performed one full run of all TPC-H and TPC-DS queries including the data refresh functions on a freshly loaded dataset without any manual tuning. Throughout this section, we compare our experimental results with the tracing information in the Snowset. We find that in comparison with the TPC workloads, cloud data warehouse workloads have much more variety (in terms of data size, database size, and query duration), are less join heavy (which suggests more denormalized schemas), and perform more complex data manipulation (hinting at an in-data-warehouse ETL process and an incremental work flow, where query results are stored and reused). This section focuses on general data warehousing workload insights, whereas cloud-specific aspects are discussed in Section 4.

3.1 Query Profile

We explore the dataset in a top-down manner, starting with high-level characteristics: In Figure 2, we show the time it took to answer each query (*Duration*), the used CPU resources (*CPU Time*), and the number of bytes that were scanned (*Read Size*). Lastly, we show an approximation of the database sizes (*Active DB Size*). The TPC-H and TPC-DS numbers were measured on a single Snowflake node.

Duration. We can see that even in a data warehouse scenario most queries complete within a few seconds (`median = 2.2 s`). In fact, there are just around 2 M (2.8 %) queries that run longer than one minute and only 6 K (0.0086 %) queries run longer than an hour. This shows that while there are large analytical queries, they are rare. However, as shown in the second row, these long-running queries are very impactful regarding the overall time. In comparison with the TPC-DS and especially the TPC-H numbers, we can see that the real life workload is much more diverse, due to the large number of heterogeneous queries. In addition, the Snowset contains many smaller queries than the TPC workloads, yet the most time is still spent in long-running queries. Lastly, Snowflake queries require at least 100 ms to finish, probably due to scheduling overheads in the control plane. This may indicate that ultra low latency is less of a concern for data warehouse applications.

CPU Time. We define the *CPU time* of a query as the total amount of time that was spent processing on all involved CPU cores to answer it (there are no TPC numbers in this plot, as these numbers are only available in the Snowset). For most queries, the CPU time is even shorter than the duration (`median = 870 ms`), because of intra-query parallelization and the fact that the initialization and scheduling is not part of the processing. However, there is a large number of CPU-intensive analytical queries: Around 4 M (5.9 %) queries require more than a minute of CPU time and 190 K (0.3 %) burn through more than an hour of CPU time. Considering the total CPU resources spent in the Snowflake cluster (second row), we can see that the long-running queries, while less numerous, use up the majority of the resources. This shows that (1) there are very big queries, (2) Snowflake can efficiently scale out, and (3) scale-out is essential for reasonable query response times in a cloud data warehouse environment.

Read Size. Lastly, each query has a `scanBytes` metric, which logs how many bytes the query scanned. It is not specified whether

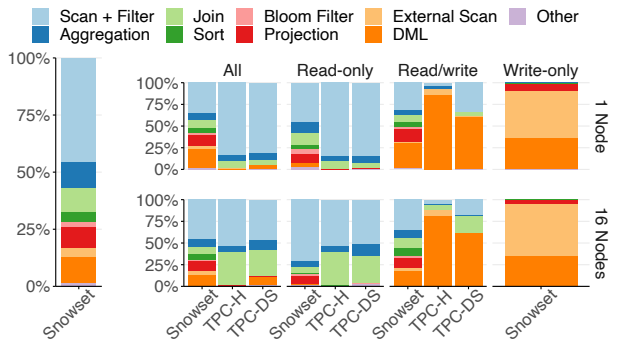

Figure 3: Operator Distribution: *Comparing the profiled operator distribution in the Snowset with TPC-H and TPC-DS (*SF = 100*).*

the number includes compression or pruning. However, it gives a rough estimate for the involved data sizes: Most queries only touch a few megabytes of data (`median` = 5.3 MB), likely due to the many short running queries. But there are also 55 K queries (0.1 %) which read multiple terabyte (≥ 1 TB) of data, with the largest query consuming 453 TB.

Active DB Size. We can use the amount of data read by a query to estimate the size of the database it is running on. When looking at the largest query for each database, we can get a rough picture of the stored datasets in Snowflake: A database where the largest query reads x bytes probably contains at least x bytes. Compared to the TPC workloads, a cloud database service has to accommodate many different database sizes.

3.2 Operator Distribution

All queries in the Snowset have profiling information on how much time was spent in each operator. Given the absence of actual SQL code, we can use this information to approximate how a typical cloud data warehouse workload looks like (cf. Figure 3).

Scans. Overall (left-most column), queries in the Snowset spend around 50 % of their time in scan and filter operators. This is even more pronounced when looking at all read-only queries where the scan and filter operators account for 44.5 % in the Snowset and around 84 % in the TPC benchmarks on a single node cluster. In comparison, Hyrise, a single node columnar in-memory database engine with only light-weight compression, spends less than 10 % of their processing time in scans [10] during the TPC-H benchmark. This is likely due to more heavyweight compression, data encryption [9], and the disaggregated storage architecture of cloud data warehouses where much data has to be read over the network from a remote storage service (e.g., Amazon's S3). Hence, caching and pruning techniques [19, 23] are crucial for performance.

Scale-out Behavior. In the Snowset queries (*All*), the operator distribution on 1 node and on a 16 node cluster is similar (except for a slightly higher DML load on 1 node). In the TPC benchmarks, we observe a much lower scan fraction when moving from a single node (above 80 %) to a 16 node cluster (around 50 %) with a significant performance increase (TPC-H: 179 s →72 s, TPC-DS: 1352 s →354 s). This is likely due to our fixed workload (sf = 100), which appears more suited to a small cluster as it experiences a similar scan fraction as Snowset queries on the same cluster size. We do not observe this effect in the Snowset: suggesting that the workload changes between different cluster sizes, which means that customers adjust the warehouse size depending on their workload size.

Table 1: Query Access Type: *Showing how many and how much time is spend on which types of queries in the Snowset.*

Query Type	Count Ratio	CPU Time Ratio
Read/write	57.6 %	71.6 %
Read-only	29.3 %	23.2 %
Write-only	13.2 %	5.1 %

Aggregation. In addition, joins account for roughly the same amount of time as aggregation in the Snowset independent of the cluster size. In contrast, joins are much more dominant in the distributed TPC benchmarks (which utilize a normalized database schema, which might not be the case in many data science applications). This suggests that either (a) the type of aggregation queries in the Snowset does not scale well (large cardinality) while the ones in the TPC queries do (low cardinality) or (b) the type of join queries in the TPC benchmark do not scale well, possibly due to ineffective bloom filters (which are used more in the Snowset).

Data Manipulation. We observe that, compared to the TPC benchmarks, a larger amount of time is spent on data manipulation in the Snowset. In particular, the single-node configuration spends ≈25 % of its time in the DML operator. When drilling down on update queries (i.e., queries that modify the database: *read/write*), we can observe that those queries in the Snowset have a similar profile than that of read-only queries: A lot of time is spent on scan, join, projection, and aggregation. In contrast, the TPC-H and TPC-DS update queries are mainly performing data manipulation. This suggests that the Snowset update queries are much more complex. In addition, as suggested in the original Snowset publication [35], more than half of all queries in the Snowset perform some write operations in the database (cf. Table 1).

In total, ≈70 % of queries perform updates. One possible explanation would be that data scientists are using a more iterative workflow, where the results of one query are saved and reused in the next query (similar to the interactive nature found in data-frame based data exploration in python).

3.3 Spilling

Next, we investigate larger-than-main-memory queries (i.e., queries that spill to disk or S3) and distributed queries (i.e., queries that use network to exchange data). We show the number of queries (*Query Count*), the used CPU resources (*CPU Time*), and the total duration of the queries (*Duration*):

	Spilling			Net. Exchange
	In-memory	SSD	S3	Network
Query Count	95.5%	4.3%	0.2%	48.1%
CPU Time	55.8%	32.7%	11.6%	95.9%
Duration	80.6%	16.8%	2.7%	67.8%

Table 2: Query Statistics: *Showing key metrics for queries aggregated by the consumed CPU time ("mat" indicated how much data was materialized and "net" is the amount of data exchanged between compute nodes for distributed queries).*

CPU-Time bucket	Count [%]	CPU time [%]	Avg(read) [GB]	Avg(write) [GB]	Avg(mat) [GB]	Any(mat) [%]	Avg(net) [GB]	Any(net) [%]	Avg(time) [s]	Avg(servers) [s]
<1 s	52.42	0.2	0	0	0	0.4	0	30.1	0.8	2.1
<10 s	31.69	1.3	0.1	0	0	6.9	0.01	61.0	2.6	5.8
<1 min	9.97	2.7	0.6	0.1	0.01	12.7	0.07	77.6	6.6	7.6
<10 min	4.65	9.2	4.6	0.3	0.10	9.3	0.85	86.0	23.7	7.7
<1 h	0.98	15.0	26.0	2.2	2.66	27.4	6.56	93.7	105.4	10.2
<10 h	0.25	26.1	159.2	11.7	28.36	49.3	39.47	97.0	439.0	15.3
>=10 h	0.02	45.6	1990.6	215.8	726.24	52.5	497.53	99.4	2717.2	32.2

The left-hand side of the table (labeled *Spilling*) distinguishes three query types: quires that do not spill (*In-memory*), queries that spill only to SSD (*SSD*), and queries that spill to S3 (*S3*). While most queries do not require spilling, out-of-memory processing is still important. While only 4.5 % of queries do not fit into main memory, they account for 44.3 % of used CPU resources. Thus, external sort, join, and aggregation are essential for data warehouses.

The right part of the table (labeled *Net. Exchange*) shows the ratios of queries that exchange intermediate data over the network (i.e., distributed query processing). While half of the queries, do not send data over the network, the vast majority of CPU resources are spent on distributed queries (95 %).

3.4 Query Statistics

Lastly, we can group the Snowset queries by the amount of CPU time they require and investigate respective resource utilization (cf. Table 2). In alignment with Figure 2, we observe that most CPU resource (45.6 %) are spent on very few very large queries. Those also are almost always run on a cluster of machines and read terabytes of data. Interestingly, only around 50 % of them need to materialize intermediate results. However, once such a query starts materializing, it materializes large amounts of intermediate results (hundreds of gigabytes). Suggesting two types of resource intensive queries: with and without the need for intermediate results. While most queries (≈84 %) fall in the smaller buckets with less than 10 s of CPU time, they do not account for a significant amount of used resources: only 1.5 % of the total required CPU time of the workload.

3.5 Lessons Learned

From our analysis of the Snowset, we obtained the following findings and research goals:

► Scan and filter operations are a large portion of the load on the data warehouse (50 %). Techniques for effective pruning, data partitioning, lightweight indexing, compression, and data transfer have the potential to greatly improve performance.

► DML is an essential part of data warehouse applications and goes beyond simple data loading queries. The iterative working style might make very read-optimized data layouts difficult.

► Out-of-memory processing is required in cloud data warehouses: 44.3 % of CPU resources are spent on those queries.

► Distributed queries are frequent and impactful. This makes efficient data exchange algorithms and bloom filters crucial.

4 CLOUD WORKLOAD CHARACTERISTICS

The traditional TPC OLAP benchmarks use a single database of one adjustable given size, fairly similar queries, and are executed in one burst (all queries are run back-to-back). This may be reasonable for testing the performance of a query engine, but it falls short in evaluating a cloud data warehouse *service*: Different customers have varying needs in terms of database size (multi tenancy), query complexity, and query arrival times (elasticity). In addition, individual queries can differ greatly in their hardware requirements: a non-time critical data loading job can be run on a single machine with little memory, a computationally-complex query can use a machine with many CPU cores, and large queries can provision a cluster with high network bandwidth. Unlike a pre-provisioned (on-premise) data warehouse, a cloud data warehouse *service* is able to adjust its hardware resources to current and specific workload demands (cost efficiency). A cloud data warehouse benchmark needs to capture these aspects. Using the Snowset, we illustrate the vast variety in resource demands of different customers over time.

Figure 4 depicts the used CPU hours over a single day (Monday, 22nd Feb 2018) for *All* virtual warehouses (VWs), the five largest VWs in terms of spent CPU hours (*L1-L5*), and five medium-sized VWs (*R1-R5*) that were pseudo randomly selected. In addition, Table 3 shows key statistics of these VWs and a per-query breakdown.

Over the two-week period, all Snowflake customers consumed 5411 CPU hours on average per hour. Assuming a `c5d.2xlarge` instance with 8 CPU cores was used and a perfect utilization (in practice Snowflake reported 51 % [35]), this would require 677 instances running on average. The actual workload, however, is not as constant and varies by as much as 3.4× between the lowest valley and the highest peak (Figure 4: *All*). In addition, when looking at individual customers, we can observe even more drastic changes in the workload over time: Several customers have "spiky" workloads, where queries are only run in short bursts. For example, *L2*, *L3*, and *R1* are only used once a day. Others, like *R3* run jobs on a periodic pattern, like once an hour. This can lead to load spikes over the system. There are also VWs that are always active, but with noisy workload like *L1*, *L4*, or *R2* which can vary by over an order of magnitude throughout the day.

The type of work that is performed in any given warehouse also differs greatly. Some warehouses, like *L2* and *L3* appear to be loading large amounts of data once a day: Each day, they first perform some write-only queries (data load) and then transform the data with some read/write queries. Other warehouses (not shown),

Table 3: Warehouse Usage *for all (*`All`*), the five largest (*`L1-L5`*), and five random (*`R1-R5`*) warehouses for a single day (Monday 22nd Feb 2018). The last row shows the usage for the entire dataset (2 weeks).*

	All queries of the customer					Per query					
	Query count	CPU [h]	Read [GB]	Write [GB]	Network [GB]	CPU [h]	Read [GB]	Write [GB]	Network [GB]	Node count	Duration [s]
L1	26 335	8258	540 925	2094	21 180	0.31	20.5	0.1	0.8	32	17.3
L2	28	5492	35 476	25 863	20	196.15	1267	923.7	0.7	128	815.4
L3	35	3562	44 484	13 404	62 808	101.78	1271	383	1794.5	128	454.4
L4	75 066	2944	74 198	8301	12 990	0.04	1	0.1	0.2	16	13.5
L5	1706	2741	44 224	57	2922	1.61	25.9	0	1.7	32	36.1
R1	2123	1710	81 930	3219	51 430	0.81	38.6	1.5	24.2	64	82.9
R2	10 992	878	8130	9488	209	0.08	0.7	0.9	0	4	26.3
R3	3454	892	7411	7383	8812	0.26	2.1	2.1	2.6	16	23.6
R4	18 897	362	16 583	844	7696	0.02	0.9	0	0.4	4	9.3
R5	100	3	296	10	45	0.03	3	0.1	0.5	2	20
Total Day	138 736	26 842	853 659	70 662	168 111	0.19	6.2	0.5	1.2	17.4	16.5
2 Weeks	69 182 074	1 736 788	98 305 381	8 194 235	22 613 812	0.03	1.4	0.1	0.3	5.6	10.6

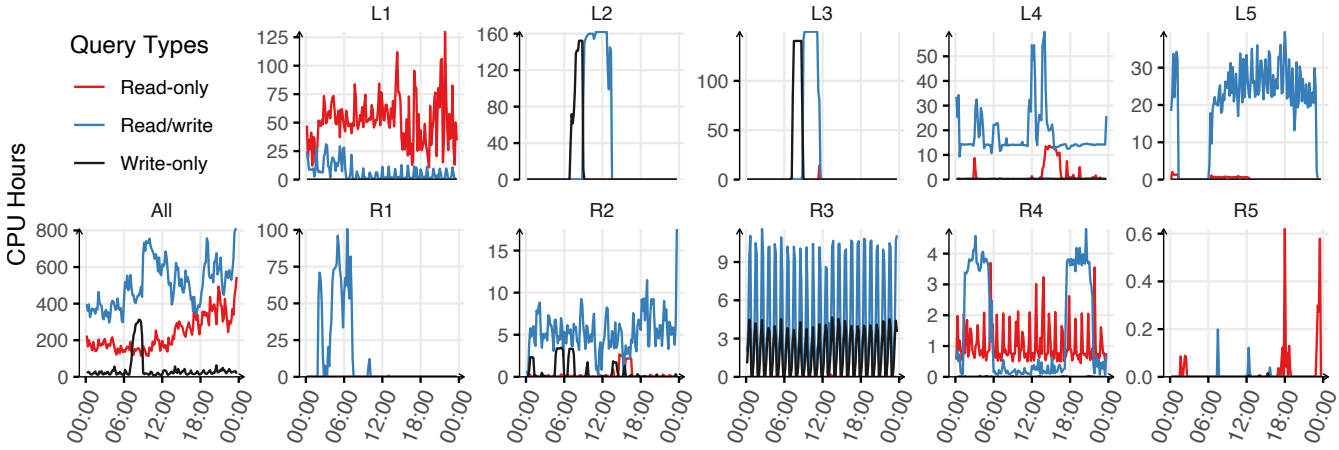

Figure 4: Warehouses over Time: *Showing the activity for all warehouses (All), 5 large warehouses (L1-L5), and 5 random ones (R1-R5). One data point per* 10 min *) for a single day (Monday 22nd Feb 2018).*

connect to the same database and perform analytical (read-only and read-write) work. Further, we can observe warehouses that perform mostly read-only queries (*L1* or *R5*) and others where read/write queries dominate (*L4*, *L5*, and *R1-R3*).

The total amount of work performed per warehouse varies greatly: A small number of large warehouses require tens of thousands of CPU hours, while the majority only uses 100-1000 hours within the two-week period. In addition, all five large warehouses (*L1-L5*) have a comparable number of CPU hours. However, *L2* and *L3* perform only a few huge queries (in terms of CPU time and touched bytes), while the others do thousands of smaller queries.

In summary, the workloads of individual customers are very heterogeneous across all dimensions and vary in intensity over the day. In particular, the workload size, query arrival rate and composition changes. The same warehouses are often used for the same tasks throughout their lifetime.

5 CLOUD ANALYTICS BENCHMARK

As discussed in the previous two sections, cloud data warehouse services face different challenges compared to those of a single on-premise database system. Therefore, we propose the Cloud Analytics Benchmark (CAB) for evaluating multi-tenant data analytics services – focusing on latency and monetary cost of user workloads.

5.1 Benchmark Basics

Based on TPC-H. CAB uses the same database schema as the TPC-H benchmark and queries derived from the original 22 query templates (plus insert + delete streams). We chose TPC-H as a foundation of CAB for the following reasons. First, we are not aware of any real-world data warehousing benchmark, and for obvious reasons the Snowset only includes telemetry data rather than the actual customer data and queries. Second, synthetic benchmarks

such as TPC-H can be scaled to arbitrary sizes. Third, TPC-H is well-known [3] and widely-used for evaluating query engines. With only 22 queries, it is also easier to implement than its larger cousin TPC-DS, which consists of 99 more complex query templates. We hope that these factors will lead to widespread adoption.

Multi-Tenancy. To model multi tenancy, CAB specifies multiple databases with varying scale factors (instead of using a single database with a given scale factor). Each database represents one customer and is independent of other databases (no cross database queries). CAB generates one *query stream* per database, which has to be processed by the System Under Test (SUT). A query stream consists of a specific mix of TPC-H queries and specifies the points of time when each query starts (relative to workload start).

Arrival Times. The arrival times follow typical patterns observed in the Snowset – exhibiting intermittency and therefore stressing the elasticity of the SUT. All query streams are executed in parallel against the SUT. It is the responsibility of the SUT to allocate resources, schedule queries, and return query results. A run of CAB always takes the same amount of time to execute, as queries are not run back-to-back but start at specific times. Hence, CAB simulates the work of a data warehouse service for a certain period (e.g., one hour or one day).

Metrics. The goal of the SUT is to execute each query as fast as possible while minimizing the cost ($) of the run. While there are secondary system metrics such as security, encryption, durability or reliability, the primary performance metrics of CAB are the query latency and total (monetary) workload cost.

Driver. In the following, we detail the data generation, the input parameter selection, and how CAB is to be executed. We provide a software tool to generate the database definitions and query streams. Similar to TPC-H, it is up to the user to implement a driver program that runs the generated queries against their SUT. We provide the driver program for Snowflake, which we use in Section 6, as an example. Note that this benchmark specification is less rigid than, for instance, the one of TPC-H. To ensure a fair competition, we therefore encourage users of CAB to open source their benchmark driver and configuration.

5.2 Benchmark Parameters & Metrics

There are four parameters to specify a CAB run:

(1) Execution time specifies the amount of time in hours that the full run should take. To accurately simulate the elasticity, it should be set to at least 1 hour. For longer experiments up to 24 hour periods can be used.

(2) CPU time is an abstract metric to specify the total amount of CPU work in a run. The specified amount is divided across all involved databases and filled up with TPC-H queries. We calibrated the amount of time per TPC-H query to a single-node single-core Snowflake cluster. While this number is tied to Snowflake (with a specific hardware and software version), it serves as a ballpark number to be able to specify relative compute loads (e.g., a run with 8 CPU hours requires more compute than one with 5).

(3) Data size is the size in terabyte of the entire dataset across all databases. The number of databases is derived from the data size: for small runs with only 1 TB of data 20 databases are used, otherwise CAB always uses 100 databases.

Table 4: Database Configuration Example: *20 generated tenants for* 10 *CPU hours and* 1 TB *of total data size.*

Tenant	Pattern	Size [GB]	Query Count	CPU [min]	Tenant	Pattern	Size [GB]	Query Count	CPU [min]
0	2	1	16	0.0	10	5	13	30	2.1
1	3	1	20	0.0	11	5	17	103	3.1
2	5	1	108	0.3	12	2	23	15	0.2
3	1	1	987	9.2	13	5	31	300	78.5
4	3	2	160	2.8	14	2	43	17	0.5
5	1	3	561	15.2	15	4	59	324	152.4
6	2	4	160	7.0	16	4	85	95	8.1
7	2	5	24	0.0	17	3	125	23	23.0
8	1	7	2406	167.8	18	2	202	8	0.4
9	5	10	48	0.0	19	4	356	103	129.2

(4) SUT Lastly, the system under test (SUT) on which CAB is executed needs to be described. This includes all available information required to reproduce a CAB run on the SUT: Software, hardware, cluster size, versions, editions, and pricing model.

The number of databases is derived from the data size: for small runs with only 1 TB of data 20 databases are used, otherwise CAB always uses 100 databases. For further comparability of benchmark results, we urge users of CAB to scale input parameters using the *CAB-factor*: A positive integer that describes the CPU time and data size as follows:

`cpu_time := CAB_factor` × 10 CPU hours
`data_size := CAB_factor` × 1 TB
`database_count := if(CAB_factor==1): 20 else: 100`

For example: CAB-4 would be a run with 40 CPU hours and 4 TB, which is roughly 2 % of what we observed in the Snowset.

Evaluation Metrics. Once all queries of a CAB run have finished, the run can be considered completed. CAB focuses on two key metrics that are used to compare among systems and configurations: The distribution of query latencies (in the form of a box plot) and the total cost of the CAB run in USD. The setup, schema creation, loading of data, and tear down of resources shall not be included in the cost as the purpose of CAB is measuring the running cost of a cloud data warehouse. While further drill-down experiments are encouraged, we argue that these two metrics are sufficient to capture the most important user concerns, while being simple and generic enough to easily compare CAB runs.

5.3 Database and Query Stream Generation

Database Generation. The CAB generator (CAB-gen) is an open source tool [33] that can be used to generate the input for a benchmark run. As outlined before, a run consists of multiple databases and one query stream for each database. CAB-gen first generates the database sizes using a deterministic log-normal distribution modeled after the one we observed in the Snowset (cf. Figure 2 "Active DB Size"). Given the small number of samples (e.g., 20 or 100), we over-sample from the desired distribution and then take average values to avoid extreme cases and thus make runs with different CAB-factors more comparable. Further, we clamp the distribution to values within two standard derivations. As most of the compute time is spent in the larger databases (cf. Table 2), CAB only

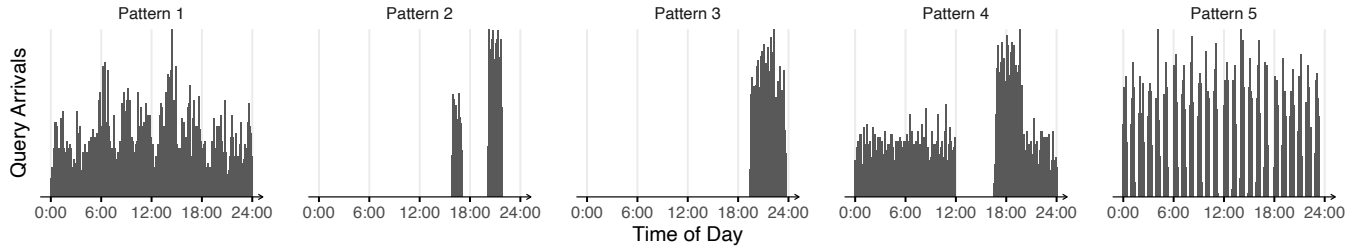

Figure 5: Generated Patterns: *The benchmark generates query arrival times according to one of these five patterns. Each one is randomly generated and has a certain variance (e.g., the number of spikes and their position in pattern 2 can vary).*

generates databases of at least 1 GB. The mean of the log-normal distribution is adjusted so that the total data size over all databases adds up to the specified data size. Moreover, each database is assigned a portion of the total CPU time depending on its data size. The CPU time for a given data size is log-normal distributed and follows the ones we observed in the Snowset. It may vary by several orders of magnitude for a particular database size (as demonstrated in Table 3), but in general larger databases use more CPU time (cf. Table 2). Further details can be found in the source code [33]. We show a sample in Table 4.

Query Stream Generation. Once the CPU time per database is determined, CAB-gen needs to generate query arrival times for each database. To do so, the run is split up into a number of discrete time slots (e.g., 100). Each slot is assigned a certain fraction of the database's CPU time following a larger pattern. To find those patterns we manually categorized the top 100 warehouses in the Snowset into five arrival time patterns as shown in Figure 5. We generate these pattern by combining several random functions over the time window (details can be found in CAB's implementation [33]). Each database is assigned a random pattern independent of its data size: **Pattern 1:** Constant usage with sinus-based variations, possibly representing a user facing application (e.g., busy dashboard). **Pattern 2:** Several short bursts throughout the run. Simulating interactive use and testing the warehouse's ability to quickly spin up and tear down. **Pattern 3:** One large burst, e.g., a daily maintenance job. Due to all the CAB query stream's work happening in this short period this also tests for scalability of the system. **Pattern 4:** A mostly-constant load with negative (12:00) or positive (16:30) outliers. Those outliers are common in the Snowset. They can be caused by user-side daily maintenance and represent an opportunity to save cost in the idle window. **Pattern 5:** A regularly-running job starting at specific times (e.g., full hour), stressing elasticity while being predictable. Repetitive workloads of this pattern could fairly easily be predicted.

Arrival Time Generation. Within each slot, we use an exponential distribution to determine the time between two events (modeling a Poisson point process): $P(x|\lambda) = \lambda e^{-\lambda x}$. Where λ is the expected time between two events. This distribution is commonly used to model the time between two random events in the real world. Compared to uniformly distributed arrival times, this generates more realistic arrival patters with random burst and idle phases. Queries are randomly chosen from the database's query pool (read/write, read-only, or write-only). CAB-gen estimates the time for a given query using the database's data size and the type of query. In addition, to the arrival time, CAB-gen also generates input parameters for the queries.

5.4 Query Pool

CAB uses the same queries as the TPC-H benchmark with a number of modification to adjust it to the findings motivated in Section 3. First, CAB includes the two TPC-H maintenance functions that are supposed to be run after each run of the 22 analytical TPC-H queries. However, in the original TPC-H benchmark these two refresh functions require files containing new tuples and the keys of tuples to be deleted. The inserts and deletes are designed in such a way that the total number of tuples in the database remains constant. This is achieved by leaving gaps in the key space and cycling tuples through those gaps: Out of 32 possible keys only 8 are used (i.e., initially only key 0-8 are used withing the first 32 possible keys). Each refresh function call updates 0.1% of the `orders` and `lineitem` table. Therefore, after 1'000 refresh functions all keys have been increased by 8. Given that all non-key attributes are independent in TPC-H, CAB can simply reuse the data: it first inserts a copy of 0.1% of the data with a key incremented by 8 and then deletes the old ones. This removes the requirement to have insert and update files during benchmark execution, which is an issue due to those files containing potentially 4 times the number of rows of `orders` and `lineitem`.

5.5 Data Loading

To load the databases for a CAB run, the regular TPC-H data generator can be used, which allows generating the tables in smaller, independent chunks. Thus, the load process can easily be done in parallel using multiple machines and should thus scale well for large scale factors. For instance, our sample implementation for Snowflake was able to load 1 TB of data in less than a day using the `c5d.large` AWS EC2 instance, which costs less than 1 $ for 24 hours when using spot pricing. One terabyte of TPC-H data in csv format compresses to roughly 300 GB using the ZSTD algorithm. Storing this on AWS S3 for two days (1 day loading plus 1 day for the benchmark run), costs around 10 cents assuming 5 $ per 1 TB on S3. Therefore, loading and storing of the data should not incur a large amount of cost.

Due to the update queries a benchmark run modifies the database. Hence, after each run, the database needs to be reset to the initial state. This can be done with a simple update query included in the

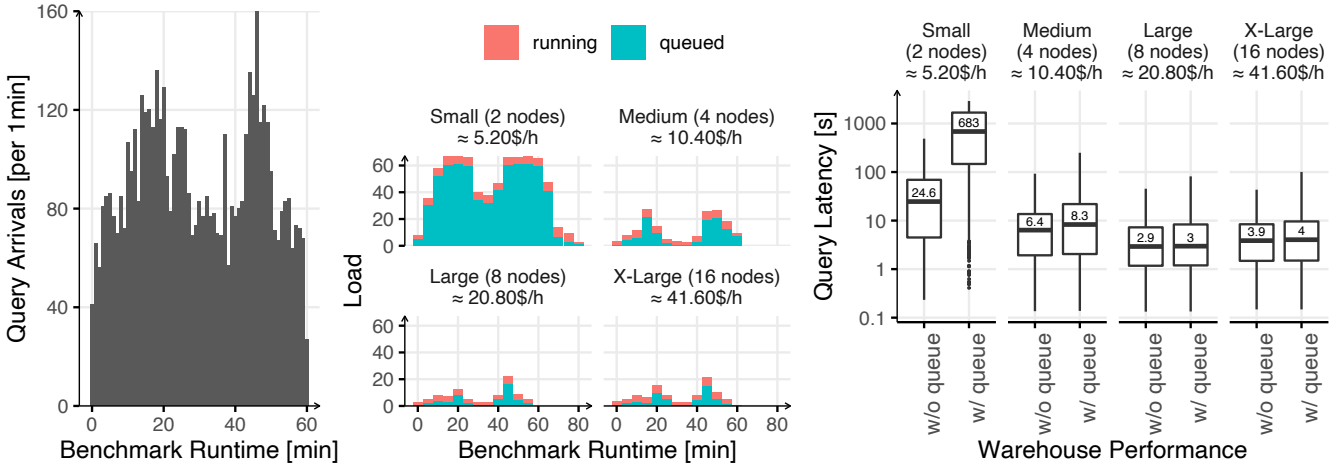

Figure 6: Shared Cluster Experiment: *CAB with* 1 TB *and* 10 *CPU hours (CAB-1) on a single Snowflake VW shared by all databases. We vary the cluster size to determine the required node count for this workload. The numbers in the right most plot represent the median latencies.*

CAB suite. Thus allowing multiple runs without the need to reload the entire dataset.

5.6 Execution of a Run

Before the start of a CAB run, all databases have to be fully loaded (and potentially reset to their initial state). There is no warm up phase: data has to be cold initially. Query templates and query streams may be loaded into memory ahead of time and be prepared on the database server. All query streams have to start at the same time and must be run in parallel. Each query is specified as a triple consisting of the query id, its start time, and parameters. Once the start time is reached, the query (or prepared statement id) with its parameters may be sent to the data warehouse service. Before that time, the query id and parameters have to be treated as unknown (thus simulating user submitted queries). The query can be considered completed once all results are materialized and the query processor has signaled the completion back to the driver. Note that the results do not have to be sent back to the driver, it is sufficient if they are materialized in a driver accessible location. As the CAB is benchmarking a cloud database service, the network communication, queue time on the server, eventual resource allocation and so on are all considered part of the query execution time from the users point of view. Thus, the driver is required to record the time from the query start till the completion for each query.

6 EVALUATION

Goals. In this section, we report experimental results using the Cloud Analytics Benchmark (CAB) on several Snowflake configurations and a serverless data warehouse. Our goal is to demonstrate how CAB can be used to compare and evaluate cloud data warehouse service architectures and pinpoint opportunities for improvement. The two systems are used as an example platform to highlight research questions and show the capabilities of CAB.

Setup. Our benchmark driver program is written in Javascript (Node.js version 17.7.2). The driver is responsible for running one query stream, hence a CAB run requires as many drivers as there are query streams. Most of the driver program is system independent: only a small adapter class is used to manage the connection details for a specific data warehouse system. The driver program has small resource requirements, as it only sends query string to the data warehouse platform and logs query runtimes. We ran all drivers for a particular benchmark run on a single EC2 instance (`c5.large`) and measured no significant delays due to the driver. Each of our drivers allows for at most 10 outstanding queries in order to avoid overloading the system under test.

Snowflake. Our Snowflake experiments were conducted in early 2022 using the Snowflake online service ("STANDARD" edition), running in AWS. Queries are sent to Snowflake via the official snowflake-SDK [28] (version 1.6.7). The limit for concurrently-running queries for each Snowflake warehouse is set to 32, the maximum allowed value. However, we observe that each warehouse only runs at most ≈6 queries concurrently. This is due to the internal Snowflake scheduler, which estimates how many queries a particular cluster can sustain at a time without over subscription [29]. Note that in this section, we use the more generic term *cluster* to refer to Snowflake's VWs.

System X. System X is another cloud data warehouse service. Unlike Snowflake, it operates as a serverless system: the user is not renting compute resources but pays on a per query bases. Further, the cluster automatically scales compute resources for the submitted queries. Similar to Snowflake, the data is stored on a cloud storage services such as S3 and has to be fetched for query processing.

Outline & Budget. We perform three kinds of experiments: (1) We run CAB on a single Snowflake virtual warehouse of varying size (i.e., number of compute nodes), demonstrating that static deployments are challenged by varying loads. (2) We compare these results with a CAB run where each database gets its own warehouse and observe how large queries can benefit from shared resources. (3) We compare the provisioned Snowflake offering to the serverless System X showing large potential in terms of cost. In total, we performed six CAB runs with cluster sizes ranging from 2 to 38

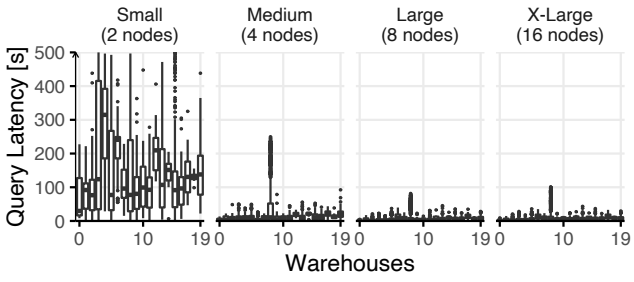

Figure 7: Shared Cluster Experiment: *Running CAB with* 1 TB *and* 10 *CPU hours (CAB-1) on a single Snowflake warehouse shared by all databases.*

machines and used around 140 CPU hours. These runs add up to around 230 $ for Snowflake and 155 $ for System X, showing that the cloud allows one to perform fairly large experiments within a reasonable budget.

6.1 Experiment: Shared Cluster Size

Setup. We configure CAB to generate a workload with 1 TB of total raw data size and 10 CPU hours split among 20 databases. This corresponds to a CAB-factor of 1 (i.e., CAB-1). The exact configuration of each database can be found in Table 4. In this experiment, all query streams are processed by one large shared Snowflake cluster (VW). We perform four runs with a varying cluster size ranging from 2 to 16 nodes. Note that this is not necessarily the intended way to use Snowflake, because in Snowflake different tenants are supposed to use different virtual warehouses. Nevertheless, this setup lets us evaluate how well a statically-sized cluster can cope with an elastic workload of multiple tenants.

Loading. The total data size, once loaded into Snowflake, is about 300 GB after compression. We used a small EC2 instance (2 vCPUs) to generate (`tpch-gen`), compress (`zstd`), and upload the data into AWS S3 (6 hours). After that, we used a small (1 node) Snowflake warehouse to copy the data from S3 into Snowflake tables (5 hours). Note that the S3 storage should be located in the same region as the Snowflake instance, to avoid network transfer costs. As loading data is not part of the benchmark, the load numbers should be taken as a rough estimate and could potentially be optimized. In fact, due the scalability of the cloud and the parallel nature of the problem, it should be possible to perform loads much quicker.

Query Arrival Rate. Each of the query streams (cf. Table 4) has its own query arrival rate pattern. However, when all query streams are put together, the large variations of individual streams average out into a noisy, but rather stable, overall load on the system under test. These combined arrival rates over the one-hour run are shown in the left most plot of Figure 6. The overall load varies between 40 and 160 queries per minute (4×), while individual query streams (especially pattern 2 or 3) have long stretches of time with no queries. We observe a similar effect in the Snowset (cf. Figure 4), where the overall load is rather stable over time while individual customers may vary greatly.

Queuing Time. Snowflake keeps track on the average number of queries queued and running in their warehouses. We show this information in Figure 6 (center) for the four runs with different warehouse sizes. First, we observe that the query arrival pattern is reflected in the queuing times: the two spikes are visible. Second, the larger the cluster, the less queuing is required. For instance, a 2 node cluster is not enough to process 10 CPU hours of work within an hour. In fact, it takes around 20 min longer for all queries to finish. Moving from 8 to 16 nodes, however, does not have a significant impact anymore.

Query Latency. This diminished return when moving to 16 nodes is also visible when looking at query latency (right-hand side of Figure 6): using a 16 node cluster does not provide any benefits for the CAB-1 workload. In addition, the plot distinguishes between the execution time (as reported by Snowflake) and overall runtime including queuing (as measured by our driver). The 2 node cluster appears to be at the edge of being oversubscribed, and we observe long queuing delays. Going up to 4 nodes, both the execution and the overall query latencies drop dramatically – with the median runtimes dropping from 683 s to 8.3 s. Query latencies decrease further (2×) when moving to an 8 node cluster. However, upgrading to 16 nodes provides little benefit and query latency even increases slightly. This is likely an artifact of Snowflake's scheduler assigning small queries too many resources.

Tenant Breakdown. Figure 7 shows the per tenant query latency (including queuing) for the four runs (2, 4, 8, and 16 node clusters). The small cluster is basically oversubscribed and almost all clients experience huge latencies (the plot is cut off at 500 s). The larger clusters distribute the queuing time evenly for the most part. However, even in the 16 node cluster, query stream number 8 still experiences higher latencies (roughly three times higher average query latency than the other query streams). This particular stream has the most queries and CPU time (cf. Table 4), which likely causes more queuing time. As mentioned before, this multi-tenant scenario may not be the intended way of using Snowflake. However, it highlights the scheduling challenges of a shared cluster architecture in a multi-tenant scenario.

6.2 Experiment: Shared vs Per-tenant Clusters

Setup. In this set of experiments, we increase the workload size to CAB-4, which corresponds to 4 TB of total raw data size and 40 CPU hours split among 20 tenants. We compare two setups: (1) One pool of fixed resources is used for all query streams (shared cluster). This corresponds to the setup from the previous section, where we determined that CAB-1 can be processed by a Snowflake cluster (VW) with 8 nodes. Therefore, we use a 32 node cluster for this CAB-4 experiment. (2) Each query stream is assigned its own pool of resources (one cluster per tenant). We determine the size of the warehouses per query stream proportionally to its required CPU hours. In total, we use 38 nodes for the cluster per tenant setup. However, due to the elasticity of many query streams, these warehouses can be suspended in periods of inactivity. To facilitate this elasticity, we configure the Snowflake warehouses to automatically shut down after 1 min of inactivity.

Results. The query latencies of this run are shown in Figure 8. The chart on the left-hand side shows the latency distribution for the shared cluster (SC) and per tenant cluster (TC) modes. The other two charts, show a per-tenant breakdown in logarithmic and linear

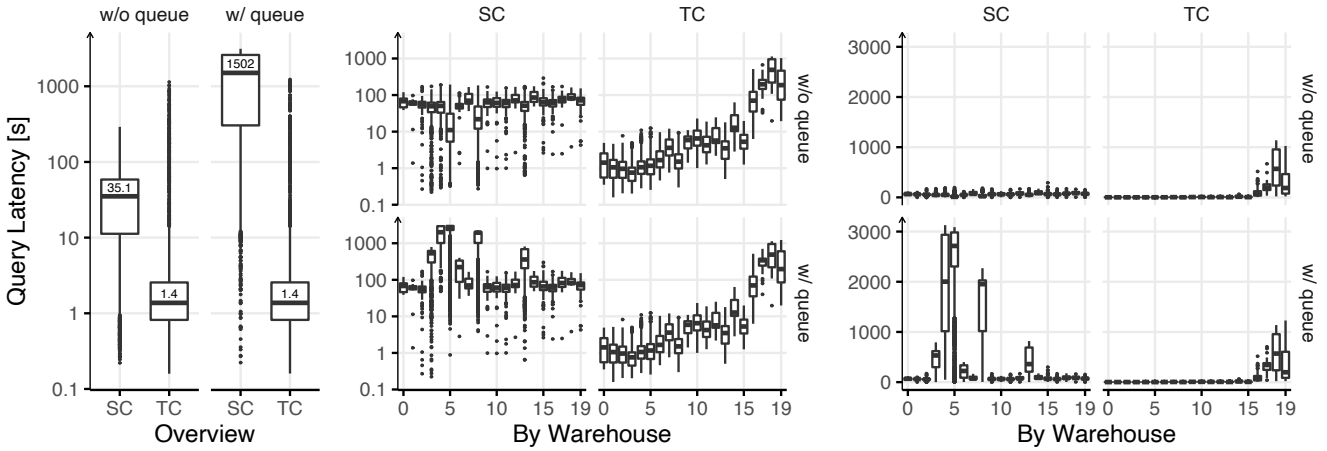

Figure 8: Cloud Architecture Comparison: *CAB-4 (*4 TB *and* 40 *CPU hours). Comparing the performance of one large shared (*32 *node) cluster for all databases (labeled as shared cluster: "SC") with individually assigned clusters of varying sizes depending on the CPU load (*39 *node) for each database (labeled as per tenant clusters: "TC"). In SC the cluster is running the whole time requiring* 32 *credits (≈ 83$/h), while the individual clusters can occasionally shut down and only require* 29 *credits (≈ 72$/h).*

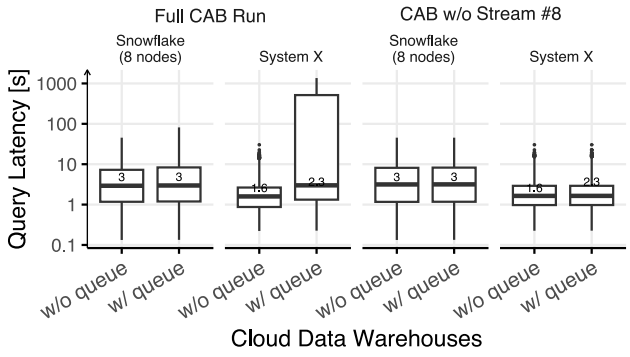

Figure 9: Snowflake vs System X: *Running CAB with* 1 TB *and* 10 *CPU hours (CAB-1) on a single Snowflake warehouse and another shared-everything cloud data warehouse service ("System X").*

scale. While the amount of compute resources and costs are similar (32 and 38 nodes), we can observe that the latency is much worse for the shared cluster, especially with queuing time. Furthermore, the per-tenant breakdown shows that query steams with many queries suffer in latency, as the cluster is occupied, and they do not have dedicated resources. However, the large shared cluster is faster in processing queries on the larger databases (i.e., the ones with higher query stream ids). Overall, it is striking how bad the SC approach performs, even though in comparison with CAB-1, we simply quadrupled the workload and hardware resources. This highlights the challenges of cluster sizing, and may indicate room of improvement regarding scalability and scheduling.

6.3 Experiment: System Comparison

Setup. Let us now compare Snowflake with a serverless data warehouse systems. Similar to the first experiment (Section 6.1), we use CAB-1 (1 TB of data and 10 CPU hours). We are using an 8 node Snowflake cluster, as this size was easily able to handle the workload as shown in Section 6.1. System X, as a serverless data warehouse with per-query pricing, is a fully managed service with few configuration options. The goal is to compare a provisioned system with one that was build as a serverless one from the ground up.

Results. The results of the full experiment are shown in the left-hand side of Figure 9, labeled "full". The significantly better query performance (without considering the queuing time) in System X might indicate a better internal resource usage, but could also be due to better hardware or number of utilized nodes. The more relevant metric for the user is the query time with queuing. Here, System X performs similar in the median, but suffers from huge latency spikes (up to 30 min). This is due to query stream 8, which can not be handled fast enough leading to increasing queue times and causing the stream to only finish after 1 h and 20 min. Consulting the query stream configuration in Table 4, we observe that query stream 8 has no unusual CPU load, but a lot more individual queries. This might suggest a contention in the front-end of System X.

As a comparison we show the numbers without query stream 8 on the right side of the chart (Figure 9). System X is roughly two times faster in the median for the remaining 19 query streams. However, this comes at a cost: the full run in System X cost around 155 $ while Snowflake only causes 22.8 $ in provisioning cost.

7 RELATED WORK

There is a large body of work [22] on benchmarking various aspects of the cloud such as Web 2.0 [30], data mining [1], No-SQL databases [5, 16, 18], or cloud infrastructure itself [15, 21]. To limit the scope of this section, we focus on those bodies of work that are related to database management systems. Our Cloud Analytics Benchmark (CAB) distinguishes itself by targeting cloud analytics services, providing an easy to compare metric, and, most importantly, being built by using an actual query trace from an existing

cloud data warehouse (i.e., the Snowset). This makes it representative of actual workloads while exercising important new cloud metrics (elasticity and multi tenancy) and providing a clear user-centric performance metric (i.e., latency and cost). In the following, we first summarize several papers that call for new cloud benchmarks and outline how CAB fulfills their requirement or why we diverged from them. After that, we describe related cloud benchmarks, outline how they differ from CAB, and motivate why we felt that a new benchmark is necessary.

7.1 Calls for Cloud Benchmarks

There have been several papers pointing to the need for new database benchmarks in the cloud area. Already in 2009, Binnig et al. [2] argued that the TPC benchmarks are not sufficient and there needs to be more focus on scalability, pay-per-use, and fault-tolerance. CAB abstracts from the payment model of the data warehouse service and is able to compare different models from a customers point of view. Scalability is an inherent aspect of CAB: with growing dataset, only a scalable query engine will avoid growing query runtimes. And, lastly, while CAB does not explicitly test or evaluate fault-tolerance, we argue that a 24 h run involving multiple databases and therefore a large number of physical machines requires the system to provide a decent amount of fault-tolerance. Any failures during the run will add to the latency and show up in the tail latency of the experiments. Binnig et al. further argue for a full-stack benchmark because different services across various clouds offer unique guarantees but need to be comparable. They believe that a user-centered benchmark will force systems to optimize for the right metric: the user. We agree with this assessment and designed CAB without any limiting requirements and focus on core database features that are relevant for the users (i.e., query processing). Since the publication of this work over a decade again, the TPC has published a number of benchmarks targeting some of these concerns, including the TPCx-BB which we discuss below.

Badir et al. [24] as well as Tosun et al. [13] identify many relevant performance metrics for cloud systems (performance, cost, scalability, elasticity, availability, cost-performance, cost-effectiveness, and SLAs). Badir et al. propose the idea of an end-to-end benchmark that simulates a Web 2.0 application. While we agree that all these metrics are relevant, we believe that many of them are subsumed in CAB's primary metrics (latency and performance). In addition, a Web 2.0 benchmark is only able to compare data warehouses as part of a Web 2.0 deployment. However, data warehouses are often used for other tasks and constitute a large enough piece of software to warrant individual benchmark.

7.2 Comparison with other Benchmarks

Similar to our work, there have been a number of papers suggesting extensions to existing TPC benchmarks. For instance, Szyperski et al. [27], recognize the need for elasticity in the cloud and propose to add this to TPC benchmarks. They extend a Hadoop-based benchmark (TPCx-BB) and model the query arrival rate based on a real-world data from Cosmos DB. In contrast, our benchmark is focused on analytical SQL database engines and instead of only a single arrival pattern, our analysis found that five unique patterns. Especially pattern 5 (cf. Figure 5), is known to be problematic due to customers starting jobs at each full hour (e.g., cron jobs). In addition, we take data refresh into account and model the multi tenancy of cloud data warehouse services.

Other work [11, 17, 32] has focused on the scalability of cloud services. They find that, in general, scale-up is preferable, in terms of cost performance ratio, to scale-out. However, scale-out can support greater elasticity. While the ability to scale (out and up) is crucial for larger data warehouses and to support elasticity, it on its own is only a building block for smart data warehouse services that can adapt to the customers needs. Hence, these evaluations are good at testing the scalability of various systems, but CAB evaluates the elasticity of a cloud data warehouse in an end-to-end scenario.

8 CONCLUSION AND RESEARCH QUESTIONS

Conclusion. This paper provides a detailed analysis of a real-world cloud analytics workload. Using these insights, we propose the Cloud Analytics Benchmark (CAB), which captures the elasticity and multi-tenancy of cloud workloads. CAB makes cloud data warehouses comparable and gives database architects optimization targets, thus hopefully guiding the design of future services.

Possible Extensions. As one of the next steps, CAB can be extended to more cloud data warehouses services. Using CAB, one can more easily compare the performance/cost trade-off for various providers and make a better decision. In addition, these results can help to determine architectural trade-offs for cloud vendors and highlight shortcomings in current deployments. Especially, the simple way (only two metrics) of comparing the elasticity capabilities of systems should be useful. Furthermore, CAB itself can be extended. TPC-H as the query-provider of CAB could be exchanged or extended to model the operator distribution of real cloud data warehouses (cf. Figure 3) more accurately. An interesting aspect of this work would to avoid overfitting to Snowflake's particularities. This might require profiling workloads in different query engines and, therefore, access to more workloads or at least logs such as the Snowset. Further, CAB could be extended with longer runs (weekly load fluctuations), specialized tenants (ETL or ML jobs), or different query patterns (more patterns and correlated load spikes).

Research Questions. Our initial experiments have already surfaced important research questions, such as: (1) In the evaluation, we used a trial and error approach to determine that 8 nodes are sufficient to run CAB-1. How can we automatically determine the cluster size for a particular workload while minimizing the latency of queries and cost for the customer at the same time? (2) Our evaluation has also shown that a shared resource pool can be beneficial for large queries, but has lower performance for smaller customers with many queries. What would be an optimal service model? (3) What is the best computational substrate (e.g., instances, cloud functions) for multi-tenant, intermittent cloud analytics? We hope that CAB will help to answer these questions.

ACKNOWLEDGMENTS

Funded/Co-funded by the European Union (ERC, CODAC, 101041375). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

REFERENCES

[1] Collin Bennett, Robert L. Grossman, David Locke, Jonathan Seidman, and Steve Vejcik. 2010. Malstone: towards a benchmark for analytics on large data clouds. In *KDD*.

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

[3] Peter A. Boncz, Thomas Neumann, and Orri Erling. 2013. TPC-H Analyzed: Hidden Messages and Lessons Learned from an Influential Benchmark. In *TPCTC*. 61–76.

[4] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. 2009. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Gener. Comput. Syst.* (2009).

[5] Yanpei Chen, Archana Ganapathi, Rean Griffith, and Randy Katz. 2011. The case for evaluating mapreduce performance using workload suites. In *IEEE international symposium on modelling, analysis, and simulation of computer and telecommunication systems*.

[6] Benoit Dageville and Thierry Cruanes. Accessed: 2022-04-14. Industry Benchmarks and Competing with Integrity. https://www.snowflake.com/blog/industry-benchmarks-and-competing-with-integrity.

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

[8] Snowflake Documentation. Accessed: 2021-10-11. Snowflake Isolation Level. https://web.archive.org/web/20211011014842/https://docs.snowflake.com/en/sql-reference/transactions.html.

[9] Snowflake Documentation. Accessed: 2021-12-09. Snowflake Data Encryption. https://web.archive.org/web/20211209111258/https://docs.snowflake.com/en/user-guide/security-encryption.html.

[10] Markus Dreseler, Martin Boissier, Tilmann Rabl, and Matthias Uflacker. 2020. Quantifying TPC-H Choke Points and Their Optimizations. *VLDB* (2020).

[11] 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 2012, London, UK, March 3-7, 2012*.

[12] Daniela Florescu and Donald Kossmann. 2009. Rethinking cost and performance of database systems. *SIGMOD Rec.* (2009).

[13] Enno Folkerts, Alexander Alexandrov, Kai Sachs, Alexandru Iosup, Volker Markl, and Cafer Tosun. 2012. Benchmarking in the Cloud: What It Should, Can, and Cannot Be. In *TPCTC*.

[14] George Fraser. Accessed: 2022-03-12. 2020 Cloud Data Warehouse Benchmark: Redshift, Snowflake, Presto and BigQuery. https://www.fivetran.com/blog/warehouse-benchmark.

[15] Simson Garfinkel. 2007. An evaluation of Amazon's grid computing services: EC2, S3, and SQS. (2007).

[16] Shengsheng Huang, Jie Huang, Jinquan Dai, Tao Xie, and Bo Huang. 2010. The HiBench benchmark suite: Characterization of the MapReduce-based data analysis. In *(ICDEW)*.

[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] Kiyoung Kim, Kyungho Jeon, Hyuck Han, Shin-gyu Kim, Hyungsoo Jung, and Heon Y Yeom. 2008. Mrbench: A benchmark for mapreduce framework. In *IEEE International Conference on Parallel and Distributed Systems*.

[19] Andreas Kipf, Damian Chromejko, Alexander Hall, Peter A. Boncz, and David G. Andersen. 2020. Cuckoo Index: A Lightweight Secondary Index Structure. *VLDB* (2020).

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

[21] Ang Li, Xiaowei Yang, Srikanth Kandula, and Ming Zhang. 2010. CloudCmp: comparing public cloud providers. In *SIGCOMM*.

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

[23] Guido Moerkotte. 1998. Small Materialized Aggregates: A Light Weight Index Structure for Data Warehousing. In *VLDB*.

[24] Rim Moussa and Hassan Badir. 2013. Data Warehouse Systems in the Cloud: Rise to the Benchmarking Challenge. *Int. J. Comput. Their Appl.* (2013).

[25] Ingo Müller, Renato Marroquín, and Gustavo Alonso. 2020. Lambada: Interactive Data Analytics on Cold Data Using Serverless Cloud Infrastructure. In *SIGMOD*.

[26] Matthew Perron, Raul Castro Fernandez, David J. DeWitt, and Samuel Madden. 2020. Starling: A Scalable Query Engine on Cloud Functions. In *SIGMOD*.

[27] 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*.

[28] Snowflake. Accessed: 2022-01-22. Snowflake SDK on NPM. https://www.npmjs.com/package/snowflake-sdk.

[29] Snowflake. Accessed: 2022-04-15. Warehouse Concurrency and Statement Timeout Parameters. https://community.snowflake.com/s/article/Warehouse-Concurrency-and-Statement-Timeout-Parameters.

[30] Will Sobel, Shanti Subramanyam, Akara Sucharitakul, Jimmy Nguyen, Hubert Wong, Arthur Klepchukov, Sheetal Patil, Armando Fox, and David Patterson. 2008. Cloudstone: Multi-platform, multi-language benchmark and measurement tools for web 2.0. In *Proc. of CCA*.

[31] 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).

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

[33] Alexander van Renen. Accessed: 2022-04-15. Cloud Analytics Benchmark (CAB). https://github.com/alexandervanrenen/cab.

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

[35] 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*.

[36] Reynold Xin and Mostafa Mokhtar. Accessed: 2022-04-14. Databricks Sets Official Data Warehousing Performance Record. https://databricks.com/blog/2021/11/02/databricks-sets-official-data-warehousing-performance-record.html.