Choosing A Cloud DBMS: Architectures and Tradeoffs

Junjay Tan[1], Thanaa Ghanem[2,*], Matthew Perron[3], Xiangyao Yu[3], Michael Stonebraker[3,6], David DeWitt[3], Marco Serafini[4], Ashraf Aboulnaga[5], Tim Kraska[3]
[1]Brown University; [2]Metropolitan State University (Minnesota), CSC; [3]MIT CSAIL; [4]University of Massachusetts Amherst, CICS; [5]Qatar Computing Research Institute, HBKU; [6]Tamr, Inc.
junjay@brown.edu, thanaa.ghanem@metrostate.edu, {mperron,yxy,stonebraker}@csail.mit.edu, david.dewitt@outlook.com, marco@cs.umass.edu, aaboulnaga@hbku.edu.qa, kraska@mit.edu

ABSTRACT

As analytic (OLAP) applications move to the cloud, DBMSs have shifted from employing a pure shared-nothing design with locally attached storage to a hybrid design that combines the use of shared-storage (e.g., AWS S3) with the use of shared-nothing query execution mechanisms. This paper sheds light on the resulting tradeoffs, which have not been properly identified in previous work. To this end, it evaluates the TPC-H benchmark across a variety of DBMS offerings running in a cloud environment (AWS) on fast 10Gb+ networks, specifically database-as-a-service offerings (Redshift, Athena), query engines (Presto, Hive), and a traditional cloud agnostic OLAP database (Vertica). While these comparisons cannot be apples-to-apples in all cases due to cloud configuration restrictions, we nonetheless identify patterns and design choices that are advantageous. These include prioritizing low-cost object stores like S3 for data storage, using system agnostic yet still performant columnar formats like ORC that allow easy switching to other systems for different workloads, and making features that benefit subsequent runs like query precompilation and caching remote data to faster storage optional rather than required because they disadvantage ad hoc queries.

PVLDB Reference Format:
Junjay Tan, Thanaa Ghanem, Matthew Perron, Xiangyao Yu, Michael Stonebraker, David DeWitt, Marco Serafini, Ashraf Aboulnaga, Tim Kraska. Choosing A Cloud DBMS: Architectures and Tradeoffs. *PVLDB*, 12(12): 2170-2182, 2019.
DOI: https://doi.org/10.14778/3352063.3352133

1. INTRODUCTION

Organizations are moving their applications to the cloud. Despite objections to such a move (security, data location constraints, etc.), sooner or later cost and flexibility considerations will prevail, as evidenced by even national security agencies committing to vendor hosted cloud deployments [17]. The reasons deal with economies of scale (cloud vendors are deploying servers by the millions; not by the thousands) and specialization (cloud vendors' business priority is infrastructure management, whereas other organizations perform this to support main lines of business).

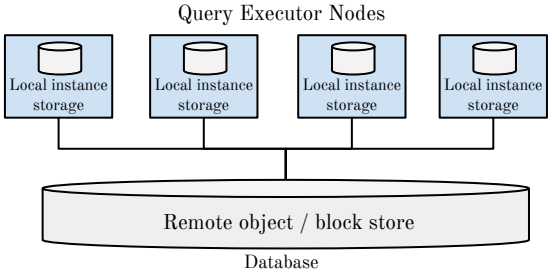


Figure 1: Shared Disk Architecture

For analytic applications running on the cloud, data resides on external shared storage systems such as S3 or EBS offerings on Amazon Web Services (AWS). Query executors are spun on-demand in compute nodes such as EC2 nodes in AWS. Compute nodes should be kept running only when strictly necessary because the running cost of moderately sized instances is orders of magnitude greater than the cost of storage services. This has resulted in a fundamental architectural shift for database management systems (DBMSs). In traditional DBMSs, queries are executed in the same nodes that store the database. If the DBMS is distributed, the dominating paradigm has been the shared-nothing architecture, whereby the database is partitioned among query execution servers. Cloud architectures fit more naturally in the alternative "shared-disk" architecture, where the database is stored by separate storage servers that are distinct from query execution servers (see Figure 1).

For the cloud provider selling a DBMS-as-a-service, an architecture that "decouple[s] the storage tier from the compute tier" provides many advantages, including simplifying node failure tolerance, hot disk management, and software upgrades. Additionally, it allows the reduction of network traffic by moving certain DBMS functions like the log applicator to the storage tier, as in the case of Aurora [27].

Shared-disk DBMSs for cloud analytics face non-obvious design choices relevant to users. This paper sheds light on resulting tradeoffs that have not been clearly identified in previous work. To this end, we evaluate six popular production OLAP DBMSs (Athena, Hive [23], Presto, Redshift [10], Redshift Spectrum, and Vertica [14]) with different AWS resource and storage configurations using the TPC-H bench-

*Work done while at the Qatar Computing Research Institute

Proceedings of the VLDB Endowment, Vol. 12, No. 12
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3352063.3352133

mark. Despite being limited to these specific DBMSs and a single cloud provider, our work highlights general tradeoffs that arise in other settings. This study aims to provide users insights into how different DBMSs and cloud configurations perform for a business analytics workload, which can help them choose the right configurations. It also provides developers an overview of the design space for future cloud DBMS implementations.

We group the DBMS design choices and tradeoffs into three broad categories, which result from the need for dealing with (A) external storage; (B) query executors that are spun on demand; and (C) DBMS-as-a-service offerings.

Dealing with external storage: Cloud providers offer multiple storage services with different semantics, performance, and cost. The first DBMS design choice involves selecting one of these services. Object stores, like AWS S3, allow storing arbitrary binary blobs that can be associated with application-specific metadata and are assigned unique global identifiers. These data are accessible via a web-based REST API. Alternatively, one can use remote block-level storage services like AWS EBS, which can be attached to compute nodes and accessed by their local file system. Google Cloud Platform (GCP) and Microsoft Azure offer similar storage choices. Which abstraction performs best and is most cost effective?

Compute node instances (EC2) are increasingly being offered with larger local instance storage volumes. These nodes have faster I/O than those with EBS storage and may be cheaper per unit storage but are ephemeral, limited to certain instance types and sizes, and do not persist data after system restarts. How to use them in the DBMS design? By initially pre-loading the database onto local instance storage, a DBMS can keep the traditional shared-nothing model. Alternatively, the local storage can be used as a cache to avoid accessing remote storage. Is local caching advantageous in a time of ever-increasing network speeds?

The DBMS design also has to deal with different data formats. Keeping data on external shared storage means that data can be shared across multiple applications. In fact, many DBMSs are able to access data stored in DBMS-agnostic formats, such as Parquet or ORC. Which compatibility issues arise in this context?

Dealing with on-demand query executors: Query executors should be kept running as little as possible to minimize costs, so they may be often started and stopped. A consequence is that query executors have different startup times and often run queries with a cold cache. Which DBMSs start quickly? Which DBMSs are designed for optimal performance with a cold vs warm cache? This relates to how well systems deal with one-off, ad-hoc analytical queries. It is reasonable to expect systems to perform better with a warm cache, but how large is the difference?

Since query executors are paid per use, scalability becomes an even more critical feature of a DBMS than usual. Consider an ideal scale-out DBMS that can execute a workload in time T using N instances or in time T/2 using 2N instances. Assume that the startup and shutdown times are negligible compared to T. Since the pricing is per-instance per-time, the cost of executing the workload with N or 2N instances is the same, so we can complete the task much faster at no additional cost. A similar argument can be made for a scale-up DBMS running on instances that are twice as powerful. How do existing DBMSs scale vertically and horizontally in cloud settings?

Dealing with DBMS-as-a-service offerings: Many cloud providers have DBMS-as-a-service offerings, such as Athena or Redshift on AWS. These come with different pricing structures compared to other services such as EC2 or S3. For example, Athena bills queries based only on the amount of data scanned. These services also offer less flexibility to users in terms of the resources they use and hide key low-level details entirely. How do these different classes of systems compare?

Summary of findings: This paper provides a detailed account of these tradeoffs and sheds light into these questions. The main findings include the following:

- Cheap remote shared object storage like S3 provides order of magnitude cost savings over remote block stores like EBS that are commonly used for DBMS storage in shared-nothing-architectures, without significant performance disadvantages in mixed workloads. (Shared nothing architectures adapted for the cloud often do not use true local storage because local data is not persisted upon node shutdown.)
- Physically attached local instance storage provides faster performance than EBS. Additionally, its cost is coupled into compute costs and this provides slight cost advantages over EBS due to AWS's contractual compute resource pricing schemes.
- Caching from cheap remote object storage like S3 to node block storage is disadvantageous in cold start cases and should not always be done by default.
- A carefully chosen general use columnar format like ORC provides flexibility for future system optimization over proprietary storage formats used by shared-nothing DBMSs and appears performant on TPC-H even without optimized partitioning. Shared-nothing systems try to bridge the gap with hybrid features (Vertica Eon and Redshift Spectrum), but their cost-performance characteristics are very different than systems focused on utilizing these general data formats like Presto.
- Redshift is unique among the systems tested in that it compiles queries to machine code. Because it is very efficient in the single-user use case on warm and cold cache, query compilation time is not disadvantageous on TPC-H. However, compilation can be disadvantageous on short-running queries or if workloads are changing, making it impossible to leverage previously compiled queries.
- Most systems gain from cluster horizontal scaling, but our limited data suggests that vertical scaling is less beneficial.

The rest of this paper is structured as follows. Section 2 highlights related work, while Section 3 explains the experimental setup. Section 4 discusses results and key findings. Finally, Section 5 provides conclusions and opportunities for future work.

2. RELATED WORK

Previous benchmarking studies have evaluated several types of systems but largely ignored OLAP DBMSs. Unlike our study, these benchmarking studies only compare against self-provisioned systems, either in an on-premise cluster or on cloud nodes. Our study encompasses a broader selection of cloud offerings and incorporates DBMS-as-a-service offerings.

[13] and [6] benchmarked OLTP systems in the cloud but did not consider recent services such as Amazon Redshift and are almost a decade old. In contrast, [4] only focused on cloud storage systems, [9] and [11] evaluated graph database systems, and [5] described cloud benchmark design goals. There have also been many studies on cloud compute servers (i.e., VMs) [26, 25, 15, 18] and comparisons of reserve vs spot instances [1]. However, these benchmarking efforts were at a much lower level and do not address simple architectural questions relevant to a cloud data warehouse user.

Outside the academic literature, vendors have presented bake-offs between different systems, but these are narrowly targeted at showing that system X is better than Y, often in vendor-proprietary setups or for specific data formats (e.g., [8, 22]). Similarly, companies using various systems have published performance results showing why they chose system X for their needs, but these are typically based on a few in-house workloads with little supporting detail, such as [20].

3. EXPERIMENTAL SETUP

This section describes the systems, configurations, assumptions, and limitations in our testing. We also describe system-specific tuning and cost calculations. Additional details can be found at `https://github.com/junjaytan/choosing-a-cloud-dbms-vldb2019`. Some high-level points are listed below:

- We focused on single-user workloads to reproduce the common use case of ad-hoc cloud analytics.
- Queries were initiated by a separate client node. This was done to allow parity with Redshift and Athena, which do not allow client code to run on the DB nodes.
- Result sets were sent to the client node from the DBMS and then to /dev/null on the client. We verified that results matched TPC-H specifications. The largest result set for any query was approximately 25MB.
- Elapsed time was measured via the Unix time utility.

3.1 Storage

On AWS, the main file systems are two block store options–Elastic Block Store (EBS) and Instance Store (InS)–and the Simple Storage Service (S3) object store. Block stores use a standard file system API and are offered as Solid State Disk (SSD) or Hard Disk Drive (HDD). EBS is remote network storage that is co-located in specified regions, while InS is physically attached to the compute node. In contrast to EBS, InS is not persistent and disappears once the compute node is shut down, with a fresh volume provided on node restart. For this reason, EBS is traditionally the more suitable option for DBMSs and is the option we use in this paper. As cloud trends are already heavily skewed towards SSD over HDD, we test only on SSD. Additionally, InS is now offered in Non-Volatile Memory Express (NVMe) variants, which are much faster than general purpose SSDs (listed at 1.9GB/s vs 250 MB/s per volume). We performed limited testing on NVMe for comparison.

In contrast to block stores, S3 is an object store that runs on dedicated S3 nodes, and storage is accessed using a web-based REST API. S3 is designed for scalable, high concurrency workloads but has higher latency and more throughput variability than block stores. Hence, block stores are traditionally preferred for performance reasons. It is feasible to partition S3 data onto multiple storage volumes, using naming of storage locations (object key prefixes) to support parallelism.

Most DBMSs can read data from multiple storage systems. Presto and Hive use the Hive connector to read data from S3 or HDFS, and Vertica can read data from S3 or EBS. However, AWS-proprietary systems tend to be more restricted. For systems that support multiple storage backends, we generally tested multiple storage types for the base 4-node r4.8xlarge cluster configuration but chose only one storage type for scalability tests due to cost.

In systems that used EBS to store workload data (Vertica and Hive on HDFS), we configured 8x200GB EBS general purpose SSD volumes per node in RAID 0. This configuration was selected for Vertica based on the user guide, and we used the same setup on Hive HDFS for parity. Additionally, we used a 512GB EBS volume per node as scratch disk on all systems. Since EBS is charged by amount of data stored (with additional costs if a user desires guaranteed higher IO rates), it is advantageous to split the total volume size into multiple smaller volumes that can be read in parallel. However, there is a lower bound on volume sizes because AWS scales volume IO throughput down by size. We varied volume sizes and did not find that TPC-H performance benefited from larger volumes.

3.2 Systems Tested

We focused on OLAP-oriented parallel data warehouse products available for AWS and restricted our attention to commercially available systems. As it is infeasible to test every OLAP system runnable on AWS, we chose widely-used systems that represented a variety of architectures and cost models.

The systems evaluated can be categorized as AWS proprietary database-as-a-service offerings, query engines (serverless services and self-provisioned clusters), and a cloud agnostic OLAP database. Table 1 summarizes the storage systems available by each system for the database and for temporary data artifacts. The database storage system refers to where the DBMS persists and reads core data from during a query (note that this storage volume may disappear after system shutdown as in the case of local instance storage), whereas the temporary storage system is used by the DBMS to hold artifacts not held in memory, such as spill-to-disk joins and in some cases data from another remote store.

Redshift is AWS's parallel processing OLAP database service that employs a traditional shared-nothing architecture. It is offered in instance sizes that are named differently than the compute types offered on EC2, so there may be other hardware associated with it that are not available to general users. Additionally, Redshift's pricing model is different from that available to both end users and companies selling database products running on AWS. For example, Redshift node snapshots are free, but EC2 snapshots are not.

Table 1: Tested Systems and Supported Storage Architectures

Category	DBMS	Database Storage System	Temp Node Storage System and Usage
Proprietary database-as-a-service offerings	Redshift	Local storage (snapshotted to S3)	Local storage for spill- to- disk
	Spectrum (Redshift feature)	Remote object store (S3)	Local storage for spill to disk and possibly remote data
	Athena	Remote object store (S3)	Unknown, but no cache effects observed
Query engines	Presto	Remote object store (S3 or HDFS)	Node mounted storage volumes (EBS or local) for spill-to-disk
	Hive	Remote object store (S3 or HDFS)	Node mounted storage volumes (EBS or local) for spill-to-disk
Cloud provider agnostic OLAP DBMS	Vertica	Node mounted storage volumes (EBS or local)	Node mounted storage volumes (EBS or local) for spill-to-disk
	Eon (Vertica mode)	Remote object store (S3 or HDFS)	Node mounted storage volumes (EBS or local) for spill-to-disk and caching S3 data

Spectrum is a recent feature added to Redshift that allows querying S3-resident data in various formats. Spectrum incurs additional costs per data scanned from S3 in addition to standard Redshift compute node costs.

Vertica is a shared nothing, multi-node parallel column store DBMS. As of Vertica 9, it can use EBS, instance store, or S3 for storage. In block store configurations (i.e., EBS or InS) Vertica runs normally with partitioned data in a shared nothing configuration. With S3, known as **Eon Mode,** all data is stored as S3 objects and each processing node accesses a partition of the data [24]. Note that on AWS, Vertica recommends running on EBS storage, even though EBS is technically remote storage that appears as individual file volumes [28]. Therefore, it does not truly run as a traditional shared-nothing DBMS on AWS unless one uses InS, which is traditionally uncommon because Vertica assumes storage persistence.

Presto is a distributed query engine originally developed by Facebook and now open-sourced [19]. It is a multi-node parallel SQL engine with a variety of built-in and third-party connectors including ones for HDFS and S3. We enabled spilling intermediate tables to disk when main memory is exhausted. Without spill-to-disk, Presto could not successfully run all the TPC-H queries. Additionally, we used the Starburst Data fork of Presto (release 0.195) which includes a query optimizer, since Presto is sensitive to join orders otherwise.

Athena is an Amazon product derived from Presto and optimized for AWS and S3 with a unique cost model: it automatically adjusts query parallelism on undisclosed nodes and charges only by amount of S3 data scanned.

Apache Hive is a data warehouse system that was built originally on Hadoop but has been retargeted to run directly on top of HDFS via Tez and YARN [12]. Experiments were run on Hive 1.2.1000, which was the version included in Hortonworks Data Platform 2.6.

We considered Apache Drill and Apache Spark SQL, but early testing determined that their performance was strongly dominated by other similar systems so we did not perform further experiments. We wanted to test Snowflake [7] but the vendor was unwilling to provide us permission to publish the results.

3.3 System Settings

We assume a "cold start" configuration unless stated otherwise. In this case, we are not measuring the quality of the DBMS caching policy and also allow comparison to on-demand query services. To remove cache effects, we followed all vendor instructions for clearing the DBMS and OS caches after each run. Where this was insufficient, as in the case of Hive and Redshift, we also restarted the system or recreated the cluster, respectively, after each query execution. Athena has no notion of caching. In the warm cache case, we first ran the entire query suite after a cold start before taking measurements on subsequent runs of the same query suite.

To ensure that a similar amount of buffer memory was available for use by each system, we configured Vertica's memory resource pool, Hive's memory per the Live Long and Process (LLAP) daemon, and the JVM used by Presto to 70% of total system memory. It is not possible to control the amount of buffer memory with Athena, Redshift, or Spectrum. Although we present results for 70% memory, we did not observe any noticeable difference on Vertica with the memory resource pool left at the default.

Vertica was installed via the vendor-provided AWS image running on CentOS 7, while manual installation of Vertica on i3 instances used RHEL 7. Presto Starburst fork was installed on AWS EMR clusters, while Hive was installed via the Hortonworks Data Platform (2.6.4.0). AWS Elastic Network Adapter (ENA) was enabled on all configurations.

3.4 Data Formats and Partitioning

We used 1000 scale factor TPC-H data (1TB uncompressed). This was large enough to be I/O constrained yet queries could complete in seconds to minutes. The longest query on the base 4 node 8xlarge configuration, described in the next section, took 10 minutes to run, with the average query runtime being about 2 minutes. As such, we are not testing extremely long OLAP workloads that take hours to complete. On block storage systems for shared-nothing style DBMSs (Vertica EBS and Redshift), we configured similar data partitioning schemes, where data was distributed mainly on the table primary keys and/or join keys.

On S3, data can be partitioned by hashing object key names rather than using sequential key prefixes. For systems that used ORC data on S3, we attempted to similarly partition data across all systems but found that this required system-specific configurations. Specifically, Hive and Presto can automatically partition the data via a partition command in the create table syntax, which rearranges the data into additional S3 subdirectories; however, Spectrum requires a different subdirectory structure and explicit commands to add each individual partition, such as one for region A, one of region B, etc. We therefore used the same unparti-

tioned ORC data on S3 for Presto, Hive, and Spectrum with the acknowledgement that partitioning has the potential to improve performance.

On Vertica S3 (Eon Mode), we loaded data to S3 from the raw data files with the same schema definition used for Vertica EBS. This means data was stored on S3 in Vertica's compression format rather than ORC, since the Vertica version used was unable to directly read the ORC tables.

For Spectrum, all data was stored on S3, although the vendor suggested that hybrid configurations are common and could be tested instead. However, we believe that testing Spectrum entirely on S3 data is representative of S3's common use case for storing large datasets, and hybrid performance can be inferred to be somewhere between that of Redshift and Spectrum on S3. Future work could explore hybrid setups in more detail.

3.5 Node Types and Cluster Sizes

AWS offers dozens of Elastic Compute Cloud (EC2) instance variants, so it is infeasible to test them all. AWS currently classifies compute nodes into five families. Each family denotes a general category of systems: general purpose, compute optimized, memory optimized, accelerated computing (for GPU applications), and storage optimized. Within each family are multiple node types, which equate to how much CPU, memory, and network speed are provided.

A performance bottleneck for databases in the cloud, and in particular shared disk architectures, is network speed. We want to understand how performance across systems compares when using fast modern network speeds. Past work found that network speeds of 1Gb make shared storage uncompetitive with shared-nothing architectures [21]. However, 10Gb+ speeds are now widely available and industry trends point towards ever faster speeds. Therefore, we limit our work to instances with 10Gb+ network speeds, which restricts the EC2 families and instance sizes that could be used.

We focused our evaluations on the memory optimized r4 and storage optimized i3 instance types. AWS markets r4 as good for high performance databases and analytics, whereas i3 is marketed as being good for high transaction, low latency workloads and data warehousing. We used the 4xlarge through 16xlarge sizes, ranging from 16 to 64 vCPUs. Basic performance and cost characteristics of these EC2 instance types are listed in Table 2.

Table 2: Tested Compute Instance Types

Type	vCPUs	Mem (GB)	Storage	Network (Gb/s)	Hourly Cost (on demand)
r4.16xlarge	64	488	EBS	25	$4.256
r4.8xlarge	32	244	EBS	10	$2.128
r4.4xlarge	16	122	EBS	10	$1.064
i3.8xlarge	32	244	NVMe SSD	10	$2.496
Redshift dc2.8xlarge	32	244	NVMe SSD	-	$4.80

The default configuration used for comparing all systems was a **4 node, r4 8xlarge cluster** (using the similar, but not identical, dc2 instance type on Redshift). We did not use other 8xlarge types like C4 (the "compute optimized" family) since the vCPUs and RAM are not equivalent to Redshift's. We also could not perform a full evaluation of Redshift's smaller node size since there was insufficient storage. We performed horizontal and vertical scaling experiments on a subset of systems, as explained next.

Redshift, Redshift Spectrum, and Athena cannot be compared directly with the r4 configurations because they execute on proprietary hardware configurations that are not clearly equivalent to those available for general usage. Redshift only offers two node families, each with only two sizes, the 8xlarge and a much smaller size. We used the closest Redshift analogous node family type that offers SSD storage, called dc2. Athena does not provide any node options and handles this behind the scenes for the user. Hence, any performance and cost comparisons for these systems will not be apples-to-apples with the previously listed configurations. However, these are the conditions by which a cloud DBMS user would need to work within, so we believe it is a reasonable and instructive–if not entirely equivalent–comparison to do in the cloud.

To gain further insight into Redshift performance, we also ran limited experiments of Vertica on the i3.8xlarge EC2 instance type, which has the same RAM, vCPU, and network characteristics as the r4.8xlarge but contains 4 NVMe SSD storage volumes and costs more. The NVMe drives were configured in RAID 0 in an attempt to achieve similar disk throughput as Redshift. We mention findings in the results section but did not do extensive experiments because we noticed insignificant differences from the performance of EBS Vertica on r4, which is the traditionally recommended configuration.

Table 3: Tested Cluster Configurations (Non-AWS Systems)

EC2 Type	4 node	8 node	16 node
r4.16xlarge	Vertica(EBS) Presto(S3) Hive(S3)	Not tested	Not tested
r4.8xlarge	Vertica(EBS) Presto(S3) Hive(S3,HDFS)	Vertica(EBS) Presto(S3) Hive(S3)	Vertica(EBS) Presto(S3) Hive(S3)
r4.4xlarge	Vertica(EBS) Presto(S3) Hive(S3)	Vertica(EBS) Presto(S3) Hive(S3)	Vertica(EBS) Presto(S3) Hive(S3)

The matrix of tested configurations for non-proprietary AWS systems is listed in Table 3, with configurations along the diagonal being equivalent in compute cost if run for the same amount of time. For example, a r4.4xlarge node costs half that of a r4.8xlarge node, so running four r4.8xlarge nodes over 1 hour would cost the same as eight r4.4xlarge nodes in an hour.

3.6 Cost Calculations

Broadly, total system costs are comprised of node compute costs, storage costs, and data scan costs.

3.6.1 Compute Costs

Node compute costs are calculated using on-demand prices for the US-East-1 region as of January 2019. Compute costs are not provided by AWS at the granularity of individual queries, so we calculated costs from the query runtimes and other associated node runtime processes (data snapshot load for Redshift, etc.). Presto and Hive require a master coordinator node, but we ignored this in our calculations since it is very small compared to the worker node costs. For example, AWS recommends using a single m4.large node for

a 50 node cluster, which is less than 10% the cost of one r4.4xlarge node [3].

3.6.2 Storage Costs

S3 and EBS storage costs used the currently listed AWS costs: $0.023 per GB for S3, and $0.10 per GB for general purpose (gp2) EBS. Presto, Vertica, and Hive on S3 utilize a scratch EBS disk (512 GB per node) to support spill-to-disk. We assume that users destroy this volume when not in use and reinitialize it upon system restart to save on costs. Scratch disk initialization time took less than 30 seconds.

3.6.3 Data Scan Costs

Data scan costs can be classified into two categories and pertain only to S3 storage. The first category consists of explicit data scan costs for Spectrum and Athena due to these systems' unique pricing models. For these systems, AWS charges $5 per TB scanned, and we measured the data scanned values using AWS monitoring tools. The second category applies to all other systems that retrieve data from S3, for which AWS charges an amount (currently $0.0004 per 1,000 GET requests). AWS does not provide a second-by-second measurement of GET requests that could be linked to specific queries, so for this cost we used the average object size in the relevant S3 dataset as the average GET request size to estimate the costs.

3.6.4 Software License Costs

Besides infrastructure costs, Vertica requires a license if users wish to run on more nodes than the community edition allows (currently 3 nodes). They also recently began offering a "pay as you go" pricing model that bundles licensing into node prices. We present Vertica costs as two extremes: without a license and as pay as you go, with the realization that most customers would own a license and pay an amount somewhere in between. Other systems do not have separate license costs.

4. EXPERIMENTAL RESULTS

This section presents results focused on the following comparison metrics of interest to a DBMS user: query restrictions; system initialization time; query performance; cost; data compatibility with other systems; and scalability.

4.1 Query Restrictions

For each query measurement, we performed three runs to ensure consistency, in many cases on different days. One complication was that neither Spectrum nor Athena could run the full TPC-H query suite. Hence, in plots where we compare all systems, we include only the 16 TPC-H queries that all systems could successfully execute. In other plots, we compare only systems that could run all queries. We note the number of queries in each figure's caption.

The 6 queries excluded when plotting all systems together were Q2, Q8, Q9, and Q21, which could not be run by Athena, and Q15 and Q22, which could not be run by Spectrum. On Athena, Q2 timed out after an hour, and Q8, Q9, and Q21 failed after exhausting resources. On Spectrum, Q15 failed because views are not supported while Q22 failed because complex subqueries are not supported.

4.2 Initialization Time

Initialization time measures how easy it is to launch and use a particular DBMS system. It can also be considered the "time to first insight." This is relevant to consider when a cloud DBMS user switches systems or shuts down a system to save on compute costs and then restarts it at a later time. It is also important for cluster scaling, which requires reconfiguring or launching additional instances. Figure 2 shows the initialization times by system in seconds.

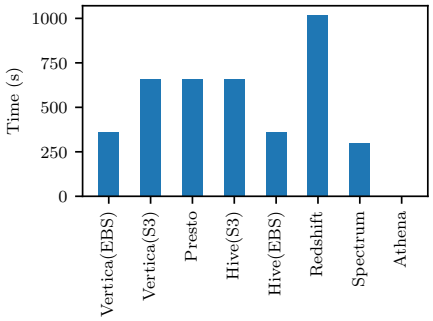


Figure 2: System Initialization Times

Athena, being an always-on service, does not require initialization before running a query. This is a key advantage of any serverless cloud service offering.

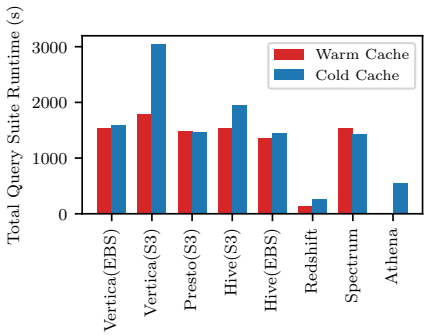
On the other extreme, Redshift takes longest to initialize because it must start the nodes and then load data from an S3 snapshot to the local NVMe drives. This latency cost is required for any system that employs ephemeral local storage, since cloud providers achieve efficiencies by retaining flexibility over how to place storage and compute units across their datacenters. For our 1 TB (uncompressed) dataset, this process takes approximately 12 minutes; larger datasets would take longer.

Other systems have initialization times in the range of 5-7 minutes. The vast majority of this time is that required for an EC2 instance to initialize and pass all system status checks, rather than the time required to initialize the DBMS. Vertica and Hive on EBS are the next fastest after Athena because their initialization time comprises only the time it takes to launch the EC2 nodes and start the systems. Vertica, Presto, and Hive on S3 are slightly slower to initialize because we must reinitialize the scratch disk instance. Alternatively, we could avoid this latency but pay the cost of keeping the EBS volume constantly attached. As we discuss in the storage costs section, this is significant and generally not worthwhile.

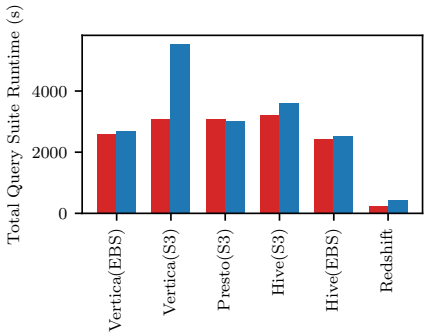
Summary. It is advantageous in the cloud to shut down compute resources when they are not being used, but there is then a query latency cost. All cloud nodes require time to initialize and pass system checks before a DBMS can be started, with systems using local storage like Redshift taking longest to initialize. Serverless offerings like Athena provide an alternative "instant on" query service.

4.3 Query Performance

This section presents query performance (runtime) using both a cold cache and a warm cache. Note that Redshift by default caches results so repeated runs of the identical query when data is unchanged take zero time; we turn this feature



(a) 16 queries runnable on all systems



(b) All queries runnable on subset of systems

Figure 3: Warm and Cold Cache Runtimes

off to get a representative warm cache runtime. However, workloads that have many repeat queries will benefit greatly from this feature.

4.3.1 System Comparisons

Figure 3 shows cold and warm cache query suite runtimes. For brevity we omit showing geometric means, which have similar distributions.

Redshift and Vertica Eon represent two performance extremes on the suite. With both a cold and a warm cache, Redshift is magnitudes faster than other systems, despite spending time precompiling each query to machine code before execution. (Compilation typically takes around 3-15 seconds per query.) This is partly a consequence of using local storage, which is fastest but requires a longer initialization time. All other systems have similar performance with the exception of Vertica Eon (S3). These performance characteristics are due to different DBMS design tradeoffs.

Both Vertica EBS and Redshift are shared nothing parallel column-store databases, which implies that they should have similar performance, yet they perform very differently. Three possible reasons for Redshift's performance advantage in our tests are its use of faster NVMe SSD storage, intraquery parallelism, and query compilation to machine code.

Regarding faster storage, Redshift's dc2.8xlarge nodes use NVMe SSD drives marketed to have an aggregate I/O bandwidth of 7.5 GB/sec per node. Based on performance results we received from Snowflake and our own table scan measurements, this advantages it over other systems like Vertica on I/O-intensive workloads. However, it is unlikely to be the main reason on TPC-H, which is rarely I/O constrained. For example, on Vertica EBS, we found that disk read activity was >80% of max throughput only 1% of the entire query suite runtime. Other systems exhibited similar patterns. (We discuss in more detail the effects of faster storage in Section 4.3.3.)

To evaluate the effect of Redshift's intraquery parallelism and query compilation, we ran tests using two randomized, 500GB (uncompressed) source data tables of 10 CHAR(16) columns each, with primary keys on column 1 and partitioned across nodes on column 2 running in a 4-node 8xlarge cluster. A cold start aggregate sum table scan query showed a 3x runtime advantage for Redshift over Vertica EBS, which aligns with NVMe's faster throughput over the EBS 8-volume RAID 0 setup. However, a single user CPU-intensive hash join over two randomized tables with zero resulting tuples showed a 10x performance advantage for Redshift, while the same query with 3 concurrent users ran only 20% slower for Vertica but 300% slower for Redshift. This suggests that Redshift schedules cores in a multicore environment differently that other systems and improves single user performance at the expense of multi-user environments. Additionally, [16] showed that query compilation can provide significant performance benefits for CPU bound queries, so we believe this also advantages Redshift in many TPC-H queries. How much of the performance gains are due to intraquery parallelism vs query compilation cannot be determined with certainty since the system runs as a black box.

While query compilation comes with overhead, an interesting finding was that because Redshift is so much faster than other systems on TPC-H, this overhead never disadvantaged it. On only one query (Q2) was Redshift performance on par with those of other systems. This was a short query consisting of a nested subquery that all systems finished in 20 seconds or less.

At the other performance extreme, Vertica Eon (S3) performs slowest on cold cache. This is because it caches data to local storage aggressively: even a simple table scan summation query caused significant disk writes.

4.3.2 Warm Cache Advantages

One would expect most systems to benefit from a warm cache, but Figure 3 shows that subsequent runs provided little or no performance advantage on the query suite, with the exception of Redshift and Vertica Eon. This section discusses reasons why, along with how generalizable these findings are to other workloads.

A major reason why caching data in memory does not noticeably reduce total suite runtime is because most TPC-H queries are not significantly I/O bound, and those that are consist of shorter queries. However, a workload such as TCP-DS with greater I/O load would see greater benefits from a warm cache. The follow up question then becomes: what is the expected benefit of data caching in an I/O constrained workload? Analyzing only the I/O bound queries shows that caching EBS data into memory provided less than a 2x speedup, and caching S3 data into memory provided up to a 4x speedup, but typically much less. For example, Vertica EBS caches the entire dataset in memory after the first cold run, and no disk reads are observed on subsequent runs, but even on queries that are I/O bound for more than a third of the query runtime there is only a 1.2x to 1.6x speedup. We see a larger improvement for Hive S3, where S3 data is cached. These include speedups of 4x in Q6 (a very short

query that takes 1s on most systems), 2.2x in Q20, 1.6x in Q1, and 1.7x in Q3.

Table 4: Warm Cache Performance Advantages on TPC-H (16 queries)

System	Suite Runtime Speedup	Reasons for Warm Cache Speedup	Number of Queries with Warm Cache Speedup
Vertica (EBS)	1.03x	Data cached into memory	3 queries, ranging from 1.2x to 1.6x speedup
Vertica (S3)	1.70x	S3 data cached into node EBS storage and memory	All
Presto (S3)	1.0x	No advantage	-
Hive (S3)	1.27x	Some S3 data cached in memory	8 queries, ranging from 1.4x to 4x speedup
Hive (EBS)	1.07x	Some data cached in memory	5 queries, ranging from 1.4x to 2.4x speedup
Redshift	1.07x	Avoid query compilation time, some data loaded into memory	14 queries, ranging from 1.5x to over 5x speedup
Spectrum	1.0x	No advantage	-

In a non-I/O constrained suite like TPC-H, the major contributors to warm cache speedups are system designs that benefit subsequent runs at the cost of penalizing initial runs. Redshift's main performance gain on subsequent runs comes from avoiding query compilation time, which is significant relative to its fast TPC-H query runtimes. This relative overhead would be smaller for longer queries. Workloads with identical queries that leverage Redshift's result cache would experience even greater performance improvements. Similarly, Vertica Eon takes a performance hit on cold start because it incurs data writes to node attached storage even if all data could have been cached in memory. In contrast, Presto and Spectrum do not appear to cache data.

4.3.3 *Effects of Faster Storage*

Since TPC-H is not significantly I/O bound, this section analyzes specific I/O bound queries within the suite to better understand the effects of faster storage. Specifically, how much faster is InS over EBS over S3? An interesting thing to note is that a system may have an efficient implementation for one storage type but not for another, so we draw conclusions using within- and cross- system comparisons.

In general, InS performance is faster than EBS, and EBS is faster than S3 on the same system, as expected. However, the magnitude of difference may be exaggerated depending on how well that system utilizes a specific storage type. We dive deeper into three queries: Q1 is an I/O bound query that runs a filtered scan of the lineitem table, which is the largest data table; Q3 joins lineitem to two other smaller tables; and Q5 joins 6 tables including lineitem. Q1 saturates storage I/O throughout, Q3 saturates I/O during the second half of the query, and Q5 saturates I/O during the last third of the query.

We ran limited experiments on the i3 instance family that utilizes physically attached InS (just Vertica), so our results are less conclusive for InS. Instance store performance was similar to the EBS warm cache case where no disk reads are performed, providing only 1.2x speedup on Q1, 1.4x speedup on Q3, and no speedup on Q5 over EBS. This is surprising because an individual NVMe SSD has greater bandwidth (1.9GB/s) than 8 gp2 EBS volumes in RAID 0 (160MB/s per volume), and the i3 system had 4 NVMe SSDS in RAID 0. We further confirmed the AWS listed sequential NVMe read bandwidth by reading a system file sequentially and measured a sequential read bandwidth of 3.5GB/s on the NVMe RAID 0 setup. Therefore, Vertica, at least, seems unable to take full advantage of very fast instance storage.

Comparing Vertica, Hive, and Presto provides insight into EBS performance advantages over S3. Table 5 shows that on Q1, Vertica EBS is 10x faster than Vertica S3, while Hive EBS is only 1.4x faster than Hive S3 and Presto S3. Vertica S3 incurs a performance penalty on cold start by performing writes to node-attached volumes, so the speedup from Vertica S3 to EBS is not representative of faster storage. Additionally, Vertica EBS is 4x faster than Hive EBS, so Vertica EBS's performance advantage over Hive S3 and Presto S3 seems due to other factors like query optimization and partitioning. Analyzing storage I/O for Q1, we see that Vertica EBS disk reads per node are in the 900-1100 MB/s range, whereas Presto S3 reads are in the 400-600 MB/s range, showing Vertica EBS has a 2x storage throughput advantage. Therefore, we conclude that Q1 shows a *2x or less advantage* from using EBS over S3. On Q3, we similarly see that Vertica EBS is 3x faster than Hive EBS, suggesting some non-storage factor is responsible for its advantage. However, Hive S3 is very inefficient on this query and performs even slower than Vertica S3. Presto S3 performs quite well and only 2x slower than Vertica EBS. Thus, if we compare Presto S3 vs Vertica EBS on Q3, there is again a *2x advantage* of EBS. A similar comparison in Q5 shows that the fastest EBS system, Vertica, has about a *1.4x speedup* compared to the fastest S3 system, Presto.

Table 5: Storage I/O bound Query Runtimes (secs) on Fastest to Slowest Storage

Query	Vertica (IS)	Vertica (EBS)	Hive (EBS)	Vertica (S3)	Hive (S3)	Presto (S3)
Q1	12.8	13.5	54.0	130.6	76.2	78.6
Q3	27.8	37.8	106.7	197.4	240.5	83.6
Q5	58.6	60.9	91.3	243.2	179.0	89.3

From these results, a performance advantage from faster storage exists but is often not as large as one would expect. We show in the next section that EBS cost per volume alone is >4x that of S3, with other practical considerations making this difference much larger, so experiencing only a 2x performance degradation is a worthwhile cost-performance tradeoff in many cases.

Summary. Most systems and configurations have comparable performance on the query suite with the exceptions of Redshift and Vertica Eon on cold cache, and Athena (which is a black box and hard to conclude much about). Using cheap remote object storage instead of more expensive EBS block storage seems fine, and even on heavily I/O bound workloads the cost advantage of S3 far exceeds its performance disadvantage. Locally attached instance store on Vertica did not provide significant performance advantage over EBS. Cold and warm cache performance is similar for

most systems on the suite, with the exception of Redshift due to its query compilation time and Vertica Eon due to its aggressive caching of remote data to node storage. However, highly I/O bound workloads may see a 2x or more speedup from data caching. Query compilation in Redshift seems beneficial and feasible for long running queries, and in concert with intraquery parallelism gives it a large performance advantage over other systems in the single-user case.

4.4 Cost

The dollar cost of running a query is another important metric when using a DBMS system on a commercial cloud offering. We consider query cost and storage cost separately, as these are related but can be modified separately via contractual pricing schemes.

For query cost, we report the cost of running the query suite on both cold start as well as subsequent runs. Cold start includes the cost of node initialization along with the cost of running the query suite. Node initialization cost is important to include because the user sometimes pays for time when a system is not yet accessible. For example, Redshift charges while data is being reinitialized from snapshots.

For each system, we report the cost in dollars to run the query suite once for each system using the "on demand" pricing model of AWS.

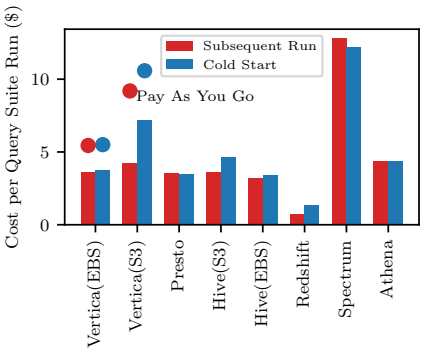
4.4.1 *Query Cost*

Figure 4 shows cold start and subsequent query suite run costs. In general, query cost is directly correlated with query performance and we see similar patterns as before except for the AWS proprietary systems

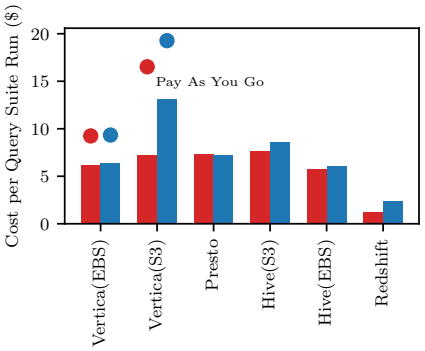
Interestingly, Redshift is both the best and worst performer on query cost. Standard shared-nothing Redshift, which reads data from local node storage, is cheapest because single user queries run extremely fast. However, Redshift Spectrum is about 3x the cost of other options even though its query performance is similar to other systems. This is due to Spectrum's pricing model, which combines both the per hour node cost of Redshift (already the highest of all evaluated systems) plus Spectrum-specific costs per amount of S3 data scanned. Therefore, Redshift is not a cost-effective system if one relies heavily on pulling data from S3.

Unlike other systems, Vertica has a license cost that varies by customer. We therefore plot both the AWS infrastructure costs without licensing as well as Vertica's pay-as-you-go cost, which is offered by the vendor as an additional per-hour software fee over AWS infrastructure costs. The latter represents an extreme high end; most customers likely would own a license and have costs in between the two extremes. Incorporating this cost, Vertica is the second most expensive system behind Spectrum.

Cost is also the only way to have a useful metric by which to compare Athena against other systems, since Athena's infrastructure is a black box making performance comparison to other systems impossible. On cost per query suite, Athena appears similar to other systems. Since Figure 3 shows that Athena is twice as fast as other systems (excepting Redshift) for the 16 query workload, cost/performance makes Athena a good choice for supported workloads in addition to its convenience as an "instant on" serverless system. Therefore, a cloud DBMS user should consider Athena as an option for select workloads and utilize storage that can be accessed by serverless options. However, one caveat to emphasize is that



(a) 16 queries runnable on all systems



(b) All queries runnable on subset of systems

Figure 4: Cold Start and Subsequent Runtime Cost

AWS offers spot and reserve pricing schemes (for long-term contracts) that can lower compute costs significantly. This applies to node costs but not to Athena and Spectrum data scan costs.

4.4.2 *Storage Cost*

Storage cost is less flexible than compute cost because it does not allow recurring to contractual means for cost reduction, such as reserve and spot pricing.

Figure 5 shows the cost of storing workload data over a 24 hour period overlaid with the cold cache query suite runtime. EBS is an order of magnitude more expensive than S3 due to two characteristics. First, the cost of EBS storage is about 5x the cost of S3 storage. Second, one must always allocate more EBS storage than is needed for the actual data. The reason for this is twofold: first, it's impractical to provision an EBS volume size that exactly matches the workload data size (unlike S3); secondly, it is common to provision additional storage in a small EBS volume since AWS's architecture scales up EBS input/output operations per second (IOPs) performance linearly with volume size, until volumes reach 1TB. Replicating data instead of using RAID 0 as we did would increase EBS costs even further. In contrast, S3 costs already include replication within a single region.

Large ephemeral, physically-attached instance stores are becoming widely available in more AWS compute node offerings, making them suitable for use as primary data stores and not just caches. From a performance standpoint, these volumes are physically attached to nodes and hence faster.

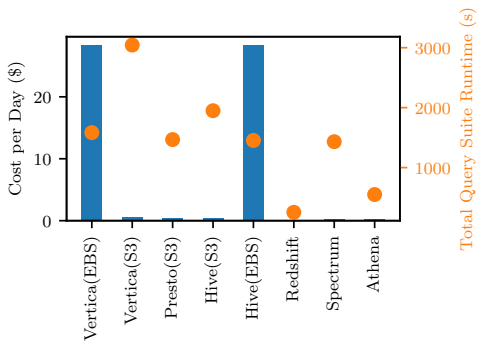


Figure 5: Storage Cost per Day (bars) and Query Suite Runtime (dots) over 16 queries

However, as noted in the previous section, we did not notice a significant performance difference running the TPC-H workload on Vertica, and less than a 2x advantage even on I/O bound queries.

Nodes with instance storage also have a price advantage compared to equivalent nodes that use EBS storage. Figure 6 shows the per hour cost of a r4.8xlarge node and an i3.8xlarge node. From a hardware perspective (RAM, vCPU, network speed), these nodes are equivalent except the i3 node has 4x1900GB volumes of attached InS. We assume the i3 node requires an equivalent volume on S3 for data persistence, while the r4 node using EBS does not. Despite the higher per hour cost of the i3 node and its additional S3 storage cost, it is still cheaper than using EBS.

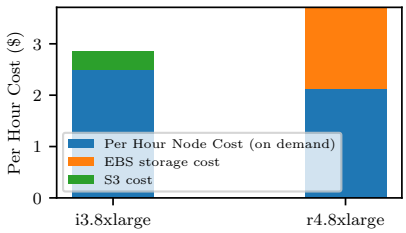


Figure 6: Per Hour Cost of Equivalent Nodes with Instance Storage (i3) and EBS (r4)

Instance stores do come with several disadvantages. First, a user has no control over how many volumes to attach to a node; this setting is predetermined by AWS based on node type. Therefore, node choices are more restricted. Second, they are ephemeral, meaning a user must keep systems always on to not lose data and/or retain data on a secondary storage source for persistence, usually S3. Accordingly, system initialization time after restart is longer because data must be reloaded. Therefore, EBS and S3 are the primary viable persistent storage alternatives while instance store typically serves as a temporary cache.

Summary. Using a persistent remote block storage unit like EBS is orders of magnitude more expensive than S3 without a proportional performance increase. Our setup found a 50x storage cost increase for EBS while only providing a 2x performance speedup on 8 RAID 0 volumes. Additionally, pricing schemes for nodes with instance stores are slightly advantageous over EBS. Therefore, cloud DBMS users should avoid EBS and are better off using instance store and/or S3. Local storage is offered on some node types, but its ephemeral nature makes it challenging to rely on for true shared-nothing configurations in the cloud, making its main purpose local caching of remote data. Athena and Redshift have unique cost models focused on data scan and node uptime, respectively, that advantage them from a cost vs performance standpoint. This is not surprising since they are sold by the platform vendor. However, in a hybrid feature like Spectrum these cost models apply simultaneously and can make it more expensive relative to other systems.

4.5 Data Compatibility with Other Systems

Data compatibility with other systems means that data used by one system can be accessed by another system. Otherwise, there will be significant extract, transform, and load (ETL) costs if one wants to switch to a different system for targeted workloads. Because the cloud offers the ability to easily launch new systems, being able to leverage different systems for different workloads is advantageous. However, ETL costs can make some system types infeasible to use when workloads change, thereby limiting the cloud offerings one can leverage.

		Athena	Vertica		Presto		Hive		Redshift	
			Eon (S3)	EBS	S3	HDFS	S3	HDFS	Red.	Spec.
Athena				LT		L		L	LT	
Vertica	Eon (S3)					L		L	LT	
	EBS	LT			LT		LT		LT	LT
Presto	S3			LT				L	LT	
	HDFS	L	L				L		LT	L
Hive	S3			LT		L			LT	
	HDFS	L	L		L				LT	L
Redshift	Redshift	LT	LT	LT	LT	LT	LT	LT		
	Spectr.			LT		L		L		

Figure 7: Matrix of data compatible systems (L = load required, T = transform required)

Figure 7 presents a matrix of data compatible systems, meaning systems that can directly read data in the same format and storage architecture used by another system. Entries on the diagonal represent the same system and hence should be ignored. An orange shaded cell with the letter L means that data must be transferred to a different storage system, but not transformed. In other words, there are one or more compatible file formats that can be used. A red shaded cell with the letters LT mean that data must be transferred and also converted to a different format. A green cell means that the systems can use compatible formats from the same storage system, for example ORC or Parquet on S3.

Note that we consider some features of the same system as separate systems, such as Vertica shared-nothing vs Vertica Eon, and Redshift vs Spectrum, since these have different cost-performance characteristics. For example, Redshift can read data from all other systems with the proper storage and data configurations (i.e., not HDFS) by using its Spectrum feature, but a user must perform ETL to have Redshift-level performance rather than Spectrum-level performance.

Systems that use general data formats like ORC on S3 and HDFS are most compatible with other systems. These include Hive, Presto, and Vertica. AWS systems support data on S3 but not on HDFS, limiting their compatibility if an enterprise has much of its data on HDFS. These AWS systems include Athena and Spectrum. The most performant systems use proprietary storage formats that make them incompatible with other systems but offer hybrid architectures to read data on S3. Redshift has this limitation, and one could use its Spectrum feature to read data from S3. Similarly, Vertica offers Eon mode. But as previously shown, the S3 features on these systems move them to a different performance-cost curve.

One caveat about Vertica Eon is that while it can read ORC and other system agnostic formats on S3, our experiments indicated it performed best on its own format written to S3, and this is what was benchmarked. Hence this setup would incur data loading and transformation costs if a user wants to use another system.

Summary. Choosing a widely compatible columnar data format and storage architecture provides more options to optimize performance if workloads change by making it easy to run different systems. AWS proprietary and shared-nothing systems tend to be less data-compatible than others.

4.6 Scalability

We analyzed performance when scaling horizontally and vertically. The systems tested were more limited for these experiments because AWS proprietary systems had fewer scalability options.

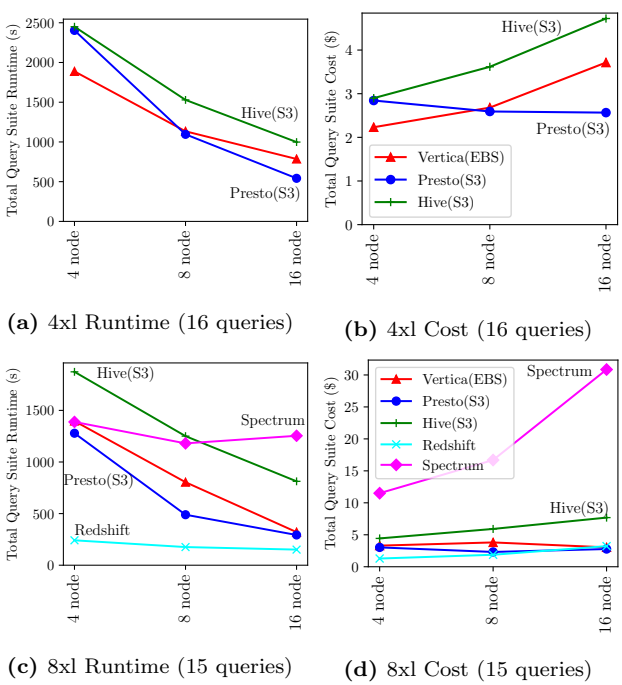
The main theme is that horizontal scaling is generally advantageous, while vertical scaling is generally disadvantageous. We did not perform scale up on AWS proprietary systems due to node type limitations.

4.6.1 Horizontal Scaling

Many systems do not scale out linearly, meaning runtime does not halve when using twice as many nodes. Presto scales horizontally especially well, with better than linear performance when using smaller nodes because more memory and CPU equals better performance. A shared-nothing OLAP database like Vertica also scales horizontally well. In contrast, a system that performs query compilation like Redshift exhibits poor performance when scaling horizontally on dissimilar query workloads that require recompilation because this overhead becomes significant relative to query runtime.

Figures 8a and 8b plot query suite runtime and cost, respectively, as we scale from 4 to 8 to 16 nodes with 4xl nodes. In the ideal scaling case, every tick on the x-axis of Figure 8a should result in halving the runtime. These results omit Redshift because it is not offered in this instance size (we will present other Redshift scale out results shortly). Presto comes closest to scaling linearly while Hive and Vertica are less scalable. Athena is excluded because it provides no control over resource allocation. In Figure 8b a system scales linearly if its graph is a horizontal line.

The good scalability of Presto indicates that it parallelizes effectively. Our evaluation suggests that the system is CPU-bound, since few disk writes to node storage were observed regardless of the configuration.



(a) 4xl Runtime (16 queries) **(b)** 4xl Cost (16 queries)

(c) 8xl Runtime (15 queries) **(d)** 8xl Cost (15 queries)

Figure 8: Horizontal Scaling, 4xl and 8xl instance size clusters

A different story emerges when we run the same experiments on nodes with double the CPU and RAM, specifically the 8xl size. Figure 8c shows that both Hive and Vertica scale linearly while Presto does not. Note in Figure 8d that Vertica is a horizontal line while both Hive and Presto increase somewhat. In both plots, we dropped Q17 from the query suite because Vertica had a severe performance bug on this query. In this case, Presto scalability gains begin visibly plateauing at 16 nodes unlike with the smaller nodes.

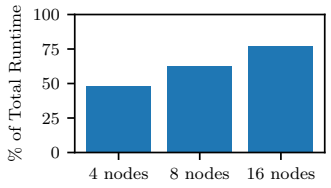
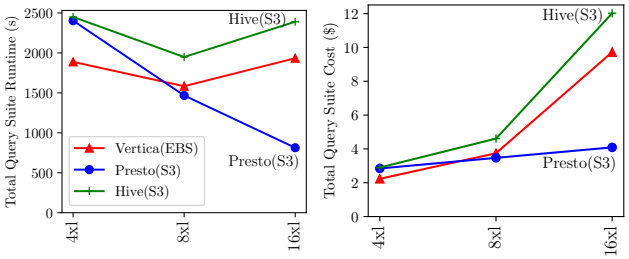


Figure 9: Redshift Query Compilation Time (% of total runtime)

We included Redshift and Spectrum in Figures 8c and 8d because they run on hardware similar to the larger nodes for the other systems. These results include Redshift query compilation time, which is representative of running ad hoc queries. Both systems exhibit essentially no scalability. For Redshift, this is because query compilation time becomes the bottleneck given how fast it runs the single-user TPC-H queries, and AWS states that this overhead is "especially noticeable" for ad hoc queries [2]. Query compilation times were in the range of 3-15 seconds per query so query compilation time encompasses an ever increasing percentage of the faster total query execution time as the cluster size is increased, as shown in Figure 9. Note that we see more

scalable performance for repeated executions of a query if it is already compiled.

4.6.2 *Vertical Scaling*



(a) Runtime **(b)** Cost

Figure 10: Vertical Scaling, 4 node cluster (16 queries)

In our limited test set of three systems, vertical scaling was disadvantageous with the exception of Presto. Scaling measurements for Redshift are not shown because it has only two node sizes and the smaller size did not have enough storage in the 4 node configuration. Athena has no notion of scale up.

Presto appears to again benefit from more cores even in the scale up case, as few disk writes to node storage were observed and network traffic from S3 was not saturated the vast majority of time.

Summary. Most systems exhibit performance benefits from horizontal scaling, with Spectrum being the exception. Vertical scaling tests suggest that larger nodes are generally disadvantageous once moderate to large nodes are already used.

5. CONCLUSIONS

Cloud economics and architectures promote fundamental shifts in DBMS design and usage. Being able to choose the optimal configuration and leverage the full breadth of DBMS-as-a-service offerings can provide significant cost and performance benefits.

Our TPC-H benchmarking experiments suggest that cloud DBMS users should prioritize shared, low-cost block storage architectures like S3 over more expensive shared block storage volumes like EBS that provide lower latency. Low cost object storage provides order of magnitude cost advantages with minimal performance disadvantages, depending on the DBMS architecture. Additionally, locally attached instance storage are faster than EBS workloads and are more cost effective than EBS, so there are few reasons to use EBS over these options.

A carefully chosen general use columnar format like ORC provides the most flexibility for future system optimization over proprietary storage formats used by shared nothing DBMSs such as Redshift and Vertica. Such system agnostic formats appear performant even without highly optimized partitioning. For shared nothing systems utilizing proprietary formats, hybrid features (Spectrum and Eon, respectively) aim to bridge this gap by allowing reading other formats from S3, but their performance is currently lacking and cost is high when multiple cost models apply to the same query, as in the case of Spectrum.

Different system architectures used in proprietary cloud offerings highlight interesting tradeoffs when they are compared to non-proprietary systems. For example, the aggressive intraquery parallelism of Redshift can offer an order of magnitude performance advantage for single user workloads, but doing so causes significantly worse performance as concurrency increases. Similarly, query compilation to machine code performed by Redshift speeds up CPU-intensive queries but reduces scale out performance gains on heterogenous workloads. It would be interesting for future studies to determine whether implementing query compilation and aggressive intraquery parallelism allows other S3 systems to achieve near Redshift-level performance without local storage.

Serverless systems like Athena have been recently introduced and are becoming popular. Athena's on demand querying capabilities provide an interesting optimization opportunity for cloud DBMSs to farm out different workloads, which is another reason for choosing a general use columnar data format over proprietary formats.

Each of these findings poses opportunities for future work to explore specific architectural tradeoffs further. Additionally, future studies could analyze concurrency, test a different suite such as TPC-DS, evaluate different data sizes, and evaluate more systems.

6. ACKNOWLEDGMENTS

Amazon kindly provided a very large number of AWS credits that we used to perform this study. We also thank Ben Vandiver, Nga Tran, and the Vertica team; Ippokratis Pandis at Amazon; and the Starburst Data team for feedback and guidance on Vertica, Redshift/Spectrum, and Presto, respectively.

7. REFERENCES

[1] O. Agmon Ben-Yehuda, M. Ben-Yehuda, A. Schuster, and D. Tsafrir. Deconstructing amazon ec2 spot instance pricing. *ACM Trans. Econ. Comput.*, 1(3):16:1–16:20, 2013.

[2] AWS. Redshift documentation: Factors affecting query performance, 2012. `https://docs.aws.amazon.com/redshift/latest/dg/c-query-performance.html`, Last accessed 2018-06-15.

[3] AWS. Cluster configuration guidelines and best practices, 2019. `https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-plan-instances-guidelines.html`, Last accessed 2019-02-01.

[4] D. Bermbach, J. Kuhlenkamp, A. Dey, A. Ramachandran, A. Fekete, and S. Tai. Benchfoundry: A benchmarking framework for cloud storage services. In *Proc. Int. Conf. on Service-Oriented Computing (ICSOC)*, pages 314–330, 2017.

[5] C. Binnig, D. Kossmann, T. Kraska, and S. Loesing. How is the weather tomorrow?: Towards a benchmark for the cloud. In *Proc. Second Int. Workshop on Testing Database Systems*, DBTest '09, pages 9:1–9:6, 2009.

[6] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. Benchmarking cloud serving systems with YCSB. In *Proc. 1st ACM Sym. on Cloud Computing*, SoCC '10, pages 143–154, 2010.

[7] B. Dageville, T. Cruanes, M. Zukowski, V. Antonov, A. Avanes, J. Bock, J. Claybaugh, D. Engovatov,

M. Hentschel, J. Huang, A. W. Lee, A. Motivala, A. Q. Munir, S. Pelley, P. Povinec, G. Rahn, S. Triantafyllis, and P. Unterbrunner. The snowflake elastic data warehouse. In *Proc. 2016 Int. Conf. on Management of Data*, SIGMOD '16, pages 215–226, 2016.

[8] Databricks. Benchmarking Big Data SQL Platforms in the Cloud: TPC-DS benchmarks demonstrate Databricks Runtime 3.0's superior performance, 2017. `https://databricks.com/blog/2017/07/12/benchmarking-big-data-sql-platforms-in-the-cloud.html`, Last accessed 2018-07-15.

[9] M. Dayarathna and T. Suzumura. Graph database benchmarking on cloud environments with XGDBench. *Automated Software Eng.*, 21(4):509–533, 2014.

[10] A. Gupta, D. Agarwal, D. Tan, J. Kulesza, R. Pathak, S. Stefani, and V. Srinivasan. Amazon Redshift and the case for simpler data warehouses. In *Proc. of the 2015 Int. Conf. Management of Data*, SIGMOD '15, pages 1917–1923, 2015.

[11] M. Han, K. Daudjee, K. Ammar, M. T. Özsu, X. Wang, and T. Jin. An experimental comparison of Pregel-like graph processing systems. *PVLDB*, 7(12):1047–1058, 2014.

[12] Hortonworks. Apache Tez: Overview, 2018. `https://hortonworks.com/apache/tez/`, Last accessed 2018-08-01.

[13] D. Kossmann, T. Kraska, and S. Loesing. An evaluation of alternative architectures for transaction processing in the cloud. In *Proc. of the 2010 Int. Conf. Management of Data*, SIGMOD '10, pages 579–590, 2010.

[14] A. Lamb, M. Fuller, R. Varadarajan, N. Tran, B. Vandiver, L. Doshi, and C. Bear. The Vertica analytic database: C-store 7 years later. *PVLDB*, 5(12):1790–1801, 2012.

[15] A. Lenk, M. Menzel, J. Lipsky, S. Tai, and P. Offermann. What are you paying for? performance benchmarking for infrastructure-as-a-service offerings. In *Proc. 2011 IEEE 4th Int. Conf. on Cloud Computing*, CLOUD '11, pages 484–491, 2011.

[16] T. Neumann. Efficiently compiling efficient query plans for modern hardware. *PVLDB*, 4(9):539–550, 2011.

[17] N. Nix. CIA tech official calls Amazon cloud project 'transformational'. *Bloomberg*, June 2018. `https://www.bloomberg.com/news/articles/2018-06-20/cia-tech-official-calls-amazon-cloud-project-transformational`, Last accessed 2018-10-01.

[18] J. Schad, J. Dittrich, and J.-A. Quiané-Ruiz. Runtime measurements in the cloud: Observing, analyzing, and reducing variance. *PVLDB*, 3(1-2):460–471, 2010.

[19] R. Sethi, M. Traverso, D. Sundstrom, D. Phillips, W. Xie, Y. Sun, N. Yigitbasi, H. Jin, E. Hwang, N. Shingte, and C. Berner. Presto: SQL on everything. In *IEEE 35th Int. Conf. on Data Eng. (ICDE)*, pages 1802–1813, 2019.

[20] A. Shiu. Why we chose Redshift, 2015. `https://amplitude.com/blog/2015/03/27/why-we-chose-redshift`, Last accessed 2018-11-05.

[21] M. Stonebraker, A. Pavlo, R. Taft, and M. L. Brodie. Enterprise database applications and the cloud: A difficult road ahead. In *2014 IEEE Int. Conf. Cloud Eng.}, pages 1–6, 2014.*

[22] D. Sundstrom. Even faster: Data at the speed of Presto ORC, 2015. `https://code.fb.com/core-data/even-faster-data-at-the-speed-of-presto-orc/`, Last accessed 2018-04-15.

[23] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy. Hive: A warehousing solution over a map-reduce framework. PVLDB*, 2(2):1626–1629, 2009.*

[24] B. Vandiver, S. Prasad, P. Rana, E. Zik, A. Saeidi, P. Parimal, S. Pantela, and J. Dave. Eon mode: Bringing the Vertica columnar database to the cloud. In Proc. 2018 Int. Conf. Management of Data*, SIGMOD '18, pages 797–809, 2018.*

[25] B. Varghese, O. Akgun, I. Miguel, L. Thai, and A. Barker. Cloud benchmarking for performance. In IEEE 6th Int. Conf. Cloud Computing Technology and Science*, pages 535–540, 2014.*

[26] B. Varghese, L. T. Subba, L. Thai, and A. Barker. Container-based cloud virtual machine benchmarking. In IEEE Int. Conf. on Cloud Eng. (IC2E)*, pages 192–201, 2016.*

[27] A. Verbitski, A. Gupta, D. Saha, M. Brahmadesam, K. Gupta, R. Mittal, S. Krishnamurthy, S. Maurice, T. Kharatishvili, and X. Bao. Amazon Aurora: Design considerations for high throughput cloud-native relational databases. In Proc. ACM Int. Conf. on Management of Data*, SIGMOD '17, pages 1041–1052, 2017.*

[28] Vertica. Configuring storage (documentation). `https://www.vertica.com/docs/9.1.x/HTML/index.htm#Authoring/UsingVerticaOnAWS/ConfiguringStorage.htm`, Last accessed 2019-01-13.