AMD EPYC

WHITE PAPER

aws

UNLOCK THE POWER OF DATA ANALYTICS WITH AMAZON EC2 M7A & R7A INSTANCES POWERED BY 4TH GEN AMD EPYC™ CPUS

AMD
together we advance_data center computing

Second Edition
July 2024

Seong Kim
Venkat Ranganathan

UNLOCK THE POWER OF DATA ANALYTICS WITH AMAZON EC2 M7A & R7A INSTANCES POWERED BY 4TH GEN AMD EPYC™ CPUS

CONTENTS

INTRODUCTION

A growing number of public and private sectors rely on data analytics tools that are being tasked with processing ever larger datasets from a proliferating number of sources. This relentless expansion is a crucial concern for computing infrastructures. Scaling out CPU does not always meet these demands due to both cost and non-linear scaling properties. This white paper demonstrates the value of using cloud instances powered by 4th Gen AMD EPYC™ processors using benchmarking results based on the TPC-C™ Benchmark Standard (TPC-C) and TPC Benchmark DS (TPC-DS). It also presents the AMD approach to the unified interface between Java®-based query engines, such as Spark® SQL and native acceleration libraries.

Various data acceleration approaches are also reshaping the data analytics landscape. The need for scalable and flexible solutions that can efficiently handle large volumes of diverse data is driving a shift from traditional software-only database processing models to software plus accelerations. This transition brings its own challenges, such as identifying the types of accelerators needed and the need for standard unified interfaces to the accelerators. Nevertheless, the momentum toward accelerators underscores the dynamic complexity of databases in the rapidly evolving field of data analytics.

The Omdia Server Workload Tracker 2023 report reveals that database and analytics are the second-largest server workloads, which demonstrates both the popularity and critical need for high-performance platforms. See Figure 1.

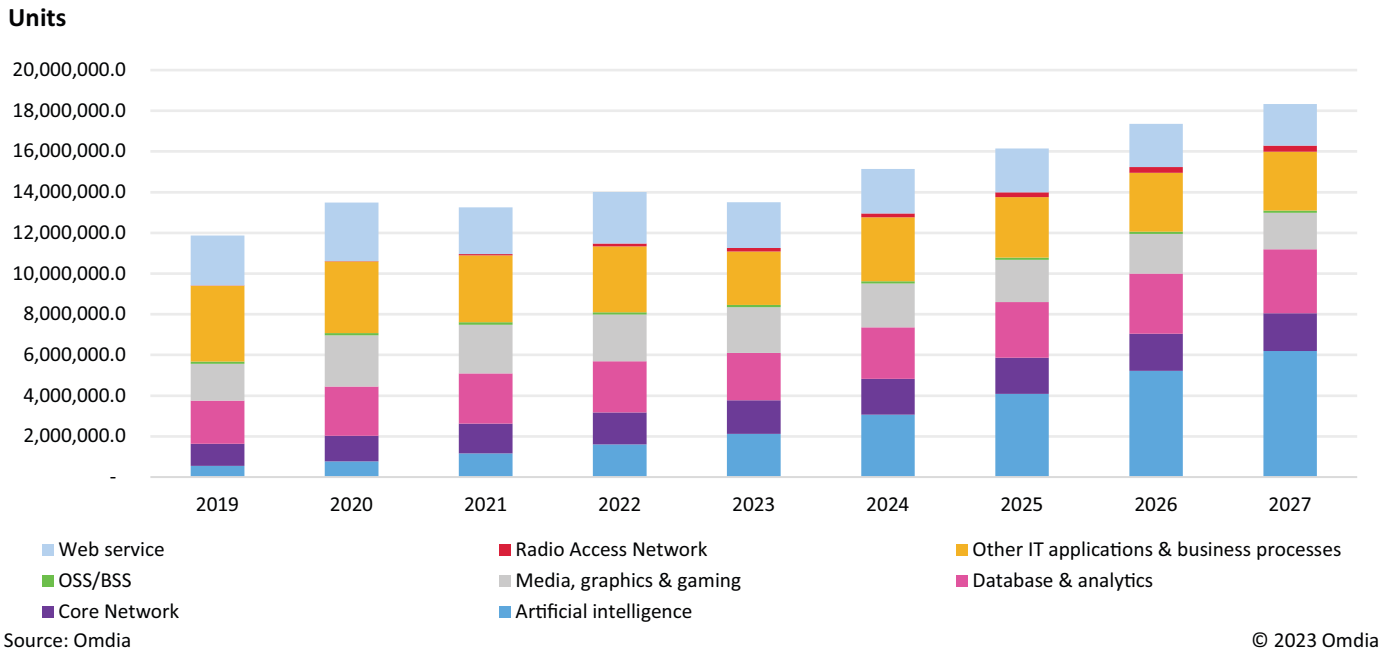


Figure 1: Omdia Server Workload Tracker 2023 (reported on June 30th, 2023)

AMD identified several database acceleration methods, as shown in Figure 2.

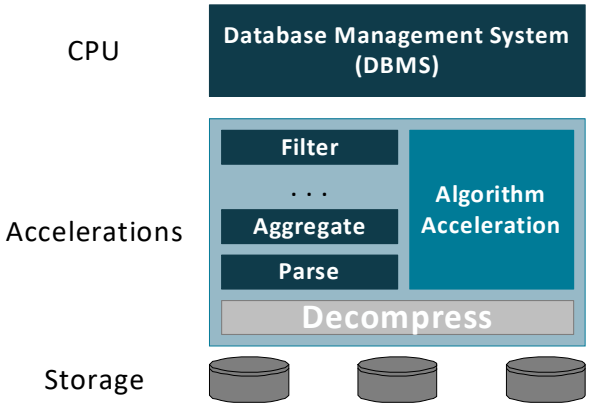


Figure 2: Database acceleration methods

These methods are:

- **Data management acceleration:** This includes the following essential functions:
  - Data decompression (Gzip, LZ4, zlib).
  - Data parser (CSV/JSON parser).
  - Format conversion (e.g., row to column, parquet to arrow).
  - Data cache.
- **Basic database primitive accelerations:** Examples include filter, aggregate, hash, JOIN, and sort.
- **Algorithm acceleration:** For compute intense operations such as:
  - Data analytics, i.e., data mining, text processing).
  - Graph processing, e.g., similarity analysis (cosine similarity), community detection, search, etc.

These accelerations can be achieved via various mechanisms, including CPU instruction sets or external accelerators. This white paper focuses on a CPU-based approach. First, let's briefly review Apache® Spark®.

ABOUT APACHE® SPARK ®

Apache Spark is a popular distributed data processing engine designed for speed, ease of use, and sophisticated analytics, with APIs in Java, Scala, Python, R, and SQL. Spark is an open-source framework focused on interactive queries, machine learning, and real-time workloads. It runs analytics on storage systems such as HDFS, Amazon Redshift, Amazon S3, and Apache Cassandra®.

Spark provides an optimized framework for fast iterative processing, such as machine learning and interactive data analysis, while retaining the scalability and fault tolerance of Apache Hadoop® MapReduce. Spark addressed MapReduce limitations by processing in-memory and caching, reducing the number of steps in a job, and by reusing data across multiple parallel operations. Figure 3 shows the high-level Spark components.

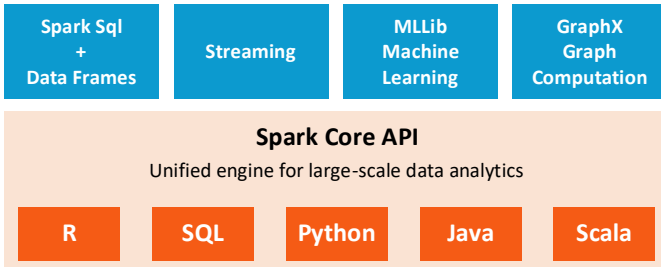


Figure 3: Spark components

TPC ANALYSIS OVERVIEW

The Transaction Processing Performance Council® (TPC®) created industry-standard benchmarks for evaluating the performance of databases and hardware. Some popular TPC benchmarks include TPC-C™ Benchmark Standard for Online Transaction Processing (OLTP), TPC-H Benchmark Standard (TPC-H), and TPC Benchmark-DS (TPC-DS) for decision support.

TPC-DS and TPC-H are the most popular database performance benchmarking tools and have a mix of OLTP and Online Analytical Processing (OLAP). Apache Spark supports all ninety-nine TPC decision support (DS) queries and twenty-two TPC-H queries. Thus, running the TPC-DS and TPC-H benchmarks is a good way to demonstrate the key performance matrix of target platforms.

TPC-H ANALYSIS

The TPC-H Benchmark Standard (TPC-H) is a suite of business-oriented ad-hoc queries and concurrent data modifications selected for wide industry relevance. This benchmark tests decision support systems that analyze large datasets, execute complex queries, and answer critical business questions and reports results in composite Queries per Hour (QphH@Size). The tests described in this white paper were derived from the TPC-H Benchmark Standard and as such are not comparable to published TPC-H results, as the results do not comply with the TPC-H Benchmark Standard.

AMD used TPC-H queries to profile a 1TB dataset (SF1000) to determine the percentage of total runtime spent performing various tasks, as shown in Table 1.

Function	Runtime %	Description
scan time	19.62%	scan decode
scan_decomp	14.04%	scan decompress
wscg_time	13.43%	wholestage code
time in aggregation process	11.47%	aggregation
totaltime_decompress	5.27%	data decompress
totaltime_compress	5.17%	shuffle compress
totaltime_split	5.10%	shuffle split
join time	5.05%	JOIN
time in sort process	4.73%	sorting
shuffle write time	4.13%	
totaltime_condproject	2.55%	
shuffle spill time	2.17%	
totaltime_collectbatch	1.94%	
time_to_read_io	1.79%	
totaltime_computepid	1.03%	shuffle pid
other misc	2.51	
Total	**100%**	

Table 1: TPC-H profiling report at 1TB

This analysis allows one to determine the functions that can benefit most from acceleration. In this case:

- The first set of target functions used up 33.66% of the total runtime. This set includes scan decode and scan decompress:
- The second set of target functions used up 6.13% of the total runtime. This set includes shuffle split and shuffle pid.
- The third set of target functions used up 34.68% of the total runtime. This set includes wholestage code, Aggregation/GBY, JOIN, and sorting.

In this case, accelerating compression and scanning can yield substantial performance gains.

TPC-DS ANALYSIS AND BENCHMARKING

The TPC Benchmark-DS (TPC-DS) models the general components of a decision support system that includes queries and maintenance of data and reports results in query throughput per second. It provides a representative performance evaluation for decision support systems. The tests described in this white paper were derived from the TPC-DS Benchmark Standard and as such are not comparable to published TPC-DS results, as the results do not comply with the TPC-DS Benchmark Standard.

TPC-DS provides highly comparable, controlled, and repeatable tasks for various database implementations. It consists of three main operations: load, query run, and data maintenance. TPC-DS has a total of 99 queries with some additional queries branched out from the main queries. The operations defined by TPC-DS appear in Table 2.[1]

SQL Feature	# of TPC-DS Queries
Common Sub-Expression	31
Correlated Sub-Query	15
Uncorrelated Sub-Query	76
Group By	78
Order By	64
Rollup	9
Partition	11
Exists	5
Union	17
Intersect	2
Minus	1
Case	24
Having	5

Table 2: Number of TPC-DS queries by SQL feature

TPC-DS works with very large FACT tables that store data transactions. For example, consider a retail supermarket that retains records of each customer purchasing tens of items. In this example, the customer names, produce, and prices become a FACT table.

The nature of this FACT table makes Extract, Transformation, and Load (ETL) a big component of TPC-DS benchmarking. Faster storage IO can improve overall performance against slower storage IO. Faster decompression capabilities can substantially improve performance compared to slower capabilities. Table 2 notes that the Group By operation is the most common TPC-DS query. Group By requires several basic primitives, such as avg, sum, min, max, and shuffle. Of these, shuffle exchanges data with other nodes and is therefore very sensitive to network bandwidth. A shuffle operation that consumes the bulk of end-to-end operation time may limit the value of any acceleration can be diminished. TPC-DS is similar to TPC-H in that the TPC-H profiling shown in Table 1 shows that shuffle operations do consume a substantial portion of the overall runtime.

TEST CONFIGURATION AND RESULTS

All tests described in this white paper were completed as of January 10th, 2024, and were performed on the Amazon EC2 instance types shown in Table 3.

	m7a.4xlarge r7a.4xlarge	m7g.4xlarge r7g.4xlarge	m7i.4xlarge r7i.4xlarge
Operating System	Ubuntu® 22.04 LTS		
CPU	AMD EPYC 9R14	AWS Graviton3	Intel® Xeon® Platinum 8488C
Cores / Threads	16/16	16/16	8/16
Clock Base | Boost[2]	2.6/3.7	2.6	Up to 3.2
L3 Cache per CPU	384MB	64MB	105MB
Network bandwidth (Gbps)	Up to 12.5	Up to 15	Up to 12.5
EBS Bandwidth (Gbps)	Up to 10		
For Additional Information	Please see Amazon EC2 m7a Instances* and Amazon EC2 r7a Instances*		

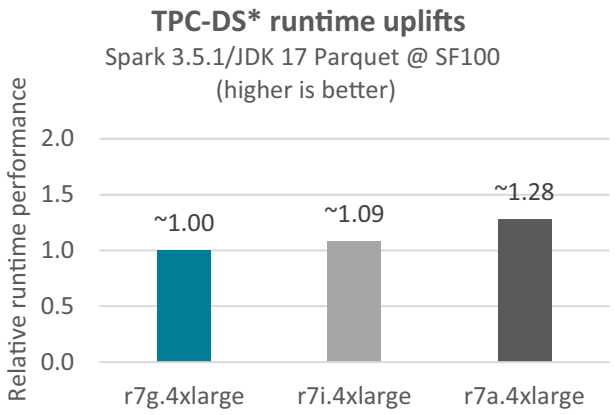
Table 3: Amazon EC2 instances used for the testing described in this white paper

These tests used the following parameters:

- **Data format:** Apache Parquet.
- **Scale factors:** Various; the results listed below used SF 50.
- **Tools:** Spark 3.5 with JDK 17.

TEST RESULTS

AMD collected the following results after running the entire suite of TPC-DS queries and measured the runtime of the three instances described in Table 3, above. See Figure 4



**These tests were derived from the TPC-DS Benchmark Standard and as such are not comparable to published TPC-DS results, as the results do not comply with the TPC-DS...*

Figure 4: TPC-DS runtime performance on Amazon EC2 r7a, r7g, and r7i instances @ SF100

Figure 4 shows that the AMD EPYC processor-powered Amazon EC2 r7a.4xlarge instance delivers performance uplifts of ~1.39x versus the Intel-powered Amazon EC2 r7i.xlarge instance and ~1.28x versus the Graviton-powered Amazon EC2 r7g.4xlarge instance. These uplifts are primarily due to the following factors:

- The AMD EPYC processor runs at higher base and boost frequencies than the competing processors, which speeds up database operations.
- The much larger L3 cache in the AMD EPYC processor compared to the competing processors helps speed up the scan, filter, JOIN, and comp/decomp operations.

UNIFIED ACCELERATION INTERFACE FOR SPARK

Spark is a popular tool because it can handle large datasets. However, the Spark community saw a need to address performance issues by implementing various optimizations over time. Many such attempts were made, including Gazelle, which improved performance by leveraging processor AVX instructions. Even so, Gazelle support ended* in February, 2023 because of a lack of community participation.

The open-source community has shifted to Gluten ("glue" in Latin), which is an Apache-supported open-source framework. It optimizes database queries using columnar stores and vectorized processing. Gluten transforms the entire Spark stage physical plan to Substrait plans and offloads performance-critical data processing to the native library. Gluten also defines extensibility to support more native accelerators.

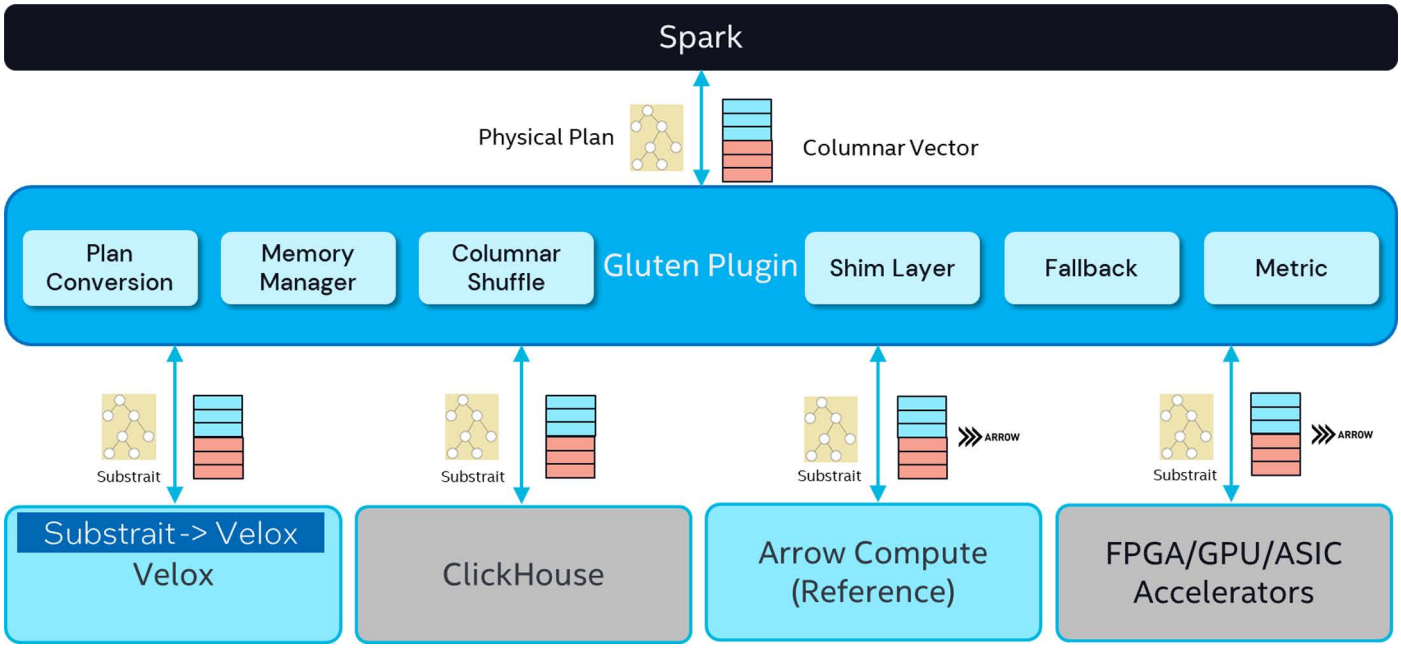


Figure 5: Gluten architecture (source: https://github.com/oap-project/gluten)*

TEST RESULTS USING GLUTEN

AMD tested the Gluten performance gain on an Amazon EC2 m7a.4xlarge instance configured as shown in Table 3, above running Spark 3.3, JDK 8, and Gluten 1.1.0 with SF50. See Figure 6.

Figure 6 shows that Gluten delivers a performance uplift of ~2.20x compared to running Spark without Gluten. Gluten does this by removing extra overhead, such as format conversion.

This bears further exploration. Let's use TPC-DS query 9 as a good example. This query creates 5 buckets of store sales discount metrics. 15 queries are defined on the same FACT table with 5 distinct filters. The Gluten optimizer can reduce this to only 5 query executions instead of 15 in the generated physical plan. Spark and Gluten execute the physical plan with physical operators, as shown in Figure 7.

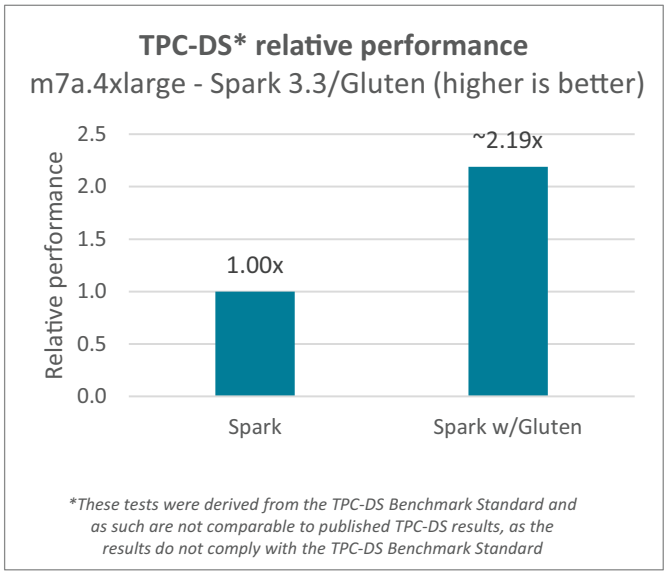


Figure 6: TPC-DS runtime and relative performance on Amazon EC2 m7a.4xlarge instance with and without Gluten @ SF50

```
select
case when (select count(*) from store_sales where ss_quantity between 1 and 20 ) > 74129
       then (select avg(ss_ext_discount_amt) from store_sales where ss_quantity between 1 and 20 )
       else (select avg(ss_net_paid) from store_sales where ss_quantity between 1 and 20 ) end bucket1.
case when (select count(*) from store_sales where ss_quantity between 21 and 40 ) > 122840
       then (select avg(ss_ext_discount_amt) from store_sales where ss_quantity between 21 and 40 )
       else (select avg(ss_net_paid) from store_sales where ss_quantity between 21 and 40 ) end bucket2.
case when (select count(*) from store_sales where ss_quantity between 41 and 60 ) > 56580
       then (select avg(ss_ext_discount_amt) from store_sales where ss_quantity between 41 and 60 )
       else (select avg(ss_net_paid) from store_sales where ss_quantity between 41 and 60 ) end bucket3.
case when (select count(*) from store_sales where ss_quantity between 61 and 80 ) > 10097
       then (select avg(ss_ext_discount_amt) from store_sales where ss_quantity between 61 and 80 )
       else (select avg(ss_net_paid) from store_sales where ss_quantity between 61 and 80 ) end bucket4.
case when (select count(*) from store_sales where ss_quantity between 81 and 100 ) > 165306
       then (select avg(ss_ext_discount_amt) from store_sales where ss_quantity between 81 and 100 )
       else (select avg(ss_net_paid) from store_sales where ss_quantity between 81 and 100 ) end bucket5.
from reason
where r_reason_sk = 1
```

Figure 7: TPC-DS Query 9

SPARK EXECUTION OF THE TPC-DS Q9 PLAN

Standalone Spark (without Gluten) scans data from the Parquet tables (Blocks 1 and 4 in Figure 8) in a columnar fashion. Next, Spark converts the data to a row format and performs further processing in a Spark execution stage (Blocks 2 and 5 in Figure 8). A shuffle occurs as part of executing the hash aggregation (Block 3 in Figure 8). AMD testing showed a total runtime for the default Spark execution of ~23.54 seconds using a SF50 Parquet dataset.

The same physical plan is transformed for optimal execution when processed by the Gluten framework. The file scan, projection, and filtering happen with the data in columnar format without conversion to row formats (see Blocks 1 & 2 and 4 & 5 of Figure 9). The shuffle that is part of executing the hash aggregation occurs in columnar format. The conversion from columnar to row format only happens when the data is passed back to Spark to show the results (Block 6 of Figure 9). AMD testing showed that this optimized execution of the same SF50 Parquet dataset reduces runtime to ~14.06 seconds—an improvement of ~67%.

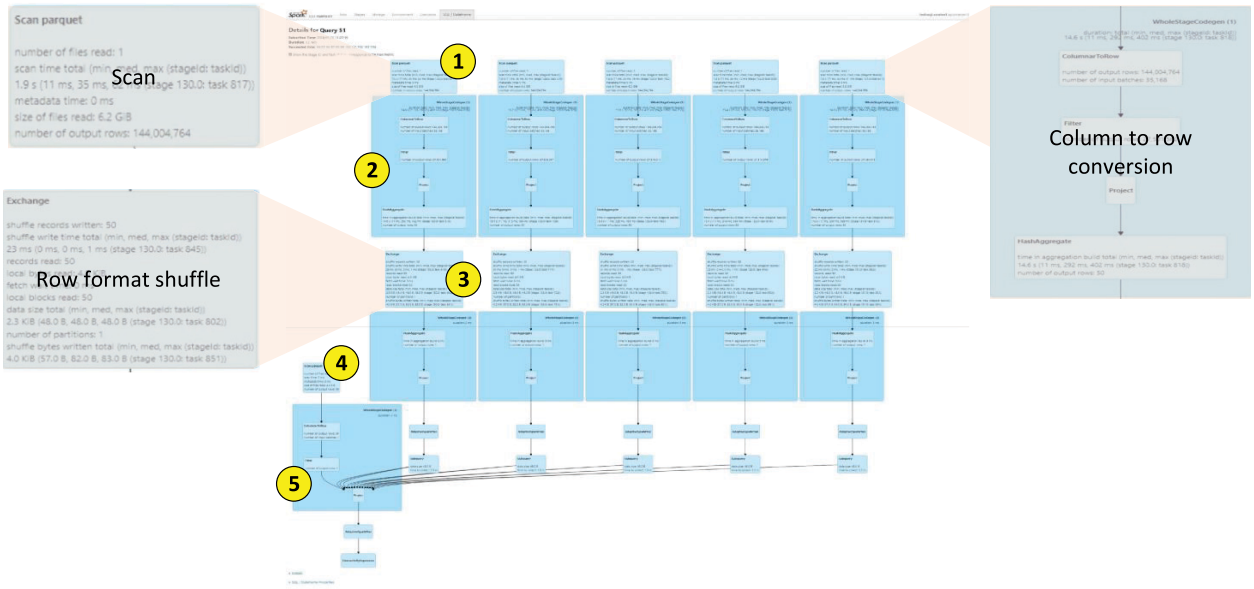


Figure 8: TPC-DS Query 9 – Spark execution

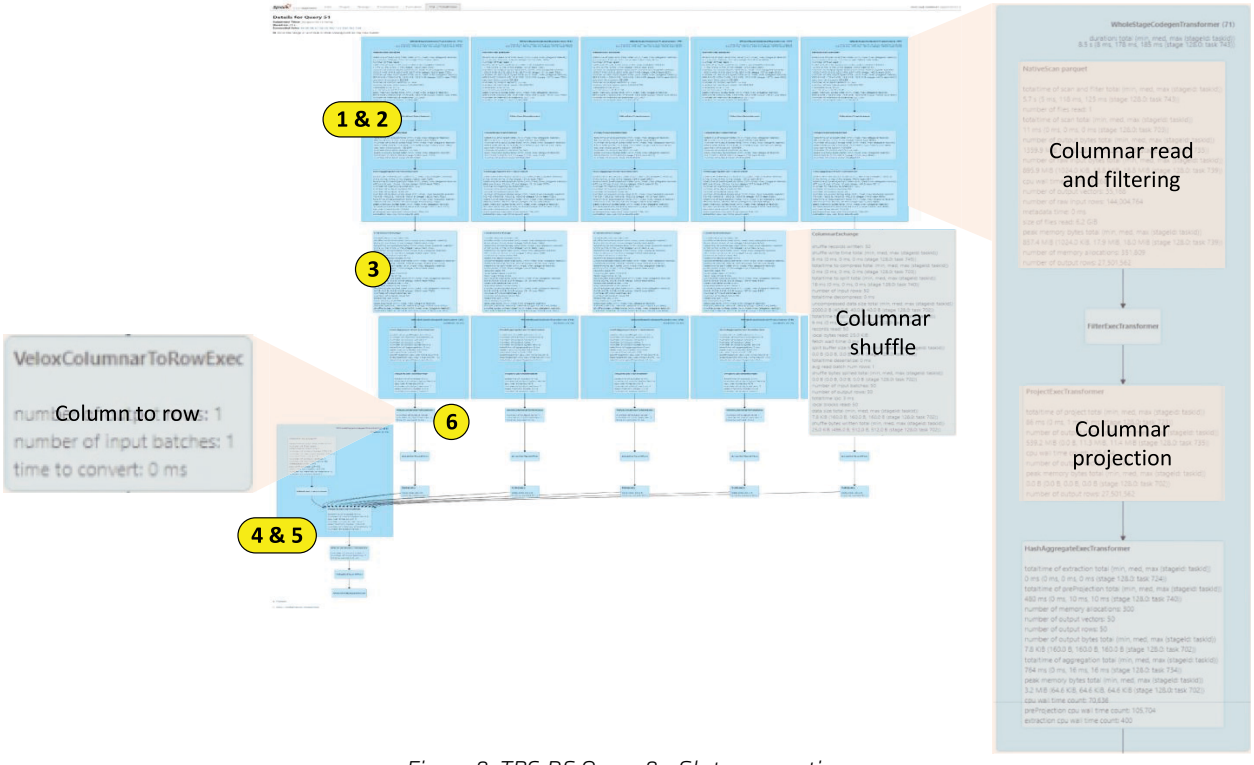


Figure 9: TPC-DS Query 9 - Gluten execution

CONCLUSION

This white paper analyzed the computationally expensive TPC-H and TPC-DS workloads and demonstrated the superior performance of Amazon EC2 m7a.4xlarge instances powered by 4th Gen AMD EPYC processors versus comparable Amazon EC2 m7 instances powered by competing processors. It also discussed the industry approach to Spark acceleration with a special focus on Gluten and showed how Gluten can greatly accelerate Spark database queries.

REFERENCES

1. Meikel Poess, Raghu Nambiar, David Walrath, "Why You Should Run TPC-DS: A Workload analysis," ACM. VLDB '07, September 23-28, 2007
2. Maximum boost for AMD EPYC processors is the maximum frequency achievable by any single core on the processor under normal operating conditions for server systems. EPYC-18

Seong Kim, Ph.D. is a Sr. Director of Cloud Solutions Architecture at AMD.
Venkat Ranganathan is a Database Solutions Architect at AMD.

DISCLAIMERS

*Links to third party sites are provided for convenience and unless explicitly stated, AMD is not responsible for the contents of such linked sites and no endorsement is implied.

The information contained herein is for informational purposes only and is subject to change without notice. While every precaution has been taken in the preparation of this document, it may contain technical inaccuracies, omissions and typographical errors, and AMD is under no obligation to update or otherwise correct this information. Advanced Micro Devices, Inc. makes no representations or warranties with respect to the accuracy or completeness of the contents of this document, and assumes no liability of any kind, including the implied warranties of noninfringement, merchantability or fitness for purposes, with respect to the operation or use of AMD hardware, software or other products described herein. No license, including implied or arising by estoppel, to any intellectual property rights is granted by this document. Terms and limitations applicable to the purchase or use of AMD's products are as set forth in a signed agreement between the parties or in AMD's Standard Terms and Conditions of Sale. GD-18