upsolver

AWS Athena Performance: Best Practices and 6 Performance Tuning Tips

Whitepaper

Executive Summary

In this in-depth technical paper, we will present best practices and tips for maximizing value from Amazon Athena. Amazon Athena was Amazon Web Services' fastest growing service in 2018. The amazing growth of the service is driven by its simple, seamless model for SQL-querying huge datasets.

However, Athena is not without its limitations, and in many scenarios, Athena can run very slowly or explode your budget. We'll help you avoid these issues, and show how to optimize queries and the underlying data on S3 to help Athena meet its performance promise.

In this white paper, you will learn:

- What is AWS Athena?
- Understanding Athena performance
- Performance issues you can face with Athena
- Top 6 performance tuning tips for Amazon Athena
- Using Amazon Athena for streaming data

What is Amazon Athena?

Amazon Athena is an interactive query service, which developers and data analysts use to analyze data stored in Amazon S3. Athena’s serverless architecture lowers operational costs and means users don’t need to scale, provision or manage any servers.

Amazon Athena users can use standard SQL when analyzing data. Athena does not require a server, so there is no need to oversee infrastructure; users only pay for the queries they request. Users just need to point to their data in Amazon S3, define the schema, and begin querying.

However, as with most data analysis tools, certain best practices need to be kept in mind in order to ensure performance at scale. Let’s look at some of the major factors that can have an impact on Athena’s performance, and see how they can apply to your cloud stack.

Understanding Athena Performance

Athena scales automatically, conducting queries at the same time. This provides high performance even when queries are complex, or when there are large data sets. However, Athena relies on the underlying organization of data in S3 and performs full table scans instead of using indexes, which creates performance issues in certain scenarios.

How Does Athena Achieve High Performance?

Massively Parallel Queries

Athena carries out queries simultaneously, so even queries on very large datasets can be obtained within seconds. Due to Athena’s distributed, serverless architecture, it can support large numbers of users and queries, and computing resources like CPU and RAM are seamlessly provisioned.

Metadata-Driven Read Optimization

Modern data storage formats like ORC and Parquet rely on metadata which describes a set of values in a section of the data (sometimes called a stripe). If, for example, the user is interested in values < 5 and the metadata says all the data in this stripe is between 100 and 500, the stripe is not relevant to the query at all, and the query can skip over it.

This is a mechanism used by Athena to quickly scan huge volumes of data. To improve this mechanism, the user should cleverly organize the data (e.g. sorting by value) so that common filters can utilize metadata efficiently.

Treating S3 as Read Only

Athena also optimizes performance by creating external reference tables and treating S3 as a read-only resource. This avoid write operations on S3, to reduce latency and avoid table locking.

Athena Performance Issues

Athena is a distributed query engine, which uses S3 as its underlying storage engine. Unlike full database products, it does not have its own optimized storage layer. Therefore its performance is strongly dependent on how data is organized in S3—if data is sorted to allow efficient metadata based filtering, it will perform fast, and if not, some queries may be very slow.

In addition, Athena has no indexes—it relies on fast full table scans. This means some operations, like joins between big tables, can be very slow.

Athena Product Limitations

According to Athena's service limits, it cannot build custom user-defined functions (UDFs), write back to S3, or schedule and automate jobs. Amazon places some restrictions on queries: for example, users can only submit one query at a time and can only run up to five simultaneous queries for each account.

Athena restricts each account to 100 databases, and databases cannot include over 100 tables. The platform supports a limited number of regions.

6 Top Performance Tuning Tips for Amazon Athena

Let’s take a look at tips to improve query performance, while paying attention to query tuning and storing data optimally in Amazon S3.

1. Partitioning Data

Partitioning breaks up your table based on column values such as country, region, date, etc. Partitions function as virtual columns and can reduce the volume of data scanned by each query, therefore lowering costs and maximizing performance. Users define partitions when they create their table. You must load the partitions into the table before you start querying the data, by:

- Using the ALTER TABLE statement for each partition.
- Using a single MSCK REPAIR TABLE statement to create all partitions. To use this method your object key names must comply with a specific pattern (see documentation).

2. Compress and Split Files

You can speed up your queries dramatically by compressing your data, provided that files are splittable or of an optimal size (optimal S3 file size is between 200MB-1GB).

Smaller data sizes mean less network traffic between Amazon S3 to Athena.

The Athena execution engine can process a file with multiple readers to maximize parallelism. When you have a single unsplittable file, only one reader can read the file, and all other readers are unoccupied.

It is advisable to use Apache Parquet or Apache ORC, which are splittable and compress data by default when working with Athena. If these are not an option, you can use BZip2 or Gzip with optimal file size. LZO and Snappy are not advisable because their compression ratio is low.

3. Optimize File Sizes

Athena can run queries more productively when blocks of data can be read sequentially and when reading data can be parallelized. Check that your file formats are splittable, to assist with parallelism.

However, if files are very small (less than 128MB), the execution engine may spend extra time opening Amazon S3 files, accessing object metadata, listing directories, setting up data transfer, reading file headers, and reading compression dictionaries and more. If your files are too large or not splittable, the query processing halts until one reader has finished reading the complete file, which can limit parallelism.

Using Athena to query small data files will likely ruin your performance and your budget.

Athena uses a data lake approach, which means you're taking in data in its raw form and transforming the data later on.

In a test by Amazon, reading the same amount of data in Athena from one file vs. 5,000 files reduced run time by 72%. We've also covered this topic in our previous article on dealing with small files on S3, so you're welcome to see the benchmarks we arrived at there.

4. Optimize Columnar Data Store Generation

Apache ORC and Apache Parquet are columnar data stores that are splittable. They also offer features that store data by employing different encoding, column-wise compression, compression based on data type, and predicate pushdown. Typically, enhanced compression ratios or skipping blocks of data involves reading fewer bytes from Amazon S3, resulting in enhanced query performance.

You can tune:

- **The stripe size or block size parameter—**the stripe size in ORC or block size in Parquet equals the maximum number of rows that may fit into one block, in relation to size in bytes. The larger the stripe/block size, the more rows you can store in each block. The default ORC stripe size is 64MB, and the Parquet block size is 128 MB. We suggest a larger block size if your tables have several columns, to make sure that each column block is a size that permits effective sequential I/O.

- **Data blocks parameter—**if you have over 10GB of data, start with the default compression algorithm and test other compression algorithms.
- **Number of blocks to be skipped—**optimize by identifying and sorting your data by a commonly filtered column prior to writing your Parquet or ORC files. This ensures the variation between the upper and lower limits within the block is as small as possible within each block. This enhances its ability to be pruned.

5. Optimize SQL Operations

Presto is the engine used by Athena to perform queries. When you understand how Presto functions you can better optimize queries when you run them. You can optimize the operations below:

ORDER BY

- **Performance issue—**Presto sends all the rows of data to one worker and then sorts them. This uses a lot of memory, which can cause the query to fail or take a long time.
- **Best practice—**Use ORDER BY with a LIMIT clause. This will move the sorting and limiting to individual workers, instead of putting the pressure of all the sorting on a single worker.
- **Example—** SELECT * FROM lineitem ORDER BY l_shipdate LIMIT 10000

Joins

- **Performance issue—**When you join two tables, specifically the smaller table on the right side of the join and the larger table on the left side of the join, Presto allocates the table on the right to worker nodes and instructs the table on the left to conduct the join.
- **Best practice—** If the table on the right is smaller, it requires less memory and the query runs faster.

  The exception is when joining several tables together and there is the option of a cross join. Presto will conduct joins from left to right as it still doesn't support join reordering. In this case, you should specify the tables from largest to smallest. Make sure two tables are not specified together as this can cause a cross join.
- **Example—**SELECT count(*) FROM lineitem, orders, customer WHERE lineitem.l_orderkey = orders.o_orderkey AND customer.c_custkey = orders.o_custkey.

GROUP BY

- **Performance issue—**The GROUP BY operator hands out rows based on columns to worker nodes, which keep the GROUP BY values in memory. As rows are being processed, the columns are searched in memory; if GROUP BY columns are alike, values are jointly aggregated.

- **Best practice—**When you use GROUP BY in your query, arrange the columns according to cardinality from highest cardinality to the lowest. You can also use numbers instead of strings within the GROUP BY clause, and limit the number of columns within the SELECT statement.
- **Example—**SELECT state, gender, count(*) FROM census GROUP BY state, gender;

LIKE

- **Performance issue—**Refrain from using the LIKE clause multiple times.
- **Best practice—**It is better to use regular expressions when you are filtering for multiple values on a string column.
- **Example—**SELECT count(*) FROM lineitem WHERE regexp_like(l_comment, 'wake|regular|express|sleep|hello’)

6. Use Approximate Functions

When you explore large datasets, a common use case is to isolate the count of distinct values for a column using COUNT(DISTINCT column). For example, when you are looking at the number of unique users accessing a webpage.

When you do not need an exact number, for example, if you are deciding which webpages to look at more closely, you may use approx_distinct(). This function attempts to minimize the memory usage by counting unique hashes of values rather than entire strings. The downside is that there is a standard error of 2.3%.

```
 SELECT approx_distinct(l_comment) FROM lineitem;
```

Using Amazon Athena to Query Streaming Data

Streaming data is being used in most modern data engineering projects, and a common requirement is to store the data to S3 and query it later. A natural choice for querying streaming data on S3 is Athena.

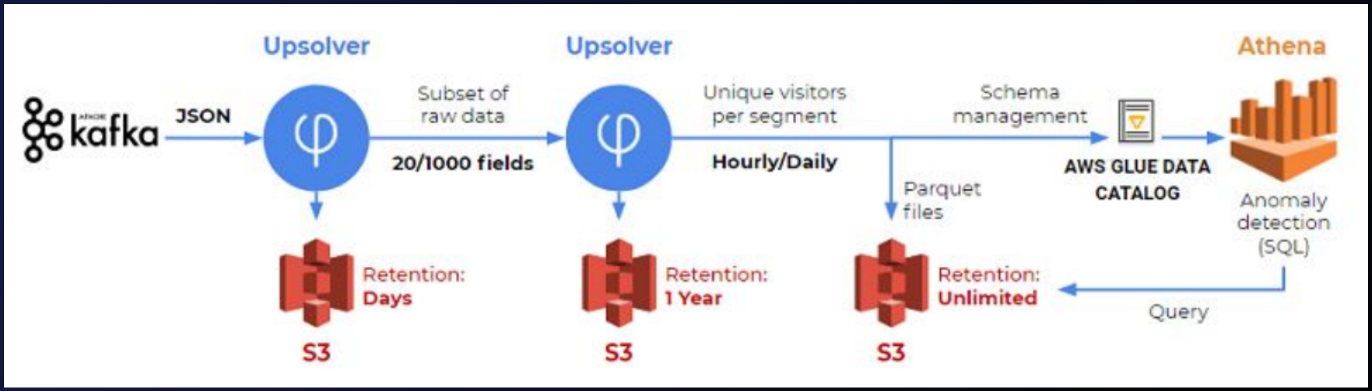
However, as mentioned, Athena can be extremely slow when working on small files, or when querying disorganized datasets that have not been optimized for its specific limitations. In streaming data scenarios, these issues can become extremely challenging to work with, necessitating extensive engineering work to orchestrate multiple open-source frameworks such as Spark/Hadoop, NiFi and MapReduce.

Upsolver provides a streaming data platform that automates and abstracts the optimization and tuning required to run Athena queries on streaming data, enabling any user to drag and drop fields and instantly send structured tables to Athena.

Key features of preparing streaming data for Athena using Upsolver:

- Upsolver's Parquet writer and reader are implemented using streaming to reduce memory footprint and they work 6X times faster than the open source implementation.
- Upsolver writes a file every minute and a file every hour. By combining both, users get fresh data with good performance. This process is called compaction.
- Upsolver automatically manages Athena table partitions using the Glue Data Catalog API. The platform uses time based partitioning using the processing time or event time.
- Big data streams are pre-joined before they reach Athena using a drag & drop interface.
- Automatically flattening nested JSON data into standard Athena tables.

Example Reference Architecture



Source: How SimilarWeb analyze hundreds of terabytes of data every month with Amazon Athena and Upsolver on the AWS blog.

About Upsolver

Upsolver is an easy-to-use service for turning event streams into analytics-ready data with the scale, reliability and cost-effectiveness of cloud storage. Using a visual interface, you can continuously ingest event streams, extract metadata and join with historical big data in real-time. Upsolver's technology adds indexing, automation and high-performance to any cloud storage - in any VPC.

Next Steps:

- Learn more about Upsolver
- Schedule a personalized live demo
- Start your free trial