Introduction to the Presto

* **What is Presto?**

**-Presto (or PrestoDB) is an open source, distributed SQL query engine, designed from the ground up for fast analytic queries against data of any size**. It supports both non-relational sources, such as the Hadoop Distributed File System (HDFS), Amazon S3, Cassandra, MongoDB, and HBase, and relational data sources such as MySQL, PostgreSQL, Amazon Redshift, Microsoft SQL Server, and Teradata.

**-Presto** can query data where it is stored, without needing to move data into a separate analytics system. Query execution runs in parallel over a pure memory-based architecture, with most results returning in seconds.

* **How does Presto work?**

**-Presto is a distributed system that runs on Hadoop, and uses an architecture similar to a classic massively parallel processing (MPP) database management system.** It has one coordinator node working in synch with multiple worker nodes. Users submit their **SQL query** to the coordinator which uses a custom query and execution engine to parse, plan, and schedule a distributed query plan across the worker nodes. It is designed to support standard ANSI SQL semantics, including complex queries, aggregations, joins, left/right outer joins, sub-queries, window functions, distinct counts, and approximate percentiles.

**-**After the query is compiled, **Presto** processes the request into multiple stages across the worker nodes. All processing is in-memory, and pipelined across the network between stages, to avoid any unnecessary I/O overhead. Adding more worker nodes allows for more parallelism, and faster processing.

**-Presto** uses a custom query execution engine with operators designed to support**SQL semantics**. Different from Hive/MapReduce,**Presto** executes queries in memory, pipelined across the network between stages, thus avoiding unnecessary I/O. The pipelined execution model runs multiple stages in parallel and streams data from one stage to the next as it becomes available.

**-**You can launch an **Amazon EMR** cluster running **Presto** in minutes. You don’t need to worry about node provisioning, cluster setup, configuration, or cluster tuning. **Amazon EMR**takes care of these tasks so you can focus on analysis. You can also use tools such as Airpal, a web-based query execution tool open-sourced by Airbnb. Airpal’s user interface simplifies data exploration and ad-hoc analysis and supports features such as syntax highlighting, the ability to export results to CSV, saving queries for later use, and the ability to explore tables to visualize schema.

**-**Run interactive queries that directly access data in **Amazon S3**, save costs using **Amazon EC2 Spot** instance capacity, use **Auto Scaling** to dynamically add and remove capacity, and launch long-running or ephemeral clusters to match your workload. You can also add other Hadoop ecosystem applications on your cluster.

**-Presto** supports the **ANSI SQL** standard, which makes it easy for data analysts and developers to query both structured and unstructured data at scale. Currently,**Presto**supports a wide variety of**SQL** functionality, including complex queries, aggregations, joins, and window functions.

* **Build your Presto implementation in the cloud on Amazon Web Services**

**Amazon EMR** and **Amazon Athena** are the best places to deploy **Presto** in the cloud, because it does the integration, and testing rigor of **Presto** for you, with the scale, simplicity, and cost effectiveness of **AWS**. With **Amazon EMR**, you can launch **Presto** clusters in minutes without needing to do node provisioning, cluster setup,**Presto** configuration, or cluster tuning. **EMR** enables you to provision one, hundreds, or thousands of compute instances in minutes. **Amazon Athena** lets you deploy **Presto** using the **AWS Serverless platform**, with no servers, virtual machines, or clusters to setup, manage, or tune. Simply point to your data at **Amazon S3**, define the schema, and start querying using the built-in query editor, or with your existing Business Intelligence (BI) tools. **Athena** automatically parallelizes your query, and dynamically scales resources for queries to run quickly. You pay only for the queries that you run.

* **How to install Presto?**

1. Go to [Presto](https://prestosql.io/). This is official site for the Presto.
2. Click on **Download** to get the current release.
3. You should look at the Requirements section on the [Deploying Presto](https://prestosql.io/docs/current/installation/deployment.html) to see what is necessary to start Presto.
4. When download is finished, you have to unpack this files.
5. You need to configure the same as in the **Configuring Presto** section of the [Deploying Presto](https://prestosql.io/docs/current/installation/deployment.html) site.

* **Configuration Parameters for Presto**

When a **Presto cluster** runs queries containing hundreds of billions of rows concurrently, it can hit an upper limit for some of the parameters. To avoid this problem, you have to understand how to configure these parameters in the config.properties and jvm.properties files:

* Presto memory
* Query optimization
* Query fault tolerance
* OS-level parameters
* Presto security

**1) Presto Memory Parameters**

**These parameters configure the amount of memory consumed by Presto on AWS:**

* **query.max-memory**: This parameter, contained in the presto.config properties file, is a cluster-level memory limit. It specifies the maximum memory a query can take, aggregated across all nodes. Setting a higher value of query.max-memory avoids a query hitting an upper limit of memory.   
  .
* **query.max-memory-per-node**: This parameter determines the maximum memory a query can use on one node. Ideally, the value of this parameter should be equal to 40 percent of worker instance memory.   
  .
* **Java Virtual Machine (JVM) memory**: The jvm.properties file contains details related to the JVM. Since the memory of the system is being shared between the system, Java, and Presto configuration, it’s important you allocate the right memory to the JVM and spare enough for the OS. The JVM memory should ideally be 80 percent of the total memory of the instance.   
  .
* **Presto Spill to Disk**: Presto, by default, is an in-memory query engine, which stores intermediate operation results only in memory. However, this does not work well with memory-intensive queries. The mechanism is like OS-level page swapping. Therefore, it’s important to enable the Spill to Disk for Presto to use disk memory to store the temporary data.   
  .
* **Presto Memory Architecture**: Presto divides memory in each node across two pools: general and reserved. Each memory pool has a certain role to play in the execution of the query. The reserved pool is used only when a worker node exhausts all its memory in the general pool.   
     
  The reserved pool fast tracks one particular query and guarantee its completion. However, due to this behavior, the execution of all other queries suffers, since the other queries on that particular node are stalled until that one query completes.

**2) Query Optimization Parameters**

**These parameters configure the behavior of queries:**

* **Cost-Based Optimization (CBO)**: The CBO makes decisions based on several factors, including shape of the query, filters, and table statistics. Enable the CBO parameter in Presto to optimize the query structure before running.   
  .
* **Session properties**: Session properties provide the flexibility to control the behavior for queries based on their properties. Session properties control resource usage, enable or disable features, and change query characteristics. When running heavy queries, one of the important session properties that will help is query\_max\_execution\_time.   
  .
* **Queueing mechanism**: Queries being submitted to the Presto cluster can be pushed to different queues, based on the content of the query or the client tags of the query. Each queue can then be configured to different parameters, to control resource usage or use different features.

**3) Related Parameters**

**These parameters configure fault tolerance, number of open files, and LDAP authentication:**

* **Presto retry mechanism**: When a node fails, by default Presto will terminate the queries that were using that particular node for querying data. This makes it difficult to scale up and down Presto with AWS Auto Scaling features. Configure the Presto retry mechanism to avoid this problem.   
     
  One trade-off Presto makes to achieve lower latency for SQL queries is to not care about mid-query fault tolerance. If any of the Presto worker nodes experience a failure, such as getting terminated, queries in progress will abort and need a restart.   
     
  Due to the absence of the retry mechanism, it’s difficult to achieve auto scaling of the cluster, as auto scaling in Amazon EMR (when scaling down) terminates random instances based on aggregate memory of the cluster.
* **Ulimits**: The number of open files and running processes in a Linux system is limited by default. We need to increase this limit in the ulimits.conf of the operating system.
* **Presto LDAP**: LDAP Authentication on Presto requires HTTPS to be enabled and a keystore to be created. The keystore then needs to be passed on to the client to be added to the property.

* **Solution Architecture**

To set up the new stack on **AWS**, we migrated the data to the storage location on **Amazon S3**. To set up **Amazon EMR**, we considered the networking configuration, security group policies, and authentication per Seagate security compliance needs.

We adjusted some**Hadoop** and **Hive**parameters to obtain optimum utilization of the **Amazon EMR cluster and S3**. The following architecture shows Seagate’s setup after migration.

* **Conclusion**

The migration to **Amazon EMR Presto SQL** allowed Seagate Technologies to continue processing petabytes of data fast, at a reduced cost of **Amazon EMR**, and resulted in a 10x improvement of the execution time of queries.

Since we deployed the solution with network load balancing, Seagate could switch clusters without impacting their end user. Seagate saved a lot of CPU time, and their end users could get query results in minutes rather than hours. This improved the overall efficiency of both their **Amazon EMR** cluster and business operations.

