MATILLION

Optimizing Amazon Redshift

A Real-World Guide

amazon web services | Partner Network

ADVANCED TECHNOLOGY PARTNER

BIG DATA COMPETENCY

Table of Contents

Introduction

Just as Amazon Web Services (AWS) has transformed IT infrastructure to something that can be delivered on demand, scalably, quickly, and cost-effectively, Amazon Redshift is doing the same for data warehousing and big data analytics. Redshift offers a massively parallel columnar data store that can be spun up in just a few minutes to deal with billions of rows of data at a cost of just a few cents an hour. It's designed for speed and ease of use — but to realize all of its potential benefits, organizations still have to configure Redshift for the demands of their particular applications.

Whether you've been using Redshift for a while, have just implemented it, or are still evaluating it as one of many cloud-based data warehouse and business analytics technology options, your organization needs to understand how to configure it to ensure it delivers the right balance of performance, cost, and scalability for your particular usage scenarios.

Since starting to work with this technology in September 2013 , Matillion has used Amazon Redshift to deliver dozens of data warehousing and analytics projects. This e-book collects the results: our practical advice, real-world examples, and best practices for optimizing Redshift configuration and performance. It also includes anevaluation of Amazon's own recommended best practices and how best to apply them.

Redshift: Under the Hood

COLUMNAR DATABASE ENGINE FUNDAMENTALS

The overall instance or configuration of Redshift is a cluster. You decide at the cluster level whether your Redshift implementation will use SSD or magnetic storage. A cluster can be resized at any time as long as it remains larger than the amount of storage you need however resizes need to be planned carefully as the database will go into a “read only” mode during the resize operation.

Each Redshift cluster is made up of one leader node, which serves as the front end for your application and which your application treats as the database. From the outside, the leader node looks like a Postgres database, from which the core technology of Redshift is descended.

The actual work is done by additional nodes behind the leader node, each one a virtual machine. Optimum performance depends on following best practices for schema design to ensure the right data is distributed to the right nodes. Otherwise, adding nodes to increase power can have minimal or even negative impact on performance and scalability while increasing the cost per cluster.

Each node has multiple slices, each one roughly analogous to a processor or core allocated to working on that node’s data. The cluster allocates work to the node, which further splits the work across its available slices as efficiently as possible.

HOW REDSHIFT STORES DATA

Because you can't know in advance what data fields your users want to filter, group by, and use, columnar data stores create a data structure similar to an index for every column of data, thus eliminating the main data store. In effect, they are only an index for each column. All values of the same type and those with similar values are organized next to each other in the indices, which makes compression far more efficient than in traditional relational database management systems.

In most databases, column encoding refers to the character set used in the column (e.g. Latin-1 or Unicode). In Redshift, all character data is UTF-8 encoded Unicode. In Redshift, "column encodings" refer instead to the 12 different compression strategies you can use to define each column of a table. You can allow Redshift to choose one for you, or you can select one manually based on your storage and performance preferences.

Best Practices

This section introduces our assessment of Amazon's best practice documentation for Redshift. Our real-world experience suggests some of Amazon's recommendations are critical, while others are optional or even unnecessary.

SCHEMA DESIGN

This is the single most important factor in making the most of Redshift. Find relevant AWS documentation here.

Amazon recommends: Use the smallest possible column size required to fit your data.
Our advice: After Compression this will have minimal effect on the stored data size. It could impact query performance in situations where data volumes contain hundreds of millions of rows. Consider implementing it if you get out-of-memory errors when running queries.

Amazon recommends: Let "Copy" choose compression encodings.
Our advice: The "Copy" command is the method we suggest for getting data into Redshift. We agree with Amazon's advice to let "Copy" automatically choose the best column encoding for uploading data. However, "Copy" only chooses a column encoding scheme for the first upload of data into an empty table. The first time you use the command to populate an empty table, use a significant and representative data set so Redshift can evaluate it properly and optimize the compression for your workload.

Amazon recommends: Choose the best distribution style and key.
Our advice: This refers to the way data is distributed across the nodes: All, Even, or Key. All copies the table to every node in the cluster, which improves query performance but increases both upload time and storage use. Even spreads table rows evenly across nodes so each slice of each node works in parallel to respond to queries, which works best for queries with no joins. Key distribution sets allow you to specify a column to distribute on, then Redshift places all rows with that value on the same node. Your choice has a significant impact on query performance for queries with joins. They also affect data storage requirements, required cluster size, and the time required to upload data into Redshift. We typically choose All for smaller dimension tables and Even for tables that are not joined to other tables or are only joined to tables with All specified. For very large dimensions, we distribute both the dimension and any fact associated with it on the join column, then choose All for a second large dimension if necessary. If your query is running slowly, optimize here.

Amazon recommends: Choose the best sort key.
If you specify a sort-key column, Redshift maintains the data in a table in that order. This is sorted within each partition, not overall. Choosing a sort key is especially useful for optimizing data filtering and for joining two tables when one or both are sorted on their join keys. We advise optimizing for joins first, then for filtering.

Interleaved sort keys attempt to optimise multiple columns for fast-filtering, without the order of the keys mattering. For the original, compound, sort keys the order is important - queries filtering on non-leading parts of the key cannot be optimised. Compound is still a good choice too for optimising joins.

Amazon recommends: Define primary key and foreign key constraints.
This enables you to tell Redshift about referential relationships in your data, just as in a conventional database. However, the relationships are not enforced, which allows for both duplicate primary key values and foreign key values that do not match primary key values. We do not currently feel this recommendation provides a significant boost to query response times, but if your experience proves otherwise, we're interested to hear the details.

Amazon recommends: Use date/time data types for date columns.
This is basic to using Redshift's rich set of date functions.

LOADING DATA

Best practices in this area focus on getting data into a Redshift cluster.
Find relevant AWS documentation here.

Amazon recommends: Use a Copy command to load data.
This allows you to move data rapidly into a table from various input sources including S3 and SSH. The command can also perform automatic column encoding at the same time. This is the best way to load large amounts of data.

Amazon recommends: Use a single copy command to load from multiple files.
The Copy command can name multiple files at once and use them as if they were a single large file. Especially when loading from S3, split the data into segments equivalent or greater to the number of slices in the cluster. By giving each slice part of the work, the load will happen in parallel and at much higher speeds.

Amazon recommends: Compress data files with gzip or lzop.
If you can compress data before or while loading, you should. What you lose in CPU time, you regain in reduced bandwidth needs and faster data transfer time as you move gigabytes of data over the network.

Amazon recommends: Use a multi-row insert.
Inserting data using SQL statements is orders of magnitude slower than using the Copy command. We recommend avoiding both multi-row and single inserts in favor of preparing an input file on S3 or creating an SSH command and bulk-loading.

Amazon recommends: Use a bulk insert.
A bulk insert creates large tables on the cluster when the input data is already in the same Redshift database. It works quickly and leverages Redshift functions to transform data already in the database into a different format.

Amazon recommends: Load data in sort key order.
This is simply a way to eliminate the need to sort data as it's loaded. We advise it only if you can easily generate data in your preferred order — for example, log data using transaction date as a sort key. Otherwise, let Redshift sort it for you.

Amazon recommends: Use time-series tables for log-style or transactional data.
This involves splitting data from one huge table into multiple smaller ones to simplify data management — for example, splitting by date and removing older data by dropping the relevant tables. It's a sensible way to manage data.

QUERYING DATA

Best practices in this area involve accessing loaded data in an aggregated, useful form. Find relevant AWS documentation here.

Amazon recommends: Design for performance.
Once your schema is filled with data, changing it will be time-consuming. Following best practices as you create a schema saves you from having to compose complicated queries later to get the answers you need.

Amazon recommends: Vacuum.
If you maintain data in existing tables, you will need to perform regular vacuums to reclaim disk space from deletes/updates and re-sort the data according to the sort key. Vacuum is CPU-intensive and only one vacuum operation can run across the entire cluster at any given time, so scheduling is difficult. We advise avoiding them if possible, and if not, implementing a system where vacuum requests go into a queue to be carried out in a batch run overnight. Since mid 2016 vaccumm perfomance in Redshift is much improved so this tends to be less of a concern. All of the above comments remain relevant

Amazon recommends: Configure WLM.
WLM (Work Load Management) assigns incoming queries to different queues carved out of your cluster's available resources. More slots can execute more concurrent queries, but each will have fewer resources. You can use this to prioritize queries from higher-priority workloads. processes that load the data warehouse such as ELT processes would normally be in a lower priority WLM queue and a users analytic queries.

Amazon recommends: Maintain up-to-date statistics.
Running Analyze frequently creates summary statistics by scanning a sample of rows in each table. We recommend doing it automatically on load, during the Copy process, so your database has the necessary statistics to determine how to run queries most efficiently.

Conclusion

Implementing the best practices described in this e-book will ensure a speedy Amazon Redshift implementation and help deliver the data warehouse and big data analytics performance necessary to answer your organization’s most pressing business questions.