International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 07 Issue: 01 | Jan 2020 www.irjet.net p-ISSN: 2395-0072

Amazon Redshift Workload Management and Fast Retrieval of Data

Palash Chaudhari[1]

[1]*Employee, Cognizant Technology Solution, Pune, Maharashtra, India*

Abstract - *Amazon Redshift is a database service that is fully managed, fast, reliable and a part of Amazon's cloud computing platform, Amazon Web Services (AWS). Redshift has many striking features like Parallel Processing, Fault Tolerance, Integration with third party tools and Limitless Concurrency which is not possible to get in any traditional databases. It has Massively Parallel Processing (MPP) architecture, which distributing SQL operations and parallelizing techniques to make use of all available resource, this process is known as Work Load Management (WLM). WLM is responsible for faster performance of this database. As similar to WLM it is important to focus on retrieval of data through Redshift database. While querying a table that has huge amount of data, the time taken to retrieve the data will be more, however fast retrieving techniques will help to reduce the retrieval time or loading time*.

Key Words: Redshift, Workload Management, Vacuum, ETL, Query, Deep Copy.

1. INTRODUCTION

Amazon Redshift uses efficient techniques for Workload Management (WLM) as well as to obtain a very high level of query performance on large amounts of datasets, ranging from hundred gigabytes to a petabyte, this feature puts this database in different league from rest of massive parallel processing databases. This enables setting up query groups and their priorities and customizes the parameters for query execution for that group. WLM configurations are managed through Amazon Redshift management console or command line interface (CLI) or the Amazon Redshift API.

2. Need of Workload Management

If there is one organization's database and two employees want to extract some information. First employee want 'Name, Employee ID, Address, Number and Salary' of all employees in organization and Second employee just want 'Name' of all employees. When they execute the query, it comes under queue. As first query is long running, so second person need to wait until successful execution of first query and hence it leads to waste of time. Workload Management provide massive role in Redshift database, which helps users to flexibly manage the workload so that fast running queries or short queries will not get stuck in the queues behind the long running queries. It helps to put users and queries into priority buckets or groups and assign customized system resources for their fulfillment through appropriate queues.

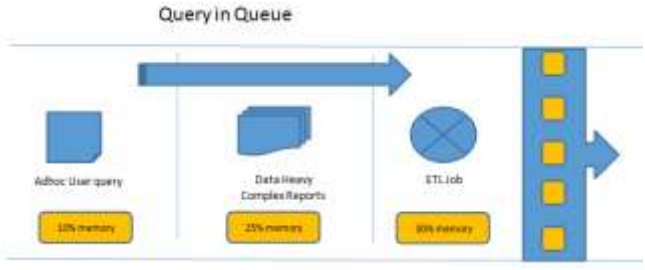

Fig -1: Query Queue

3. WLM Configuration Parameters

a) Queue
Queues are organized buckets where waiting queries are lined up. Redshift have capability of creation of 8 queues. Workload management allows use of wildcards to match pattern or detect presence of certain characters in query labels to tag those to appropriate queue.

b) User Group / Query Group
User Groups or Query Groups are labels that can be assigned to queries to determine the queue through which they can be processed. These labels are assigned by user to the query at run time. WLM assigns the query to appropriate query queue. By default there are two groups Super Users and Regular Users. Super Users have high priority and Regular Users have low priority.

c) Concurrency
It means how many queries can be processed at same instance of time. By default, the concurrency level is 5 for each group. Redshift have maximum concurrency level can be 50. The Concurrency is inversely proportional to Memory. It means higher the concurrency, lesser the memory available to each query slot. There could be specific memory intensive queries which could need more number of slots to process. This can be achieved by tuning the wlm_query_slot_count parameter. WLM creates as many query slots as the concurrency level of the queue.

d) Timeout
When query takes long time to process which it will be automatically cancel or hopped.

e) Hopping
Moving the query to other queue where it can be processed earlier than current one is called hopping.

f) Short-Query Acceleration (SQA)
Short queries are the ones which need very minimal memory and less time to process. The SQA feature is enabled by default. This provides automatically, the short queries in user defined queues to get prioritized ahead of long queries by WLM. Thus a separate queue and group is not required to be created when this feature is turned on.

4. Dynamic Memory Allocation

In every queue, numbers of query slots are created by WLM which is equal to queue's concurrency level. The memory allocated to query slot is equal to the queue divided by the slot count. Amazon Redshift dynamically shifts to a new WLM configuration if memory allocation or concurrency gets change. Hence, the running queries can able to execute successfully by using currently allocated memory. While same time database make sure that the total amount of usage memory will not exceeds hundred percent of available memory.

The workload manager uses following method for transition.

a) For the new query slot WLM frequently calculate the memory allocation.
b) If the query is not running in the query slot, WLM immediately remove that slot and make the memory available.
c) In the query slot if query is running, WLM waits for successfully execution of query. Then these inactive slots are removed and make the memory available.
d) New slots are added if large amount of memory is available.
e) The slot count equals the new concurrency level, after finishing of all active queries at the time of change and the transition process to the new WLM configuration is complete.

5. WLM Queue Assignment Rules

WLM assigns the query submitted by a user as per a set of rules. Based on the user it determines which queue the query should be added to or if a query group is labeled. If there is no specific group or label then the query is added to default group. Below flow chart helps to understand the WLM queue assignment rule.

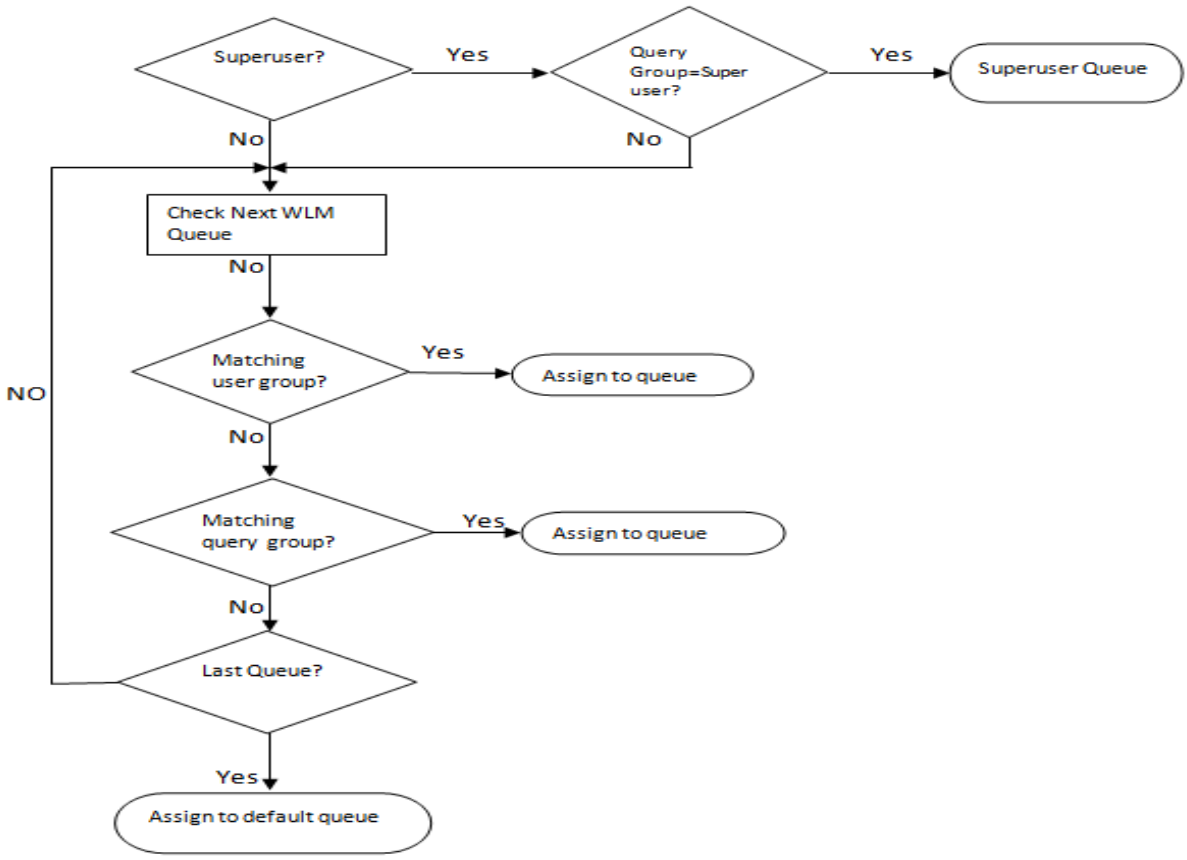

Fig -2: Flowchart of WLM Queue Assignment Rule

6. Where WLM Helps

Redshift itself being a Data Warehouse service, it's implied this data would be read significantly more number of times than updated. Concurrent query access could be a performance impact, where there could be multiple consumers of same information. The consumers could be data analyst, reporting applications, power users or downstream API consumers. They could be requesting a lot of information in one go or may be small subsets but more frequently.

A typical use case could be a sales data warehouse, which is refreshed with new data continuously. Daily and weekly sales data aggregated and sliced and dices across various dimensions like divisions, products, geographies could be consumed by various sections of users like ETL (load/unload), APIs, power users, data analysts, canned reports etc. Here we could leverage the WLM capabilities so that we have separate query queues for ETL, ELT and rest of users. Power users could be making shorter queries on pre-aggregated datasets and expect very quick turnaround times. Data analysts could be applying complex logic and process large queries which are more memory intensive but not time critical and thus can wait. Canned reports could be stitching data from across data marts to deliver information and could be both memory intensive and time critical. Downstream API consumers could be third party and could have varying priorities. If the data hits were uneven, for example very high towards financial period ends and normal otherwise, concurrency scaling would be something that can come handy. By labeling the queries, forming queues and assigning appropriate parameters, we can make Redshift data consumption scalable and yet seamless.

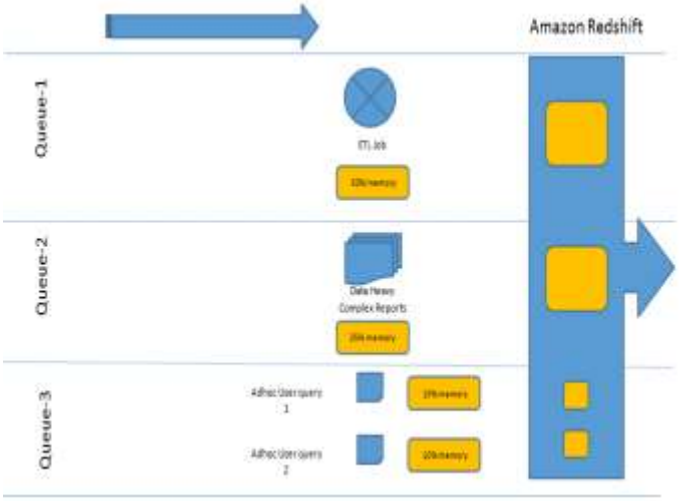

Fig -3: Query Execution using WLM feature

Table 1: Memory Allocation to Resource

Queue	Conc.	User Group	Memory	Memory per Slot
ELT Job	1	ELT	40%	40%
Reports	1	Reports	40%	40%
Adhoc	2	Power User	20%	10%

In above illustration, we can see that due to multiple queues and their allocated resources, respective query groups become independent of the other. By applying monitoring rules, when one query exceeds the threshold of its allocated resources like memory or time, appropriate action can be taken. There would be no underutilization of available slots if weighted resources are available. Thus, WLM can help in optimum query performance for Redshift.

7. Fast Retrieval of Data

There are 4 aspects of fast retrieving of data.

a) Specify column name in SELECT query

While fetching data from the table, specify the column names instead of putting *. Since the Redshift uses columnar storage, data will be retrieved fast. Instead of using 'SELECT * FROM TABLE_NAME', use 'SELECT COL_1, COL_2, COL_3 FROM TABLE_NAME'. In columnar storage, each block stores a data of single column for multiple rows. When fetching the data, it drastically reduces disk I/O requirements which intern increases the performance. Also, each block stores a data of same data type. Hence we can apply compression techniques that further reduce the disk space and I/O. Typical database block size ranges from 2KB to 32KB. Amazon Redshift uses the block size 1 MB which is more efficient and further reduces the number of I/O requests that needed to be performed on database loading or part of query execution.

b) Vacuuming a Table

When we delete a data from the table, the space associated with the data will not be deleted. Due to this, the query performance may get lower. In extreme situations, we might even end up with queries which may time-out due to the extra overhead the rows are deleted but not reclaimed space. Amazon Redshift does not reclaim free space automatically. So, we need to use some techniques to reclaim the space. There comes a command called as vacuum. The Vacuuming process, is quite important for the health and maintenance of AWS Redshift cluster. By running a Vacuum command on one of our tables, we reclaim any free space that is the result of delete and update operations. At the same time, the data of the table get sorted. This way, we end up with a compact and sorted table, which are useful for the performance of our cluster.

Syntax:-
VACUUM [FULL | SORT ONLY | DELETE ONLY | REINDEX] [[table_name] [TO threshold PERCENT]];

Vacuuming process should happen during periods of inactivity or at least minimal activity on your cluster. So query planning is again essential here. The longer the time between consecutive vacuum commands for a table, the longer it takes for the vacuuming process to end. As vacuuming is about going through your data and reclaiming rows marked as deleted, it is an I/O intensive process. So, it affects any other queries or processes that you might be running concurrently, but the good thing is, Vacuuming can happen concurrently with other processes, so it may not block any ETL processes or queries you might be running.

Types of Vacuum

- **VACUUM FULL:** Vacuum Full is the default configuration of a vacuum command, so if we do not provide any parameters to the command, this is performed on your data. With a Full Vacuum type, we both reclaim space, and we also sort the remaining data. These steps happen one after the other, so Amazon Redshift first recovers the space and then sorts the remaining data.
- **VACUUM DELETE ONLY:** If we select this option, then we only reclaim space and the remaining data in not sorted.
- **VACUUM SORT ONLY:** With this option, we do not reclaim any space, but we try to sort the existing data.
- **VACUUM REINDEX:** This command is probably the most resource intensive of all the table vacuuming options on Amazon Redshift. It is a full vacuum type together with re-indexing of interleaved data. It makes sense only for tables that use interleaved sort keys.

c) Deep Copy

Deep copy recreates and repopulates table by using a bulk insert, which automatically sorts the data. If a table has large unsorted data, then deep copy is much faster than vacuum. Below are the methods to create a copy of original table for deep copy activity.

Use the original table DDL:

Use CREATE TABLE DDL available for the table and create a copy.

Steps:-

- Create a copy of the table using the original CREATE TABLE DDL.
- Use an INSERT INTO … SELECT statement to populate the copy with data from the original table.
- Drop the original table.
- Use an ALTER TABLE statement to rename the copy to the original table name.

Use CREATE TABLE LIKE:

If the original DDL is not available, we can use CREATE TABLE LIKE to recreate the original table. The new table inherits the encoding, distkey, sortkey and notnull attributes of the parent table. The new table doesn't inherit the primary key and foreign key attributes of the parent table, but we can add them using ALTER TABLE.

Steps:-

- Create a new table using CREATE TABLE LIKE.
- Use an INSERT INTO … SELECT statement to copy the rows from the current table to the new table.
- Drop the current table.
- Use an ALTER TABLE statement to rename the new table to the original table name.

Create a temporary table and truncate the original table:

If we need to retain the primary key and foreign key attributes of the parent table, or if the parent table has dependencies, you can use CREATE TABLE ... AS to create a temporary table, then truncate the original table and populate it from the temporary table. Using a temporary table improves performance significantly compared to using a permanent table, but there is a risk of losing data. A temporary table is automatically dropped at the end of the session in which it is created. TRUNCATE commits immediately, even if it is inside a transaction block. If the TRUNCATE succeeds but the session terminates before the subsequent INSERT completes, the data is lost. If data loss is unacceptable, use a permanent table.

Steps:-

- Use CREATE TABLE AS to create a temporary table with the rows from the original table.
- Truncate the current table.
- Use an INSERT INTO… SELECT statement to copy the rows from the temporary table to the original table.
- Drop the temporary table.

d) Analyze

Analyze is used to update stats of a table. When a query is issued on Redshift, it breaks it into small steps, which includes the scanning of data blocks. To minimize the amount of data scanned, Redshift relies on stats provided by tables. Stats are outdated when new data is inserted in tables. These stats information needs to be updated for better performance of queries on Redshift.

Syntax:-
ANALYZE [VERBOSE] [[table_name [(column_name[,...])]] [PREDICATE COLUMNS | ALL COLUMNS];

REFERENCES

[1] https://aws.amazon.com/redshift/
[2] https://docs.aws.amazon.com/redshift/latest/mgmt/managing-parameter-groups-console.html
[3] https://docs.aws.amazon.com/redshift/latest/dg/cm-c-wlm-dynamic-example.html
[4] https://docs.aws.amazon.com/redshift/latest/dg/cm-c-wlm-queue-assignment-rules.html
[5] https://docs.aws.amazon.com/redshift/latest/dg/tutorial-wlm-understanding-default-processing.html
[6] https://docs.aws.amazon.com/redshift/latest/mgmt/workload-mgmt-config.html