Integrated querying of S3 and SQL database data using Amazon Redshift

Romit Kankaria

Disaggregated Storage Systems

1. **Why?**

a. Support important features such as elasticity, independent scaling, cost efficiency, and fast recovery
b. resource utilization at the cloud level has led to wide adoption

2. **Storage disaggregation majorly divided into:**

a. Software-level disaggregation
b. Shared-storage

3. **Types?**

a. OLTP - Aurora
b. OLAP - Snowflake and Amazon Redshift

Some recent updates on Redshift…

1. Amazon's fully managed OLAP database service optimized for storage disaggregation

1. Initially, shared nothing multiprocessing style, unlike Snowflake

1. Redshift Managed Storage added to support independent scaling

1. Redshift scales compute nodes via multi-cluster autoscaling (called Concurrency Scaling).

1. It also introduces many optimizations, e.g., compression, query compilation, offloading, and FPGA accelerations

https://www.cs.purdue.edu/homes/csjgwang/pubs/SIGMOD23_Tutorial_DisaggregatedDB.pdf

How Redshift helps in scalable disaggregation?

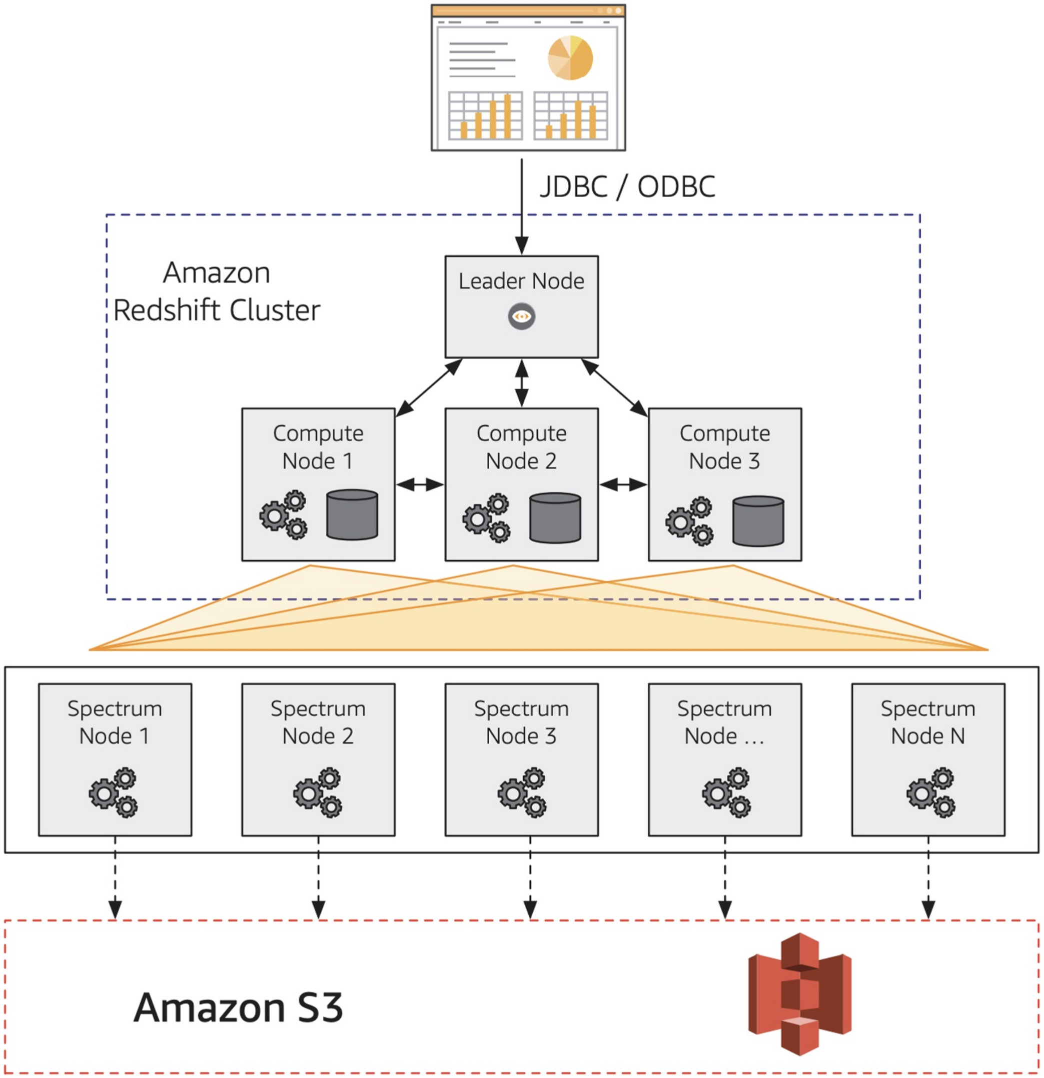
Motivation and Idea for Redshift

1. Redshift is more of a mediator for serving scalable object storage systems like S3

1. Provides integrated and in-place access to data sources like SQL and S3 can be accessed

1. Redshift acts as integration query processor

1. Issues pertaining to Query planning and query processing are the focus here



1. Amazon Redshift and the Spectrum processing layer

Query Processing by Redshift

1. Adds a multi-tenant (sub)query execution layer, called Amazon Redshift Spectrum (more on this later)

1. Provides a logical view of the S3 data as external tables in addition to providing access to the Redshift tables

1. SQL syntax for querying tables stays the same, irrespective of data source

Amazon Redshift Spectrum

1. Reads and processes records efficiently before streaming very small amount of data back to the Redshift compute nodes, providing massive parallelism

1. Works majorly for queries that involve: filter, project and aggregate

1. Joins, order-by and final aggregations handled by Redshift compute nodes

Challenges

1. Nodes are stateless

1. Memory capacity issues, does not use a local disk

1. SQL functionality not the same as Redshift compute nodes

Case Study

1. Primary use case of Redshift is querying a very large fact data residing in S3

Some context

1. The case study here is the marketing campaign for the 8th Harry Potter book

1. Need to find places where and how much the past billboard marketing campaign succeeded in Miami

1. Thus, they want to find the regions where books sales were boosted by the billboard campaigns

So, what are the initial steps?

1. Create a temporary table hp_book_data that holds:
   the raw aggregated data about each Harry Potter book sales per Miami zip code, for the sales that followed within 7 days of its release.

1. Computation of hp_book_data is quite difficult

```
SELECT
   SUM(D.QUANTITY * D.OUR_PRICE) AS SALES,
   P.TITLE, R.POSTAL_CODE, P.RELEASE_DATE
FROM
  S3.D_CUSTOMER_ORDER_ITEM_DETAILS D,
  ASIN_ATTRIBUTES A, PRODUCTS P, REGIONS R

WHERE
    D.ASIN = P.ASIN AND P.ASIN = A.ASIN AND
    D.REGION_ID = R.REGION_ID AND
    A.EDITION like '%FIRST%' AND P.TITLE like '%Potter%' AND
    P.AUTHOR = 'JK Rowling' AND
    D.ORDER_DAY >= P.RELEASE_DATE AND
    D.ORDER_DAY < P.RELEASE_DATE + 7 Days AND
    R.COUNTRY_CODE='US' AND R.STATE = 'WA' AND
    R.CITY = 'Miami'
GROUP BY
   P.TITLE, R.POSTAL_CODE, P.RELEASE_DATE
```

Next Steps of Query Planning

1. The second step of the analysis is to compare the book-over-book improvements per zip code and join with knowledge of which release/zip code combinations had billboard campaigns.

1. Helps determine which billboards actually helped the sale of books

1. hp_book_data has small amount of data (rings a bell?)

1. Data present in both S3 as: S3.D_CUSTOMER_ORDER_ITEM_DETAILS

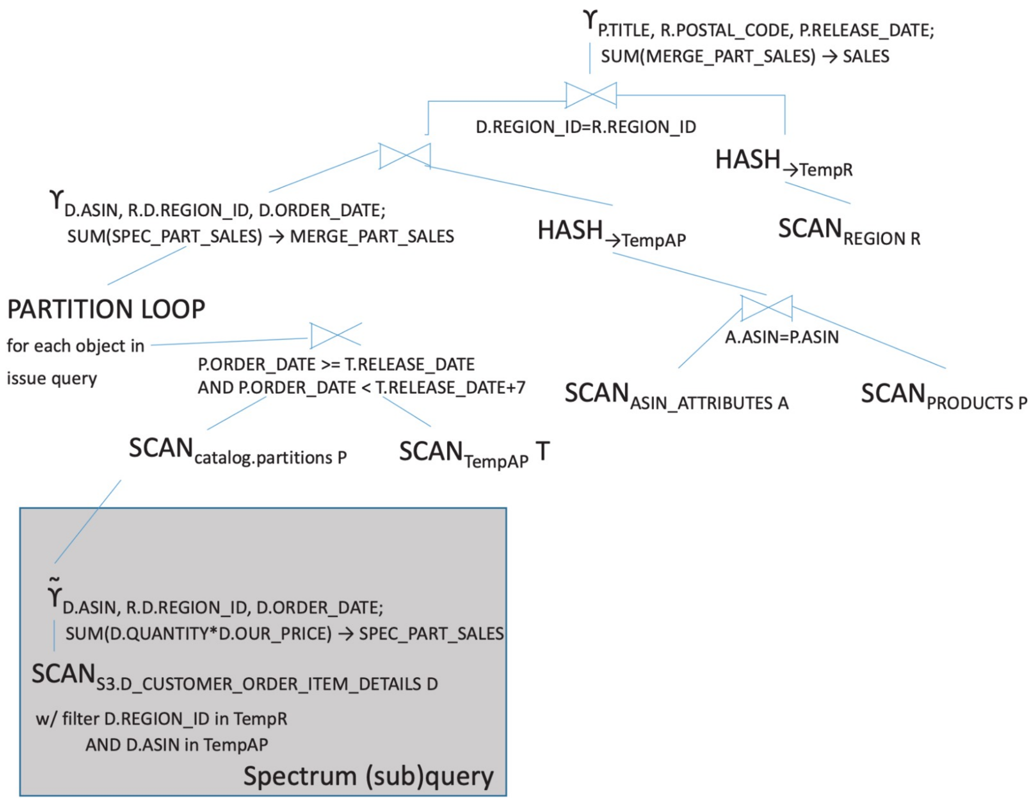
1. Dimension tables for the same present in Redshift

1. Great use case, where Redshift makes use of external fact tables with dimensions for the same present in Redshift itself

Next Steps of Query Planning

1. Redshift has the PRODUCTS, REGIONS and ASIN_ATTRIBUTES tables

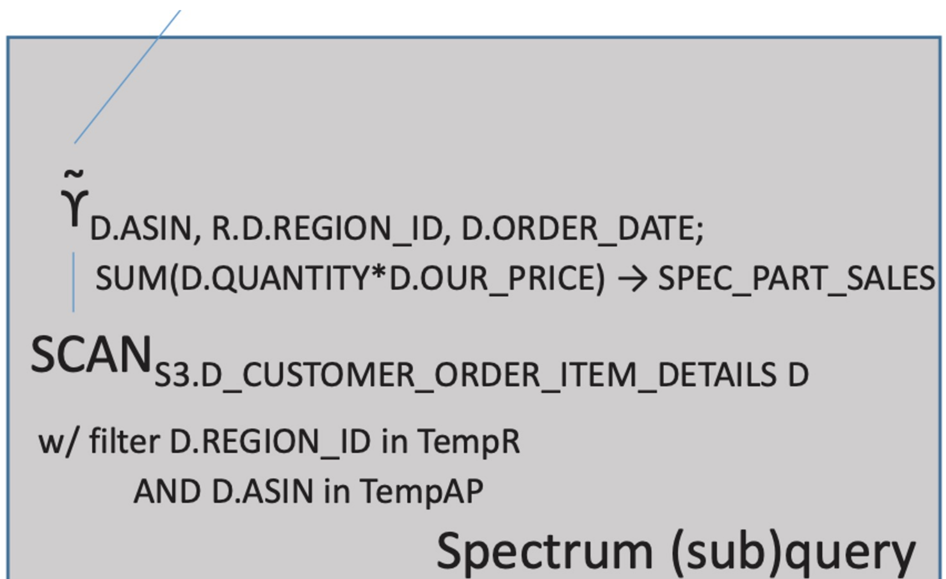
1. So here is the final query plan:



Efficiency of the Query plan

1. Pruning is done at partition level, which are identified by ORDER_DAY as all we want is sales with 7-days-post-release condition

1. Spectrum layer comes into the picture:
    a. large amount of data per object boils down to returning only a few tuples, thanks to the IN filters and the presence of the aggregation



Redshift Dynamic Distributed Query Optimization

1. **Join Ordering**
   a. Creates a query plan in a cost-based fashion to yield smallest intermediate results and data that is to be exchanged
   a. S3 tables are at the left-most side due to their size
   a. Many SQL-on-Hadoop engines do not offer a join ordering optimization

2. **Aggregation PushDown**
   a. Queries of this nature sent to Spectrum layer and enhances scalability and performance by multifold
   a. Drastically reduces the amount of data returned back to the compute nodes
   a. There are two kinds of aggregation: merge and pre aggregation
   a. There is another final aggregation handled by the compute nodes

Redshift Dynamic Distributed Query Optimization

3. Partial Aggregation

a. Aggregation might cause memory issues

a. Pre-aggregation transformed to partial aggregation, outputs more than one tuple with the same grouping value

a. Allows to deal with aggregations with results larger than available memory

4. Semijoin Reduction by Dynamic Optimization

a. Should focus only on what matters, for example a few harry potter books and a few Miami regions - Spectrum helps with that

a. Post REGION_IDs and ASINs are known, the IN filter carries out the semijoin reduction

a. Decision for semijoin done during runtime, because pre-planning might cause memory issues

Redshift Dynamic Distributed Query Optimization

5. Smart Partitioning, driven by Joins

a. Partition loop operator responsible for finding relevant objects to the task at hand

a. Interesting case, when constraint has a JOIN involved

a. Thus, only the partitions that are associated with these ORDER DAYs should be queried, which in turn can be found out using the JOIN on PRODUCTS and ASIN_ATTRIBUTES

a. the semijoin's condition is derived as follows: Detect the join conditions on the partitioned S3 table that involve the partition attribute

To Conclude

1. Amazon Redshift provides integrated access to relational tables and S3 objects.

1. data is accessed via a highly parallel, multi-tenant processing layer, called Amazon Redshift Spectrum.

1. Multiple optimizations ensure that the queries executed at the scalable Spectrum layer process only the relevant S3 objects

1. return to the compute node cluster small results, while cost-based optimizations, such as join ordering, are still in effect.

ANY QUESTIONS?
THANK YOU

WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON