dremio

Dremio Software

Query Performance Analysis and Improvement

Introduction

This document aims to help identify the reason for poor query performance and suggest ways to improve it. The document assumes the reader is familiar with Dremio, either as an administrator or a user, can run queries and navigate around Dremio's UI (specifically the "Jobs" page).

As with any data platform, the amount of memory and the number of processors has the highest impact on the response time of a query. Those resources together make up the Dremio Cluster. Note: data storage is not part of the Dremio Cluster (the main difference between a Data Warehouse and a Data Lakehouse). Of course, IO performance is also critical to any analytics platform.

Additionally, query performance is directly related to the complexity of the query (aggregations, joins, etc), the concurrent number of users running queries on a Dremio cluster and how the data is stored versus how it's scanned.

We use Dremio's Query Profile report (referred to in the Docs as "Raw Profile") to analyze the query, identify where most of the time is spent, potential bottlenecks, and discuss ways to improve the query.

Gather Dremio Cluster Information

It will be beneficial to have the following information handy. You can refer to these details while analyzing the query in the *Profile* report.

Cluster Size	No. of Nodes	No. of Cores	Memory (GB)
Coordination nodes			
Executor nodes			

Workload Details	Details
File format(s)	
File size	
Number of concurrent queries	

Dremio Cluster Components

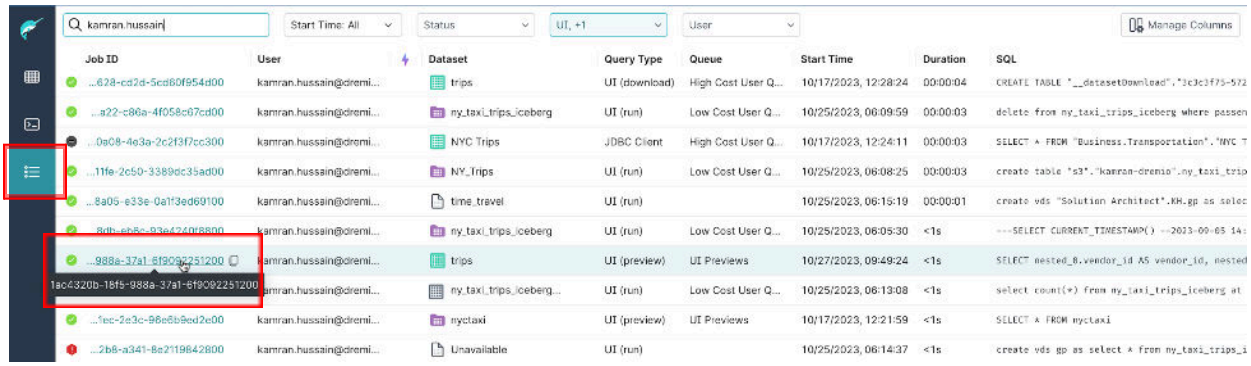
Here are the roles for each node type:

- Coordinator(s) - responsible for handling connections, serving the UI and query planning. There can be more than one coordinator node (scale-out coordinator).
- Executors - running the queries, joining, sorting and combining result sets, metadata and reflection refresh

Query Profile

A query profile report is generated for each executed query. It contains query metrics that can be used to analyze query performance. To view a Query Profile report, click on the Jobs icon on the left side panel in the Dremio UI, then click on the Job ID of the query you want to analyze, then click on the Raw Profile link on the top right.

If you need to provide this information to Dremio Support, you generate the zip file using the ‘Download Profile’ button at the bottom of the page.


kamran.hussain
Start Time: All
Status
UI, +1
User
Manage Columns
Job ID
User
Dataset
Query Type
Queue
Start Time
Duration
SQL
...628-cd2d-5cd80f854d00
kamran.hussain@dremi...
trips
UI (download)
High Cost User Q...
10/17/2023, 12:28:24
00:00:04
CREATE TABLE "__datasetDownload"."3c3c3f75-572
...a22-c86a-4f058c07cd00
kamran.hussain@dremi...
ny_taxi_trips_iceberg
UI (run)
Low Cost User Q...
10/25/2023, 06:09:59
00:00:03
delete from ny_taxi_trips_iceberg where passen
...0a08-4e3a-2c2f3f7cc300
kamran.hussain@dremi...
NYC Trips
JDBC Client
High Cost User Q...
10/17/2023, 12:24:11
00:00:03
SELECT * FROM "Business.Transportation"."NYC T
...11fe-2c50-3389dc35ad00
kamran.hussain@dremi...
NY_Trips
UI (run)
Low Cost User Q...
10/25/2023, 06:08:25
00:00:03
create table "s3"."kamran-dremio".ny_taxi_trip
...8a05-e33e-0a1f3ed69100
kamran.hussain@dremi...
time_travel
UI (run)
10/25/2023, 06:15:19
00:00:01
create vds "Solution Architect".KH.gp as selec
...8db-eb5c-93e4240f8800
kamran.hussain@dremi...
ny_taxi_trips_iceberg
UI (run)
Low Cost User Q...
10/25/2023, 06:05:30
<1s
---SELECT CURRENT_TIMESTAMP() --2023-09-05 14:
...988a-37a1-6f9092251200
kamran.hussain@dremi...
trips
UI (preview)
UI Previews
10/27/2023, 09:49:24
<1s
SELECT nested_0.vendor_id AS vendor_id, nested
1ac4320b-18f5-988a-37a1-6f9092251200
ny_taxi_trips_iceberg...
UI (run)
Low Cost User Q...
10/25/2023, 06:13:08
<1s
select count(*) from ny_taxi_trips_iceberg at
...1ec-2e3c-96e6b9ed2e00
kamran.hussain@dremi...
nyctaxi
UI (preview)
UI Previews
10/17/2023, 12:21:59
<1s
SELECT * FROM nyctaxi
...2b8-a341-8e2119842800
kamran.hussain@dremi...
Unavailable
UI (run)
10/25/2023, 06:14:37
<1s
create vds gp as select * from ny_taxi_trips_i


Jobs » 1ac4320b-18f5-988a-37a1-6f9092251200
Overview
SQL
Visual Profile
Raw Profile
Summary
Status: COMPLETED
Total Memory: 819.96 MB
CPU Used: 01.36s
Query Type: UI (preview)
Start Time: 10/27/2023 09:49:24
Duration: <1s
Wait on Client: <1s
User: kamran.hussain@dremio....
Queue: UI Previews
Input: 14.19 MB / 97.2K Rows
Output: 20.15 MB / 82K Rows
Total Execution Time
Pending
Metadata Retrieval
Planning
Queued
Execution Planning
Starting
Submitted SQL
1 SELECT nested_0.vendor_id AS vendor_id, nested_0.pickup_datetime AS pickup_datetime, nested_0.dropoff_date AS dropoff_date, nested_0.
dropoff_datetime, nested_0.passenger_count AS passenger_count, nested_0.trip_distance_mi AS trip_distance_mi, nested_0.pickup_longitu
nested_0.pickup_latitude AS pickup_latitude, nested_0.rate_code AS rate_code, nested_0.store_and_fwd_flag AS store_and_fwd_flag, nest
dropoff_longitude, nested_0.dropoff_latitude AS dropoff_latitude, nested_0.payment_type AS payment_type, nested_0.fare_amount AS fare_
AS surcharge, nested_0.mta_tax AS mta_tax, nested_0.tip_amount AS tip_amount, nested_0.tolls_amount AS tolls_amount, nested_0.total_a
nested_0.pickup_date AS pickup_date, join_weather_nyc."date" AS "date", join_weather_nyc.station AS station, join_weather_nyc.name AS
AS awnd, join_weather_nyc.prcp AS prcp, join_weather_nyc.snow AS snow, join_weather_nyc.snwd AS snwd, join_weather_nyc.tempmax AS tem
join_weather_nyc.tempmin AS tempmin
2 FROM (
Queried Datasets
trips
Preparation.trips
weather_nyc
pg.public.weather_nyc
Scans
trips_pickupDate
Acceleration
Reflections Used
Reflections Not Used
Aggregation Reflection [AlgMatch]
Business.Transportation.NYC Trips
Did not cover query

Query Lifecycle - Raw Profile

From the time you hit run to the time the query returns data, it goes through several phases. Let's take a look at each phase of the query. This information is available in the Raw Profile. Right below the SQL on the Query tab of the Raw Profile is the **Job Summary** section (see screenshot below). This section of the Profile will show the times associated with each query phase, which will help you determine what to focus on to improve the performance by reducing that wait time. (start with the highest wait time).

Phase 1: **Command pool** wait - Dremio can run n-1 queries in parallel, where n is the number of cores on the coordinator node. If more than n-1 queries are executed simultaneously, then those queries go into the command pool. Command pool wait time should be at most 2-3 seconds.

Phase 2: **Metadata retrieval** - command parsing, dataset retrieval from source and KV store. Metadata is stored on the executor nodes. (starting with v18).

> ⚠ IMPORTANT
> Metadata retrieval should be done before the query needs to be run. This step should be avoided during query execution time.

Phase 3: **Query planning time** - query planning should not take more than 500ms to 1 sec.

> ⚠ IMPORTANT
> If you see a long planning time, it's directly related to the coordinator resources, query complexity or the number of concurrent queries.

Phase 4: **Queued** - waiting for resources.

> ⚠ IMPORTANT
> If there is a lot of queueing, that would either mean that the concurrency is high or the queries are running for a long time. This would imply that the Dremio cluster is not correctly sized for the workload.

Phase 5: **Execution planning** - Dremio has the fully optimized plan at this point. Now, Dremio splits the plan and assigns it to the executor nodes.

Phase 6: **Starting** - propagates fragments (units of work) to the executors.

> ⚠ IMPORTANT
> If this step takes more than a few seconds, the executors may be too busy with other work. This would imply that the Dremio cluster is not correctly sized for the workload and you need to add more executors.

Phase 7: **Running** - waits for fragment completion
Completed, Failed, or Canceled

Below is an example screenshot of the Job Summary information:

Job Summary

State:	COMPLETED
Coordinator:	dremio-master-0.dremio-cluster-pod.default.svc.cluster.local
Threads:	7
Command Pool Wait:	0ms
Total Query Time:	77,814ms
# Joins in user query:	1
# Joins in final plan:	1
Considered Reflections:	3
Matched Reflections:	0
Chosed Reflections:	0

State Durations

Pending:	0ms
Metadata Retrieval:	60ms
Planning:	127ms
Engine Start:	0ms
Queued:	12ms
Execution Planning:	5ms
Starting:	3ms
Running:	77,607ms

This query ran for over 77 seconds. The **Total Query Time of** 77 seconds consists of 60 ms of **Metadata Retrieval,** 127 ms of **Planning**, and 77,607 ms of **Running**, etc. As with any query tuning exercise, you must identify and address the state that takes the longest time.

The metadata retrieval, planning and queued times are within the expected range; however, the Running time is unacceptable. Further investigation needs to be done to determine why the query took 77 seconds in the **Running** state. This paper's Optimization Concepts section will cover some possible reasons for extended run times.

Please refer to the Reading Dremio Job Profiles paper to learn more about the information in Dremio profiles and where to find it.

Common reasons for slow performance

Given the various states of query execution mentioned in the previous section, here we will briefly examine some common reasons for slow performance for a few of those states.

If the **Command Pool Wait** is more than 1-2 seconds, then either there were too many queries initiated at the same time (for the cores available on the coordinator), or the query planning is taking too long, which is consuming the cores, due to the complexity of the query. Please review the resources available on the Coordinator node(s) and take corrective actions. Dremio supports scale-out coordinator nodes, which may be needed to reduce this wait.

Metadata Retrieval should not happen at query runtime (aka inline refresh). If the duration of this phase is high, the metadata is either unavailable or possibly expired. Please review the configuration for the Source and adjust it accordingly:
Click on the Datasets icon > right-click on the Source in question > Settings > Metadata tab on the left.
Optionally, review the Doc "Administration > Refreshing Metadata" to avoid inline metadata refresh.

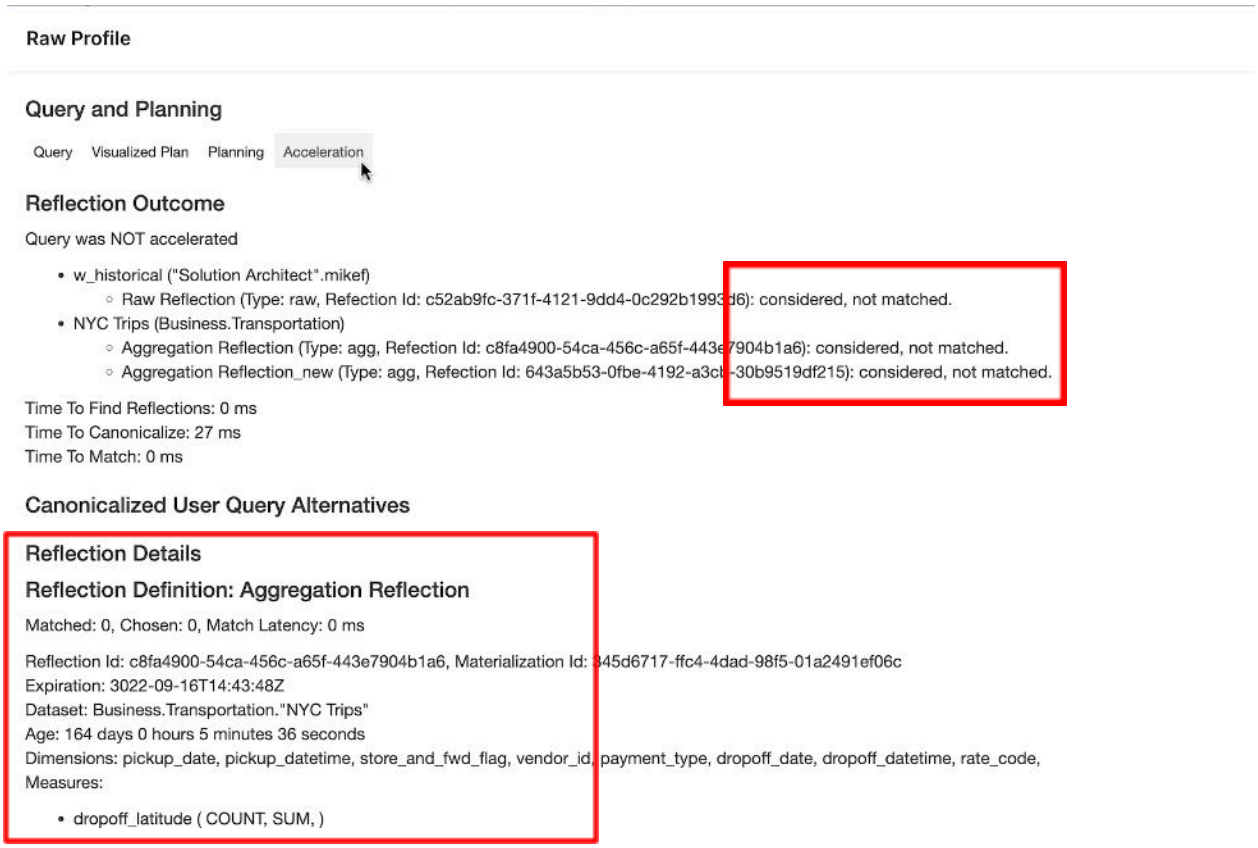
Details of the **Planning** phase are in the Raw Profile > Planning tab:

One possible reason for high **Planning** time is too many reflections on the datasets in the query. The planner has to analyze all the reflections, calculate the cost, compare the cost and choose the best reflection. This can take significant time, sometimes more than running the query and fetching the data. Scroll down in the Planning tab and look at the values of 'Normalization' and ' Logical Planning' (see screenshot below). If these are high, more than 10-20 seconds, then it points to having too many reflections. Please review how many reflections you have and if they are needed.

Starting with version 24.2, reflection hints are now available to control which reflections are considered or excluded. You can consider hints to accelerate reflection selection.

Next, you can go to the **Acceleration** tab and review the Reflection Outcome section.

As you can see in the example below, there are three reflections (one raw reflection and two aggregate reflections), and none were selected - the query was not accelerated. The reason the reflection was not selected is: 'considered, not matched'. In the Reflection Details section, you will find more information about the reflection, why it may or may not have been selected, the age of the reflection, expiration date, etc.



Let's briefly review a few of the steps of the **Planning** state:

1. Validate and Convert To Rel - Dremio looks at the physical and virtual datasets by pulling this information into the planning phase. These should typically be in milliseconds and not the cause for any slowness.
2. Logical Planning - reflection substitution, partition pruning, filter pushdown
3. Final Physical Plan (Optimized Plan) - Dremio determines how to divide the work into Phases (or units of work).

Scroll further in the report to get to the **Threads** section.

Threads

Overview

Phase	Weight	Thread Reporting	First Start	Last Start	First End	Last End	Min First-run	Avg First-run	Max First-run	Min Wall-clock	Avg Wall-clock	Max Wall-clock	Min Sleep	Avg Sleep	Max Sleep	Min Blocked	Avg Blocked	Max Blocked	Last Update	Prog
00-xx-xx	2	1 / 1	0.151s	0.151s	0.813s	0.813s	0.004s	0.004s	0.004s	0.662s	0.662s	0.662s	0.000s	0.000s	0.000s	0.653s	0.653s	0.653s	14:49:25	14:4
01-xx-xx	2	8 / 8	0.151s	0.163s	0.810s	0.822s	0.002s	0.003s	0.004s	0.657s	0.657s	0.659s	0.001s	0.008s	0.024s	0.607s	0.626s	0.638s	14:49:25	14:4
02-xx-xx	1	8 / 8	0.151s	0.163s	0.715s	0.822s	0.002s	0.006s	0.010s	0.560s	0.584s	0.659s	0.003s	0.005s	0.011s	0.002s	0.376s	0.510s	14:49:25	14:4
03-xx-xx	2	1 / 1	0.163s	0.163s	0.725s	0.725s	0.003s	0.003s	0.003s	0.562s	0.562s	0.562s	0.000s	0.000s	0.000s	0.494s	0.494s	0.494s	14:49:25	14:4
04-xx-xx	1	8 / 8	0.152s	0.164s	0.708s	0.744s	0.005s	0.009s	0.019s	0.555s	0.562s	0.580s	0.003s	0.009s	0.021s	0.009s	0.462s	0.549s	14:49:25	14:4
05-xx-xx	1	1 / 1	0.153s	0.153s	0.708s	0.708s	0.003s	0.003s	0.003s	0.555s	0.555s	0.555s	0.001s	0.001s	0.001s	0.497s	0.497s	0.497s	14:49:25	14:4

Each Phase can have different degrees of parallelism - how this is derived is discussed in the next section. The example above has 6 phases: 00, 01, ..., 05. Phases 00, 03 and 05 are single threaded. Look at the 'Thread Reporting' column, which shows 1/1. Phases 01, 02, and 04 have eight threads each.

Note: Phase 00 is always single-threaded; this is the phase that sends the results back to the client.

Degree of Parallelism

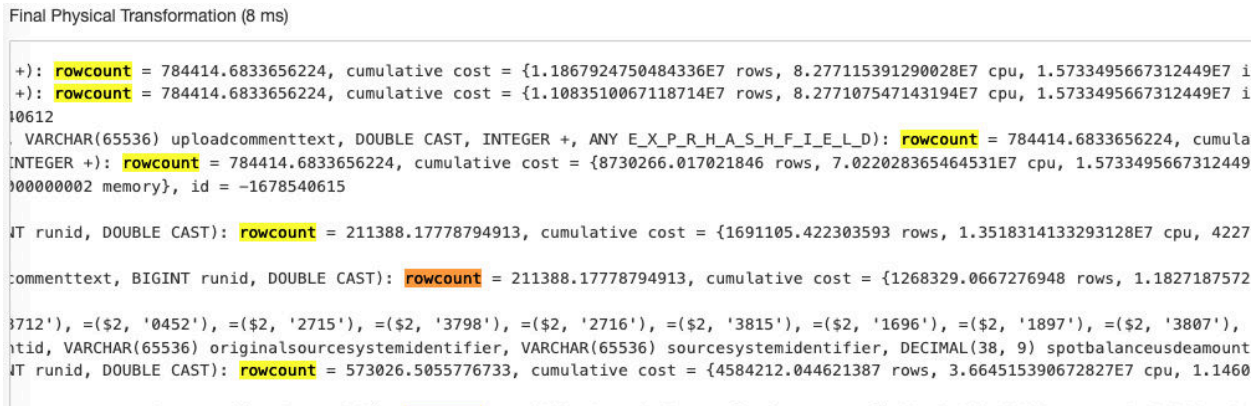
Based on the format of the data and the data source, the degree of parallelism in Dremio is determined in slightly different ways.

Parquet, ORC (read-only) and AVRO file formats fall into Dremio's unlimited splits flow, where the metadata is stored in the distributed store. During the Planning phase, the coordinator cannot access the number of splits, and partition pruning happens during the first part of query execution. In this case, the degree of parallelism is determined during the Execution Planning phase. It is calculated as the minimum of these two values:

- Number of cores on the executors (70% of executor cores for each query)
- Estimated number of rows. You can see each dataset's estimated number of rows in the Final Planning section. (search for 'rowcount')

For all other file and table formats and external sources, the metadata still resides in the Master Coordinator's KVstore and pruning happens during the Planning phase. In this case, the degree of parallelism is calculated as the minimum of these three values:

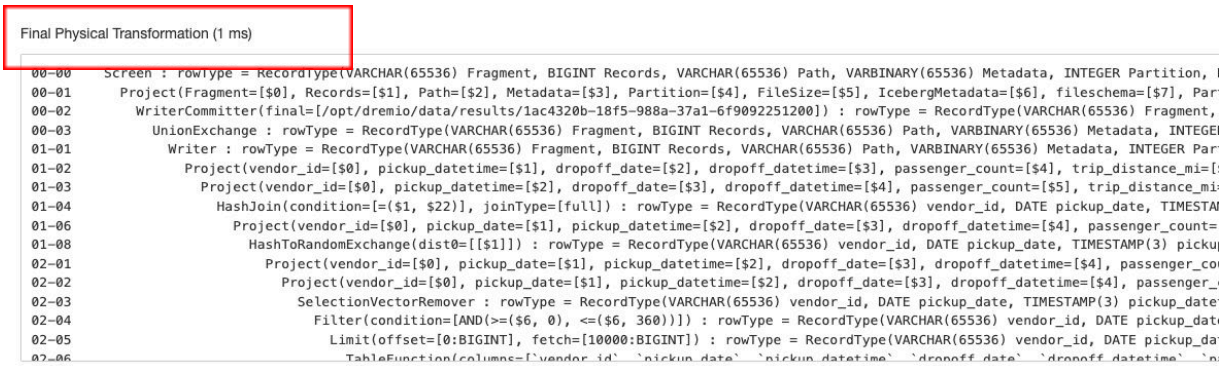
- Number of splits to be scanned (in Final Physical plan)
- Number of cores on the executors (70% of executor cores for each query)
- Estimated number of rows. You can see each dataset's estimated number of rows in the Final Planning section. (search for 'rowcount')



Optimization Concepts

Optimization concept #1: Runtime Filtering

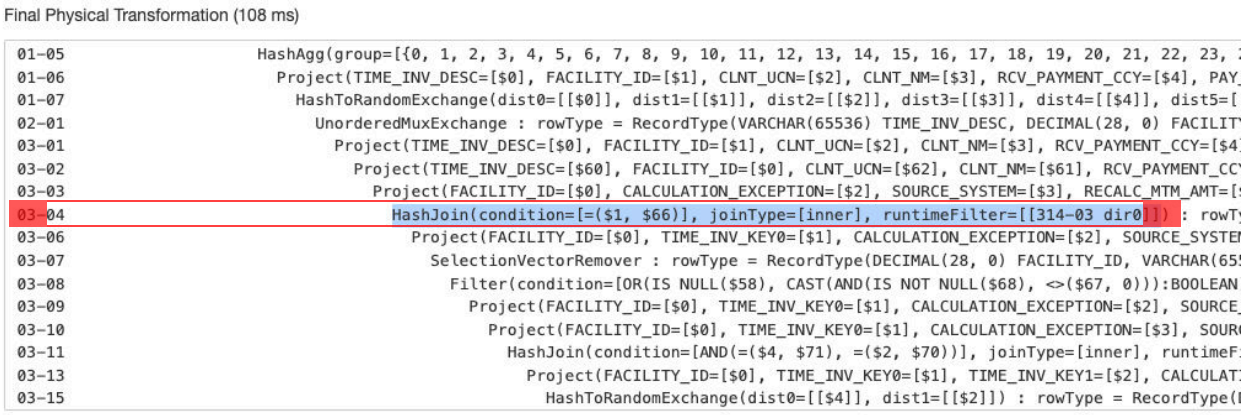
The remainder of this paper will focus on the Physical Plan. We're looking for optimization patterns that will ensure efficient execution.



> **NOTE**
> **If your query does an inner or right outer join - your query should use runtime filtering.**

Runtime filtering improves query performance by reducing the amount of data that will be projected when joining fact and dimension tables. Dremio will quickly determine the join keys from the dimension table and project only the matching data in the fact table, making this step much faster. This is only applicable for inner joins or right outer joins.

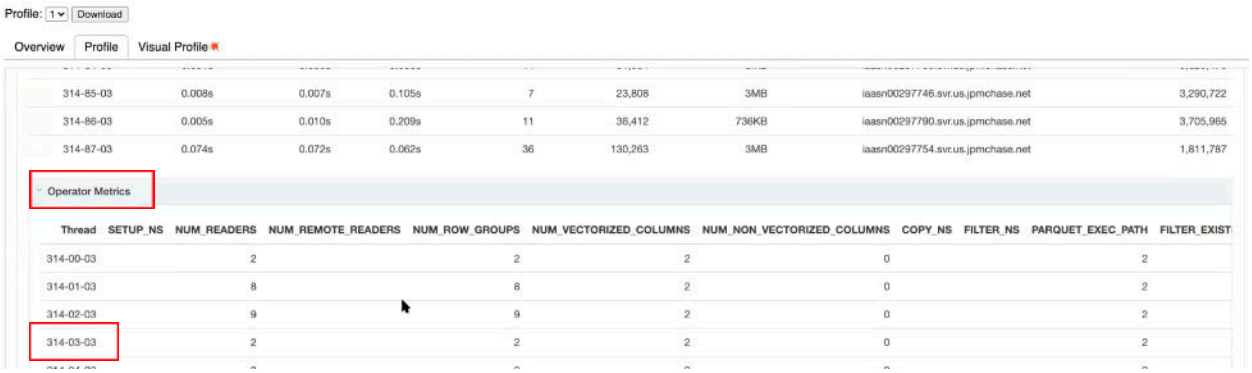
Let's now find out if the runtime filter was used (or not used). Click on the Profile tab > go to the Planning tab > and search for 'runtime' (see highlighted row in the screenshot below).

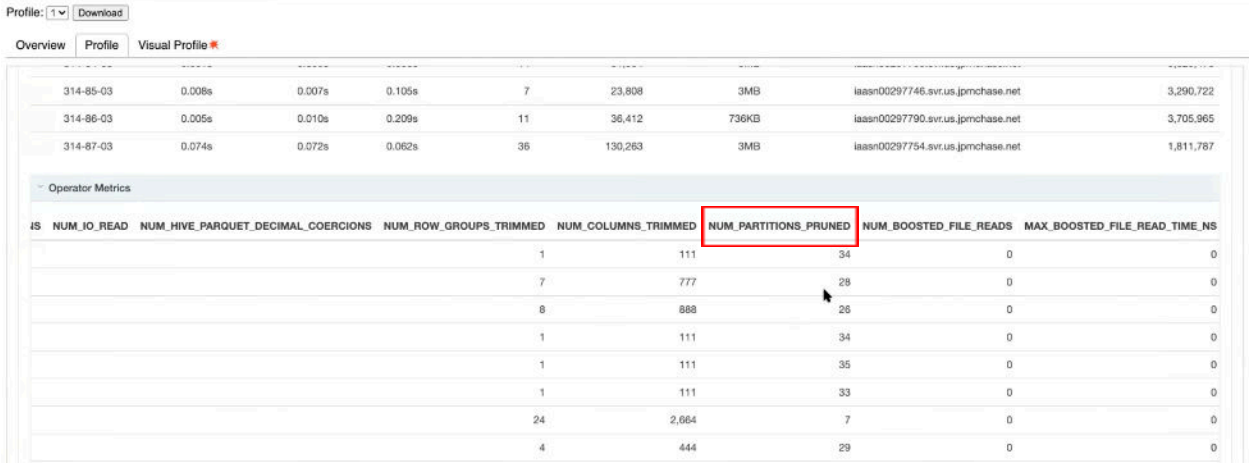


This example shows that the runtime filter is being used; it is a HashJoin, join type=[inner] join, and the column name used to filter is dir0. The scan that benefited from this filter was in Phase

314-03. (we'll use this phase to drill deeper into the operation to see the benefits of runtime filtering).

Referring to the screenshots below, scroll down to the **Operators** section > Find and expand the phase from the 'runtimeFilter' > then Expand **Operator Metrics** (in this example, it's Phase 314-03). Scroll to the right until you see NUM_PARTITION_PRUNED. Because of runtime filtering, the query is optimized. Dremio can skip this many row groups (approx. 30 per phase x number of phases = 80 in this example).





Optimization concept #2: Partition Pruning:

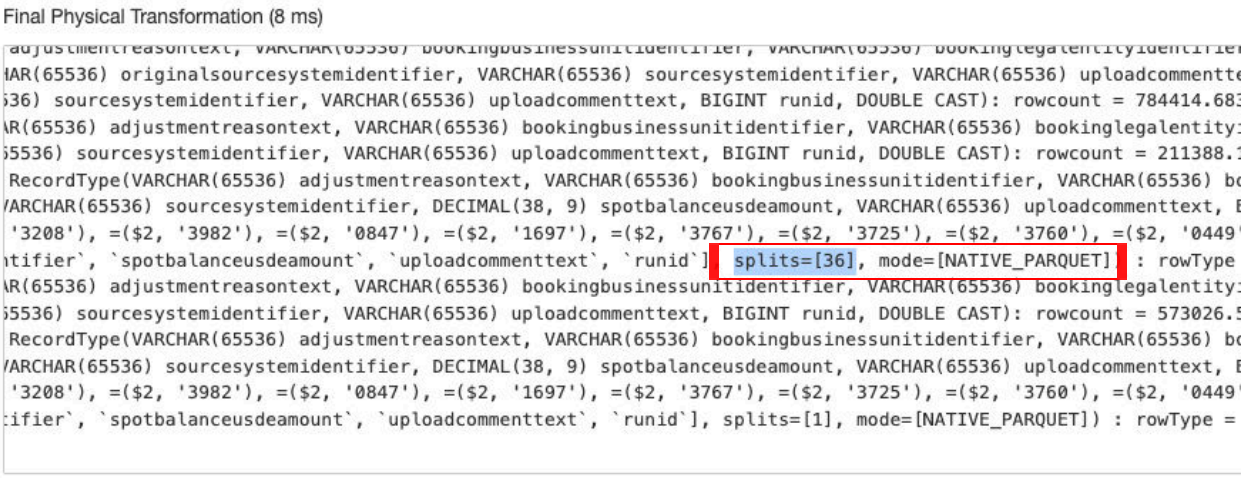
Every query should do partition pruning. If it is not, then it needs to be investigated.

To see if the query is doing partition pruning, let's look at the physical splits for the dataset and then compare that to the number of splits scanned to retrieve the data for this query.

To get the physical splits for the dataset, look at the raw metadata in the 'Convert To Rel' section in the **Planning** tab. Search for 'splits' - in the example below, the dataset has 214,109 physical splits.

Scroll left and capture the table for the dataset.

Next, scroll down to the Physical Plan (Final Physical Transformation) and search for that table name to get the number of splits scanned for that table. Scroll to the right and see the value of ‘splits’:



In this example, Dremio scanned only 36 splits out of 214,109 physical splits, resulting in a much faster query response time.

Optimization concept #3: Filter Push Down

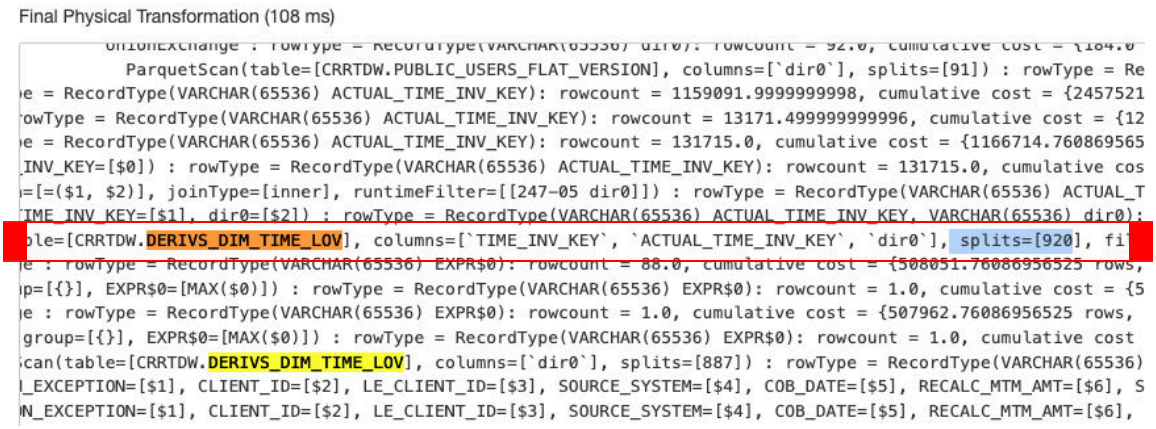
Filter Push Down is an optimization technique that minimizes the data that must be scanned, improving query performance. It's a feature of Dremio and nothing explicit has to be done by the data engineer to use this feature. Filter push-down is not as optimal as the other optimization methods discussed earlier - because Dremio will still need to open the splits and read the footer.

Let's look at a scenario when you are querying a dataset using a filter column, but the data is not partitioned by that column. In this case, the number of splits scanned will not be reduced, but the number of parquet bytes read should be less.

To identify a 'filter push down' in a query Profile, search for 'filters=' in the Planning tab > Final Physical Plan section. Look at the number of splits, which are the splits to be scanned, and compare it to the number of splits from the 'Convert to Rel' section for the same table, which shows the actual number of splits for that table. In the example below, you can see it's 920 in both cases. This means that there was no reduction in splits.

This section shows the actual number of splits for that table.

This section shows the number of scanned splits for that table.



However the query is still optimized because Dremio looks at the parquet file footer to see if that column exists in that file and only scans that file. Dremio will also know which row groups

have that column and only scan those row groups. You can see this in the Profile by looking at the number of parquet bytes read. Determine the phase where the filter happens > go to Operators > for that scan, expand the Operator Metrics and see the value of parquet_bytes_read.

It's essential to understand the impact of a filter on a non-partition column if there are **too many small files**. In this case, Dremio has to look at the footer of all the small files, which will adversely impact performance.

Conclusion

In this document, we introduced some key concepts to analyze and tune queries in Dremio. We started by looking at the metrics in the query profile report and took action accordingly.

Remember, looking at the whole cluster is essential, as other queries may impact the query. To minimize the impact of other workloads, you can set up engines to isolate workloads. Please review the Workload Management section in the Docs.

We recommend having a Dremio Professional Services resource do a thorough health check of your cluster at least once yearly. Please reach out to your account executive to request a health check.