dremio

Dremio Software

Dremio Upgrade Testing Framework

Introduction

This document provides a framework for testing new Dremio versions on non-production environments for performance, stability, and regressions ahead of an upgrade of the respective production cluster.

The performance testing aspect of this document may also be used to test and evaluate cluster sizing and scaling, independent of a Dremio version upgrade.

This document is not intended to provide detailed steps on the upgrade process itself.

For detailed instructions on how to install, upgrade, or migrate a Dremio cluster on Kubernetes, please refer to the Helm chart README, Dremio documentation, or the Migrate a Dremio Standalone Cluster to Kubernetes whitepaper.

Assumptions

There are several assumptions to consider as you go through this document.

- The non-production and production environments are close or identical to each other (with regard to tables and schemas).
- For accurate performance testing, the non-production and production environments must also be close or identical to each other with regard to cluster size and available resources.
- The non-production environment runs Dremio version 24.3.2 or higher, allowing us to take advantage of the latest sys.jobs_recent system table.
- Historical queries from the production cluster are available (via queries.json files) or can be collected by a cluster admin (e.g. via Dremio Diagnostic Collector).

Prerequisites

- **Java 8+**: Check if you have Java installed on your machine by running `java --version`
- **Executable**: Clone the `dremio-stress` repo and follow the steps to build the Java executable.
  → run `./script/build` to create a JAR file in your target folder
- **Query Types**: Running REST API and JDBC queries are supported out of the box, so no additional drivers need to be installed.
- **Queries**: The `dremio-stress` tool can run queries based on a manually specified config JSON file or auto-generated queries from (zipped) Dremio queries.json files.

Methodology

This Dremio upgrade testing framework contains two main components: Functional testing and performance testing. The testing will cover re-running historical queries from a production cluster, as well as load testing for high concurrency.
The testing framework leverages an open-source tool called `dremio-stress`, which specifies and sends queries to Dremio. For more details on `dremio-stress`, please refer to its Github repository.

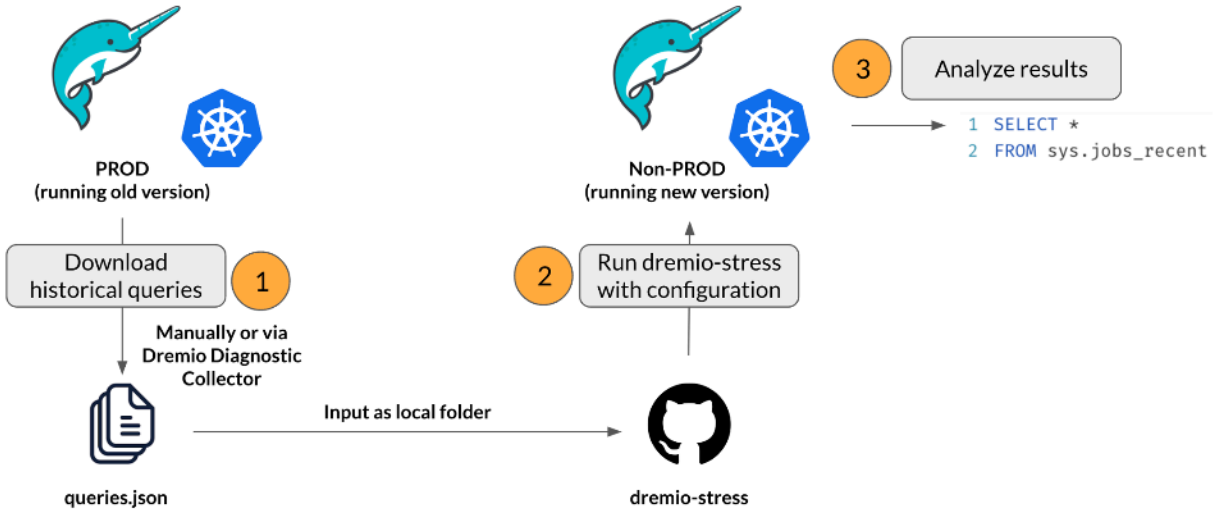
Functional Testing (Re-running historical queries)

Re-running historical queries from an existing cluster will require downloading its `queries.json` files, which will then act as the input for dremio-stress.
Downloading logs, like `queries.json`, can be done either manually or via Dremio's official tool Dremio Diagnostics Collector.

The goal of functional testing is to ensure that all queries that previously ran on the production cluster will be verified to perform on the new version.
As part of this re-run, we can verify that no significant performance regressions of individual queries have occurred.

The main configuration parameters to consider are the following:

- In this query replay scenario, the query concurrency is expected to be either relatively low or equal to one. This behavior is controlled by the command line flag `-q / --max-queries-in-flight` set to a low number, like 1.
- In addition, the execution sequence of the input queries is assumed to be in sequence, so it can be paused and resumed at any point. This behavior is controlled by the command line flag `-x / --execution-sequence` set to "`SEQUENTIAL`".
- When using queries.json files as the query input, we set `-g / --generator-type` to "`QUERIES_JSON`".

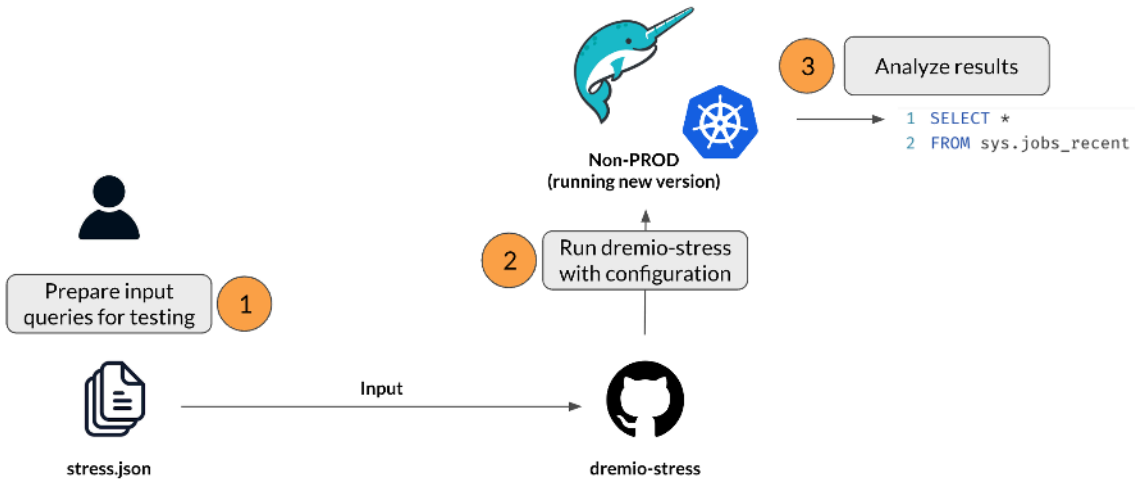


Performance Testing (Load or stress testing)

Stress testing can be seen as non-functional upgrade testing. After having successfully re-run historical queries in a relatively low concurrency to ensure that they satisfy the functional requirements, we can now focus on testing and benchmarking Dremio performance under load. This stage can be run using the same historical queries as in the first stage. However, it is advisable to prepare manually curated tests and queries to create a more standardized benchmark. These benchmark queries can be specified using the `stress.json` input format, which allows queries, query groups, parameters, and frequencies to be set (example stress.json).

The main configuration parameters to consider are the following:

- In this stress test scenario, the query concurrency is expected to be high. Thus, we can set `-q / --max-queries-in-flight` to its maximum of 32. To simulate even higher workloads, we can run several `dremio-stress` client applications in parallel with 32 concurrent queries each.
- In this scenario, we are less likely to care about the execution sequence of the input queries. Therefore, `-x / --execution-sequence` can be set to "`RANDOM`".
- When using a `stress.json` file as the query input, we set `-g / --generator-type` to "`STRESS_JSON`".



Configuring and running dremio-stress

Step-by-step description

1. Follow the steps described in the README to build the Java executable on the client machine used to run the queries.
2. If `queries.json` is being used as input, verify that the logic for skipping input queries is adequate for your use case. The logic is defined in the `skipQuery()` method here.
    a. Skipping of Dremio-internal queries (run by user $dremio$)
    b. Skipping of queries that did not complete
    c. Skipping of non-SQL query texts (e.g. “NA” from ODBC catalog calls)
    d. Skipping of DDL/DML queries (e.g. CREATE, DROP, INSERT etc.)
3. Run the JAR using either REST or JDBC (this can be controlled or limited by the duration parameter) while referencing the downloaded Dremio queries.json file or folder.
   A list of CLI flags can be found here.

Example

If you are located in the `dremio-stress` root folder and have previously run the `./scripts/build` command, you (as of v0.3.0) should be able to execute the following JAR:

```
java -jar ./target/dremio-stress.jar \
      -g QUERIES_JSON \
      --protocol JDBC \
      -l "jdbc:arrow-flight-sql:<CONNECTION_URL>" \
      -q 1 \
      -x SEQUENTIAL \
      ./queries_json_folder
```

Please review the Dremio documentation for how to determine the `<CONNECTION_URL>` for Arrow Flight JDBC.

Analyzing the results

Assuming we are using Dremio version 24.3.2 or higher, we can leverage the system table `sys.jobs_recent` to analyze query failures and query performance from within Dremio using SQL.

Errors and cancellations

Assuming the environments to be tested are not completely identical, we should see some errors highlighting missing views, objects, or columns. These can be excluded from our analysis since they are unlikely to have been caused by the Dremio version upgrade.
Any remaining error messages should then be grouped and analyzed for potential causes.

Total summary of errors and cancellations

In the following examples, we analyze results by filtering on `dremio-stress` queries for the time period of our testing using the predicate:
`WHERE query like '--Replay of %' AND "submitted_ts" BETWEEN '2024-01-26 12:00:00' AND '2024-01-26 12:20:00'`.

This is a query for generating an overview of successful and failed `dremio-stress` queries:

```
SELECT status,
   count(DISTINCT query) AS unique_queries_run,
   count(*) AS total_queries_run
FROM sys.jobs_recent
WHERE query like '--Replay of %' AND "submitted_ts" BETWEEN '2024-01-26 12:00:00' AND
'2024-01-26 12:20:00'
GROUP BY status
```

abc status	# unique_queries_run	# total_queries_run
COMPLETED	725	1427
FAILED	39	39
CANCELED	23	23

Isolation of relevant error messages

Next, we collect all error messages of queries that failed in the run and try to qualify whether they are related to software changes, which should be investigated further, or whether they were caused by changes in the underlying data and schema, which can be ignored as part of

upgrade testing. In the former case, avenues of investigation might be to look at the detailed job profiles of the queries or to consult Dremio's logs for potential errors.

```
SELECT
   error_msg,
   status AS dremio_stress_outcome,
   count(*)
FROM sys.jobs_recent
WHERE query like '--Replay of %' AND "submitted_ts" BETWEEN '2024-01-26 12:00:00' AND
'2024-01-26 12:20:00'
   AND status = 'FAILED'
GROUP BY 1, 2
ORDER BY error_msg
```

abc error_msg	abc dremio_stress_outcome	# ctd
Column 'id' not found in any table	FAILED	6
Column 'id_col' not found in any table	FAILED	2
Object 'NYC-taxi-trips' not found. Please check that it exists in the selected context.	FAILED	2
Object 'jobs_recent_full' not found within '$scratch'. Please check that it exists in the se	FAILED	3
Object 'jobs_recent_full' not found within 'maxmargalithstorage.customer-test'. Please check	FAILED	1
Object 'jobs_recent_inc' not found within '$scratch'. Please check that it exists in the sel	FAILED	1
Object 'jobs_recent_inc' not found within 'dremio_space'. Please check that it exists in the	FAILED	4
Object 'jobs_recent_inc' not found within 'maxmargalithstorage.customer-test'. Please check	FAILED	1
Object 'subfolder' not found within 'dremio_space.dremio_folder'. Please check that it exist	FAILED	7
Object 'taxi_100' not found within 'demo_space'. Please check that it exists in the selected	FAILED	8
Object 'taxi_inc' not found within 'maxmargalithstorage.customer-test'. Please check that it	FAILED	3
Object 'tempIcebergTable' not found within 'maxmargalithstorage.customer-test'. Please check	FAILED	1

In the previous screenshot, we can see several errors of tables, views, and columns not being found in the underlying data, which suggest schema discrepancies compared to when (and where) the query was originally run. To exclude these types of results from our analysis, we can adjust the filter condition with logic as follows:

```
status = 'FAILED' AND NOT ((error_msg LIKE 'Column%' OR error_msg LIKE 'Object%')
AND error_msg LIKE '%not found%')
```

During analysis, we can keep iterating on the filter rules, until we isolate the query errors that matter in the context of a software version update.

Canceled queries from `dremio-stress` run

We should also identify the queries that were canceled, to have them re-run:

```
SELECT DISTINCT
   SUBSTRING(query FROM 50) AS original_query,
```

```
   status AS dremio_stress_outcome
FROM sys.jobs_recent
WHERE query like '--Replay of %' AND "submitted_ts" BETWEEN '2024-01-26 12:00:00' AND
'2024-01-26 12:20:00'
   AND status = 'CANCELED'
ORDER BY original_query
```

Query cancellations may happen on the client side due to the volume of queries getting replayed. Re-run those queries, either via `dremio-stress` or manually, to ensure full test coverage.

Performance

Note: This following example assumes that performance benchmarks for both pre- and post-upgrade workloads were run on the same Dremio cluster. This approach can simplify analysis since we do not need to export data from one cluster to another. Instead, we can rely on the `sys.jobs_recent` table to compare results directly inside Dremio.

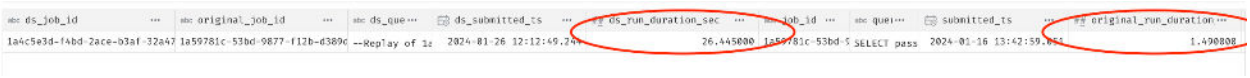
- Based on the `dremio-stress` query metadata, which is added as a SQL text comment to every query, we can match new queries with their original counterparts. If we observe significant negative deviations in query times, we should investigate potential causes.
- Account for aspects such as metadata and reflection refreshes, which may slow down the initial query immediately following the upgrade.
- Some performance divergences may be ignored, if the underlying queries are of time criticality, such as background jobs.

The following example query joins query replays ("`--Replay of xyz`") with the original jobs via their job_id. We apply a filter that focuses on queries that

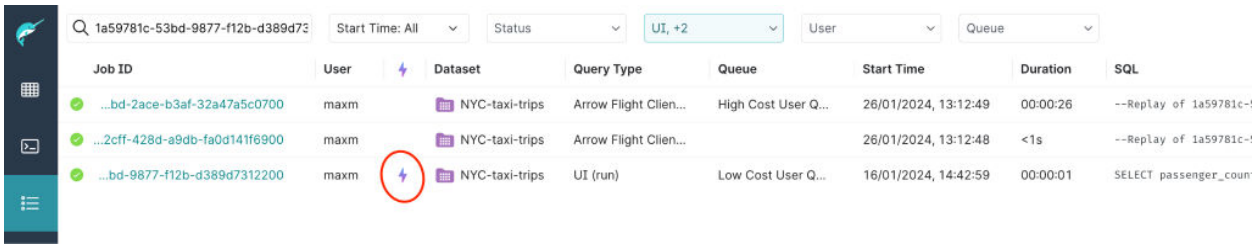
A. took more than twice as long in the `dremio-stress` replay than they did originally and
B. had a minimum running time of 1 second to ignore outliers (`WHERE ds.ds_run_duration_sec/j.original_run_duration_sec > 2.0 AND j.original_run_duration_sec > 1.0`).

```
WITH dremio_stress_queries AS (
   SELECT
       job_id AS ds_job_id,
       SUBSTRING(query FROM 13 FOR 36) AS original_job_id,
       query AS ds_query,
       submitted_ts AS ds_submitted_ts,
       (final_state_epoch_millis - attempt_started_epoch_millis)/1000.0 AS
ds_run_duration_sec
  FROM sys.jobs_recent
  WHERE query like '--Replay of %' AND "submitted_ts" BETWEEN '2024-01-26 12:00:00' AND
'2024-01-26 12:20:00'
), jobs_recent AS (
   SELECT
      job_id,
      query,
      submitted_ts,
      (final_state_epoch_millis - attempt_started_epoch_millis)/1000.0 AS
original_run_duration_sec
  FROM sys.jobs_recent
)
```

```
SELECT *
FROM dremio_stress_queries ds
JOIN jobs_recent j
ON ds.original_job_id = j.job_id
WHERE
   ds.ds_run_duration_sec/j.original_run_duration_sec > 2.0 AND
   j.original_run_duration_sec > 1.0
```



We can then search the Dremio jobs tab for the Job ID to analyze further. In this case, the query was significantly faster on the older Dremio version because it used a reflection.



Conclusion

This paper described a framework, based on the open-source tool `dremio-stress`, which allows Dremio administrators to test and benchmark both the functionality and performance of their Dremio cluster before and after version upgrades. The test result analysis can be done in SQL thanks to the Dremio-native system table `sys.jobs_recent`.

Appendix

Quick Recap: How to upgrade a Dremio cluster (on Kubernetes)

Dremio Software is most commonly deployed using Helm charts on Kubernetes. The Helm chart repo can be found on GitHub.

For a detailed instruction on how to install or migrate a Dremio cluster on Kubernetes, please refer to the Helm chart README, Dremio documentation, or the Migrate a Dremio Standalone Cluster to Kubernetes whitepaper.

After running a Dremio backup (just to be safe), upgrading a Dremio Cluster on Kubernetes follows three simple steps:

1. Pull and merge the latest version of Dremio's official helm charts dremio-cloud-tools into your cloned git repository. Please make sure to review any new changes, features and fixes that have been introduced
2. Pull the latest Dremio version image from Dremio's official Dockerhub repository and upload it to your container registry.
3. In the values.yaml file of Dremio's helm chart, update the image reference and version number to the image from Step 2, for example:

   ```
   image: dremio/dremio-ee
   imageTag: 25.0.0
   ```

   then run the `helm upgrade` command.