Starburst 性能白皮书三 - Presto Dynamic Filtering

Dynamic filtering optimizations significantly improve the performance of queries with selective joins by avoiding reading of data that would be filtered by join condition. In this respect, dynamic filtering is similar to join pushdown discussed above, however it is the equivalent of inner join pushdown across data sources. As a consequence we derive the performance benefits associated with selective joins when performing federated queries. That significantly boosts the performance of query federation use cases.

Consider the following query which captures a common pattern of a fact table store_sales joined with a filtered dimension table date_dim:

```
SELECT count(*)
FROM store_sales JOIN date_dim
    ON store_sales.ss_sold_date_sk = date_dim.d_date_sk
WHERE d_following_holiday='Y' AND d_year = 2000;
```

Without dynamic filtering, Trino pushes predicates for the dimension table to the table scan on date_ dim, and it scans all the data in the fact table since there are no filters on store_sales in the query. The join operator ends up throwing away most of the probe-side rows as the JOIN criteria is highly selective.

When dynamic filtering is enabled, Trino collects candidate values for join condition from the processed dimension table on the right side of join. In the case of broadcast joins, the runtime predicates generated from this collection are pushed into the local table scan on the left side of the join running on the same worker.

Dynamic filtering is enabled by default using the enable-dynamic-filtering configuration property. To disable dynamic filtering, set the configuration property to false. Alternatively, use the session property enable_dynamic_filtering.

Additionally, these runtime predicates are communicated to the coordinator over the network so that dynamic filtering can also be performed on the coordinator during enumeration of table scan splits.

For example, in the case of the Hive connector, dynamic filters are used to skip loading of partitions which don't match the join criteria. This is known as dynamic partition pruning.

The results of dynamic filtering optimization can include the following benefits:

- improved overall query performance
- reduced network traffic between Trino and the data source
- reduced load on the remote data source

Support for push down of dynamic filters is specific to each connector, and the relevant underlying database or storage system.

Analysis and confirmation

Dynamic filtering depends on a number of factors:

- Planner support for dynamic filtering for a given join operation in Trino. Currently inner and right joins with =, = or IS NOT DISTINCT FROM join conditions, and semi-joins with IN conditions are supported.
- Connector support for utilizing dynamic filters pushed into the table scan at runtime. For example, the Hive connector can push dynamic filters into ORC and Parquet readers to perform stripe or row-group pruning.
- Connector support for utilizing dynamic filters at the splits enumeration stage.
- Size of right (build) side of the join.

You can take a closer look at the EXPLAIN plan of the query to analyze if the planner is adding dynamic filters to a specific query's plan. For example, the explain plan for the above query can be obtained by running the following statement:

```
EXPLAIN
SELECT count(*)
FROM store_sales JOIN date_dim
    ON store_sales.ss_sold_date_sk = date_dim.d_date_sk
WHERE d_following_holiday='Y' AND d_year = 2000;
```

The EXPLAIN plan for this query shows dynamicFilterAssignments in the InnerJoin node with dynamic filter df_370 collected from build symbol d_date_sk. You can also see the dynamicFilter predicate as part of the Hive ScanFilterProject operator where df_370 is associated with probe symbol ss_sold_date_sk. This shows you that the planner is successful in pushing dynamic filters down to the connector in the query plan.

```
Fragment 1 [SOURCE]
  Output layout: [count_3]
  Output partitioning: SINGLE []
  Stage Execution Strategy: UNGROUPED_EXECUTION
  Aggregate(PARTIAL)
    Layout: [count_3:bigint]
```

```
    count_3 := count(*)
    InnerJoin[(""ss_sold_date_sk"" = ""d_date_sk"")][$hashvalue,
    $hashvalue_4]
      Layout: []
      Estimates: {rows: 0 (0B), cpu: 0, memory: 0B, network: 0B}
      Distribution: REPLICATED
      dynamicFilterAssignments = {d_date_sk -> #df_370}
      ScanFilterProject[table = hive:default:store_sales, grouped =
      false, filterPredicate = true, dynamicFilters = {""ss_sold_date_
      sk"" = #df_370}]
        Layout: [ss_sold_date_sk:bigint, $hashvalue:bigint]
        Estimates: {rows: 0 (0B), cpu: 0, memory: 0B, network: 0B}/
        {rows: 0 (0B), cpu: 0, memory: 0B, network: 0B}/{rows: 0 (0B),
        cpu: 0, memory: 0B, network: 0B}
        $hashvalue := combine_hash(bigint '0',
        COALESCE(""$operator$hash_code""(""ss_sold_date_sk""), 0))
        ss_sold_date_sk := ss_sold_date_sk:bigint:REGULAR
        LocalExchange[HASH][$hashvalue_4] (""d_date_sk"")
          Layout: [d_date_sk:bigint, $hashvalue_4:bigint]
          Estimates: {rows: 0 (0B), cpu: 0, memory: 0B, network: 0B}
          RemoteSource[2]
            Layout: [d_date_sk:bigint, $hashvalue_5:bigint]

Fragment 2 [SOURCE]
    Output layout: [d_date_sk, $hashvalue_6]
    Output partitioning: BROADCAST []
    Stage Execution Strategy: UNGROUPED_EXECUTION
    ScanFilterProject[table = hive:default:date_dim, grouped = false,
    filterPredicate = ((""d_following_holiday"" = CAST('Y' AS char(1)))
    AND (""d_year"" = 2000))]
        Layout: [d_date_sk:bigint, $hashvalue_6:bigint]
        Estimates: {rows: 0 (0B), cpu: 0, memory: 0B, network: 0B}/
        {rows: 0 (0B), cpu: 0, memory: 0B, network: 0B}/{rows: 0 (0B),
        cpu: 0, memory: 0B, network: 0B}
        $hashvalue_6 := combine_hash(bigint '0',
        COALESCE(""$operator$hash_code""(""d_date_sk""), 0))
        d_following_holiday := d_following_holiday:char(1):REGULAR
        d_date_sk := d_date_sk:bigint:REGULAR
        d_year := d_year:int:REGULAR
```

During execution of a query with dynamic filters, Trino populates statistics about dynamic filters in the QueryInfo JSON available through the Web UI. In the queryStats section, statistics about dynamic filters collected by the coordinator can be found in the dynamicFiltersStats

structure.

Push down of dynamic filters into a table scan on the worker nodes can be verified by looking at the operator statistics for that table scan. dynamicFilterSplitsProcessed records the number of splits processed after a dynamic filter is pushed down to the table scan.

Dynamic filters are reported as a part of the EXPLAIN ANALYZE plan in the statistics for ScanFilterProject nodes.