SHUFFLE: A DEEP-DIVE

- Shuffling: data points randomly rearranged (#1 enemy of parallelism!)
- Parallel pipeline: no need of specific chunk of data to be in one machine - can be split out
- Level of parallelism in a pipeline can be scattered. (more parallel = more split, less shuffle)

⚠ How the data is structured determines the SQL keywords to use & impact parallelism differently.

\+

- SELECT
- FROM
- WHERE

SELECT + FROM + WHERE w/o WINDOW function = Infinitely scalable query

Instant & Cheap Processing, NO Shuffle,... as parallel as you want!

Extreme case: 1 bio machines to process 1 bio rows (1 row per machine). Each machine doesn't need more data to evaluate 1 row (fits SELECT + FROM + WHERE conditions)

- GROUP BY
- JOIN
- HAVING

GROUP BY

- Triggers shuffle (need to have all data from given id in one machine)
- Issue: aggregations only right if all data is in same machine (otherwise no means to validate if aggregation total rows is correct).

Two ways to fix shuffle from GROUP BY:

1. Bucketization

   Split the data into n buckets using a key w/ high cardinality (ie. user_id, device_id). It pre-shuffles the data doing all the modulus grouping when you writing the data out in your computing solution (Spark, S3, Iceberg, Delta Lake,...) . Then no need to do shuffle (buckets are set & data is guarateed in a given bucket).

2. Reduce data volume

JOIN

Same previous process but repeated twice! (keys of both left & right tables to be pushed to 1 machine

HAVING (hand in hand with GROUP BY)

- As parallel as SELECT, WHERE, but filtering condition adds step further (shuffle happens only after HAVING).

- ORDER BY

ORDER BY (avoid use in distributed computing). Uses:

- End of query: not parallelizable. A global data sorting is only feasible if data is in 1 machines (opposite to parallel).

Workaround: order by after data aggregation (reduced row volume)

- In window function : parallelizable if with 'OVER PARTITION' (no global sort).

\-

PARALLEL

SHUFFLE: A DEEP-DIVE

Levels on handling FACT data (the lower the level, the smaller the data & the less flexible it becomes)

A FACT DATA

- Schema: user_id, event_time, action, date_partition
- Properties:
  - High volume
  - 1 row per event
- Ideal for answering specific questions w/ small time horizons (less data volume used)

user_id	event_time	action	date	other_properties
3	2023-07-08T11:00:31Z	like	2023-07-08	{"os": "Android", "post": 1414}
3	2023-07-09T09:33:34Z	comment	2023-07-09	{"os": "iPhone", "post": 111}
3	2023-07-10T03:33:11Z	comment	2023-07-10	{"os": "Android", "post": 3434}

Tracking on action metrics at Facebook's approach for regular analysis on likes/comments data that look at a 90-day timeframe (fact, anonymized data)

B DAILY AGGREGATED FACT ~ 'Metric Repository'

- Schema: user_id, action_cnt, date_partition
- Properties:
  - Medium volume
  - 1 row per user per event (partitioned by date + metric).
  - Easy to bring dimensions in
- Ideal for longer time horizons (ie. 1-2 years)

user_id	metric_name	date	value
3	likes_given	2023-07-08	34
3	likes_given	2023-07-09	1
3	likes_given	2023-07-10	3

Deltoid Framework at
Internal A/B testing and experimentation tool.
Allows to know how features affect core metrics before launching them.
Deltoid tracks statistical movements between test & control groups against FB's core metrics (ie. DAU, MAU, user retention, engagement), a team's more relevant metrics (page views, interactions, shares), and key hypothesis and guardrail metrics.

C REDUCED FACT

Zach improved it (w/o comprising data loss)

- Schema: user_id, action_cnt array, month/yr_start_partition
- Properties:
  - Low volume,
  - 1 row per user per month/year (no aggregation, data maintained within array)
  - Less flexible, limited type of analytics
- Ideal for very long time horizons, large fact data

user_id	metric_name	month_start	value_array
3	likes_given	2023-07-01	[34, 3, 3, 4, 5, 6, 7, 7, 3, 3, 4, 2, 1, 5, 6, 3, 2, 1, 5, 2, 3, 3, 4, 5, 7, 8, 3, 4, 9]
3	likes_given	2023-08-01	[8, 3, 3, 4, 5, 0, 7, 0, 3, 3, 4, 2, 1, 5, 6, 3, 2, 1, 5, 2, 3, 3, 4, 5, 7, 8, 3, 4, 9]
3	likes_given	2023-09-01	[17, 3, 3, 4, 5, 6, 7, 7, 3, 3, 4, 2, 1, 5, 6, 3, 2, 1, 5, 2, 3, 3, 4, 5, 7, 8, 3, 4]

Zach's solution w/ Reduced Fact at
A backfill of FB entire data history (~10 years) processing time optimized from ~ 1 week to 1-2 days.
Also allowed:

1. Correlation analysis between user-level metrics & dimensions
2. Root-cause analysis

- Daily dates stored as offset
  - 1st index: month_start + zero_days
  - Last index: month_start + array_length - 1
- Dimensional joins impact performance (SCD accuracy is the same in month or year start). Solution: work with snapshots in time and treat dimensions as fixed.

Albert Campillo

Repost