GQL and Its Implementation @NebulaGraph

Xuntao Cheng

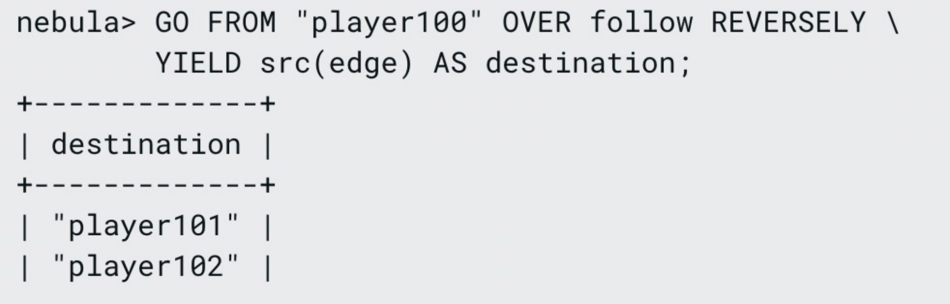
Staff Engineer, Vesoft Inc.

Before GQL

- NebulaGraph has been an open-source distributed graph database since 2018

- Primarily targeting at graph OLTP or latency-sensitive graph queries, with home-grown syntax

- Some compatibility with openCypher



```
nebula> GO FROM "player100" OVER follow REVERSELY \
        YIELD src(edge) AS destination;
+-------------+
| destination |
+-------------+
| "player101" |
| "player102" |
```

Users get stuck with writing/debugging bizarre queries

```
$top_stories = YIELD ”xxxxxxxxx” AS id

| GO FROM $-.id OVER relationship WHERE relationship.like > 0 AND relationship.highlight_time > 20 AND relationship.like >
timestamp(datetime(now()) - duration({days: 14})) YIELD DISTINCT relationship._dst AS id, relationship.highlight_time AS
highlight_time

| ORDER BY $-.highlight_time DESC

| LIMIT 10;

$final_user = GO FROM $top_stories.id OVER relationship REVERSELY WHERE relationship.like > 0 AND relationship.like >
timestamp(datetime(now()) - duration({days: 14})) AND relationship.highlight_time > 20 YIELD DISTINCT $top_stories.id as
story_id, relationship._dst as user_id LIMIT [100];

$shared_stories = GO FROM $final_user.user_id OVER relationship WHERE $final_user.user_id != $egoNode AND
relationship._dst != $final_user.story_id AND relationship.like > 0 AND relationship.like > timestamp(datetime(now()) -
duration({days: 14})) AND relationship.highlight_time > 20 YIELD $final_user.story_id AS top_level_id, $final_user.user_id
AS eng_user_id, relationship._dst AS share_id LIMIT [20000]

| GROUP BY $-.top_level_id, $-.share_id YIELD $-.top_level_id AS top_level_id, $-.share_id AS share_id, count($-
.eng_user_id) AS comm_user_cnt

| YIELD $-.top_level_id AS top_level_id, $-.share_id AS share_id, $-.comm_user_cnt AS comm_user_cnt

| ORDER BY $-.comm_user_cnt DESC

| LIMIT 100;$shared_props = FETCH PROP ON story $shared_stories.share_id YIELD properties(vertex).story_version AS
story_version, id(vertex) AS share_id;YIELD $shared_stories.top_level_id AS top_level_id, $shared_stories.share_id AS
share_id, $shared_stories.comm_user_cnt AS comm_user_cnt, $shared_props.story_version AS story_version FROM
$shared_stories INNER JOIN $shared_props ON $shared_stories.share_id == $shared_props.share_id;
```

MATCH queries were slow

- No modern query runtime, data structure, etc.
- Insufficient optimization techniques
- High overhead when querying the KV-based storage recursively for matching long path patterns
- …

How we view GQL

- A systematic & standardized graph-native query language.
- Users shall only need to declare patterns, not the detailed procedure to retrieve data.
- Potential to express simple or complex path patterns and graph algorithms in the same language, probably with some further extensions.

Our GQL database product

- Same distributed architecture, but with the new GQL frontend and a new kernel.

- We prioritize the GQL features that are necessary for LDBC queries.

- We use the LDBC SNB Interactive benchmark as our nightly regression test workload.

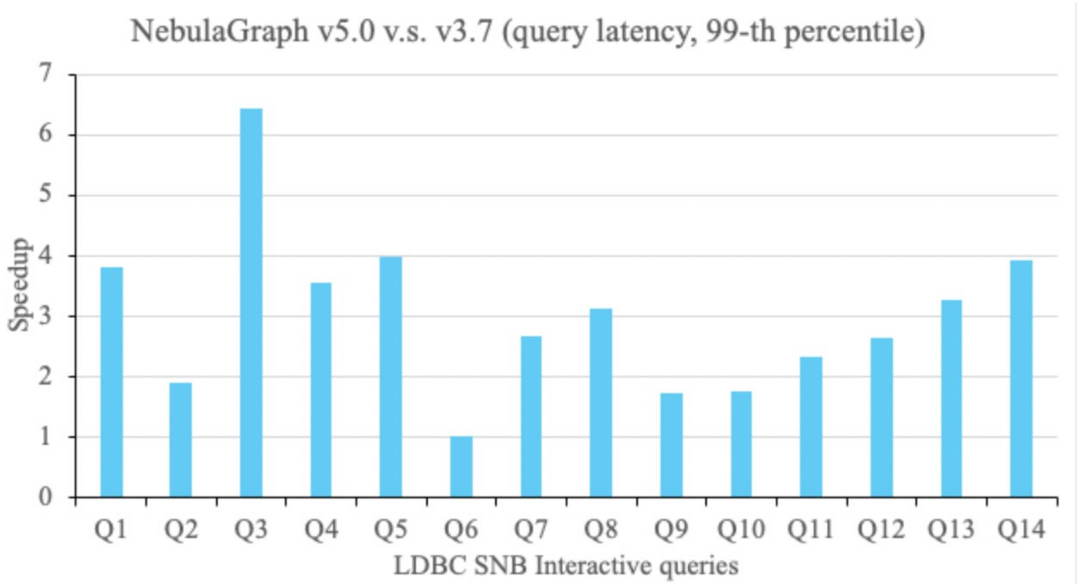
- Currently in RC, GA in two months.

Key kernel features

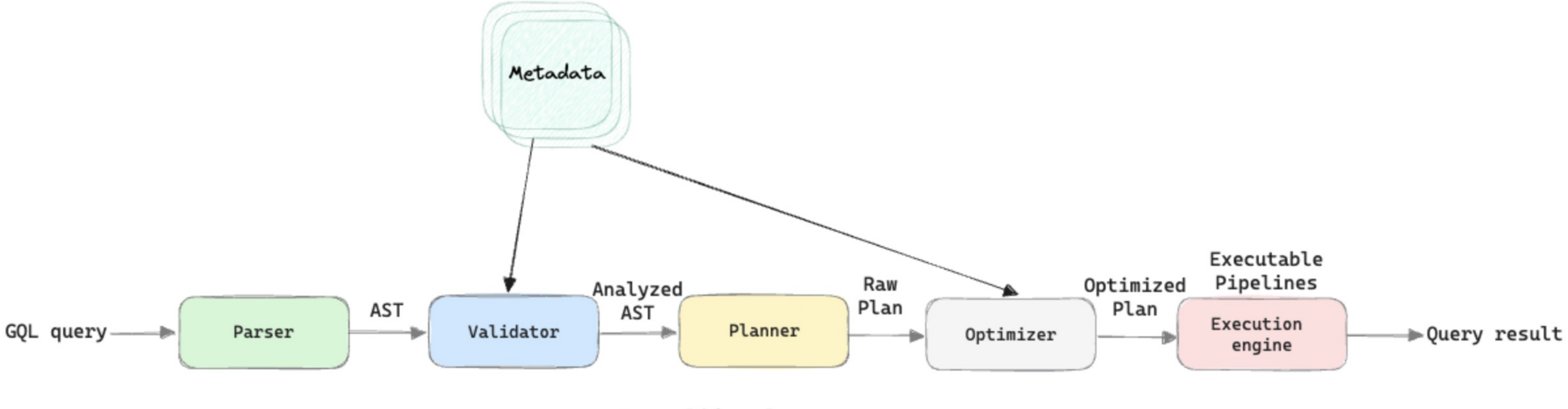
- Vectorized and Arrow-compatible data structure
- Zero-copy RPC
- Pipelined and vectorized execution runtime
- Sub-query plan pushdown
- In-memory graph
- …

- ~3x performance improvement for LDBC
- Reduced the memory footprint by 60-90%

- Still working on more projects

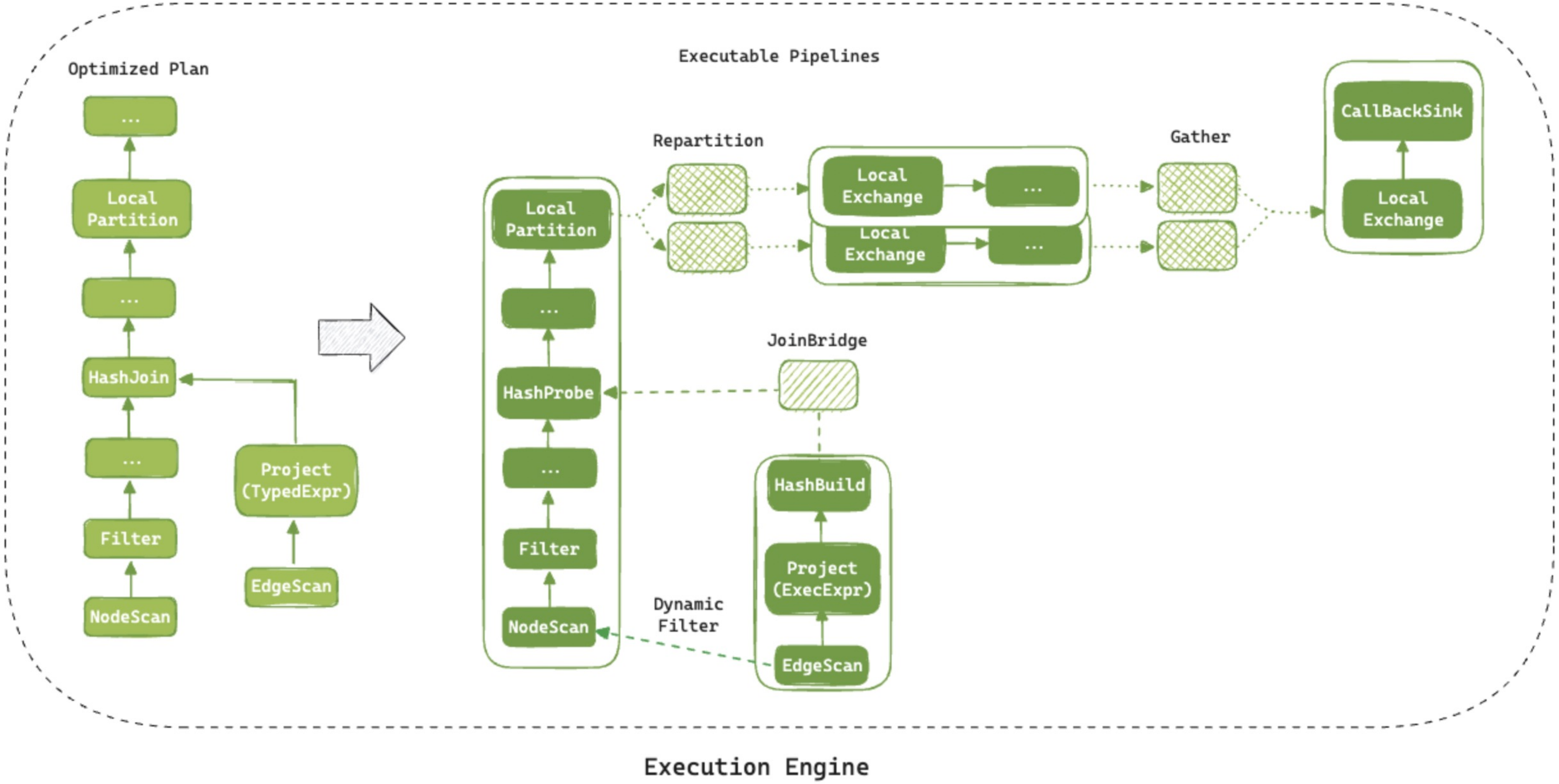


Lifecyle of a GQL query

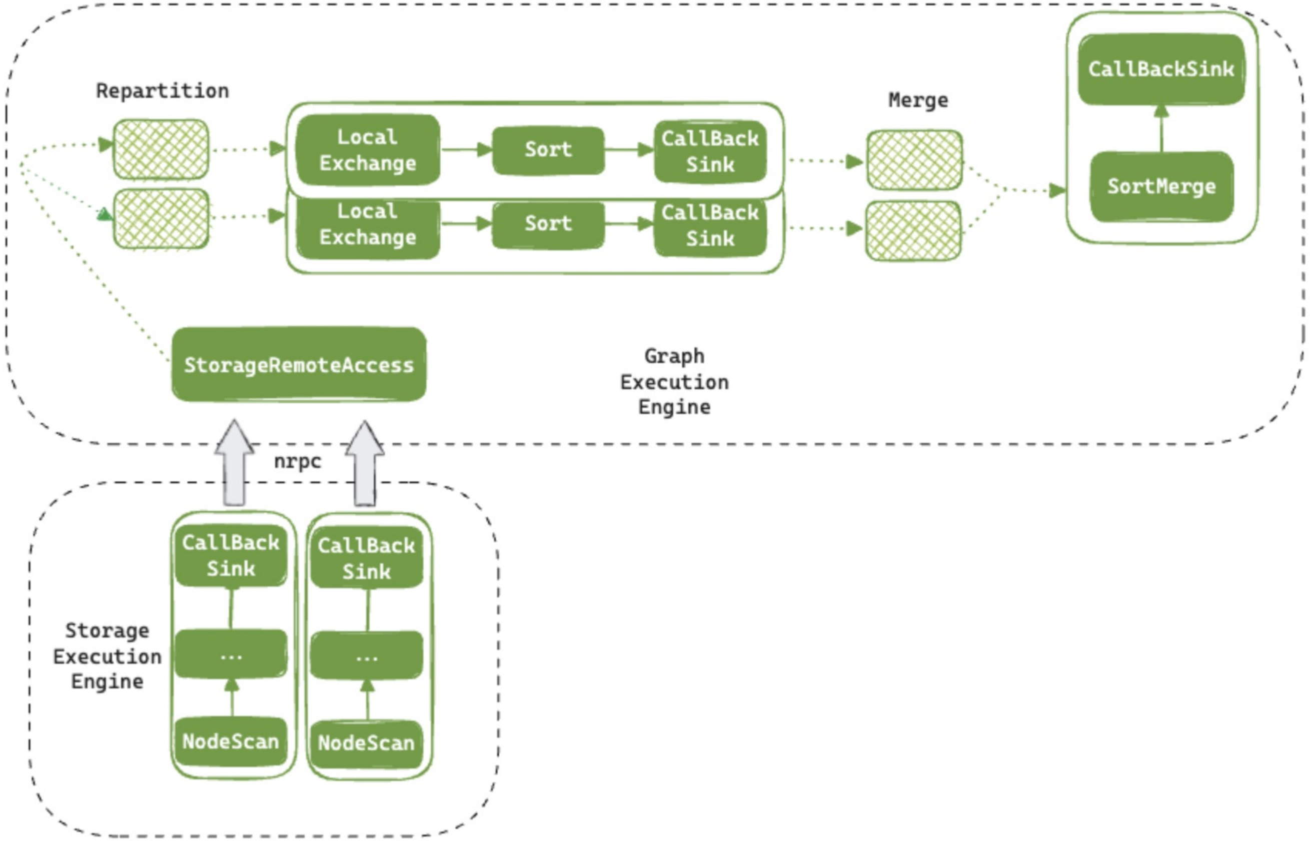


Query lifecycle

Execution Engine Overview



Sub-query push-down

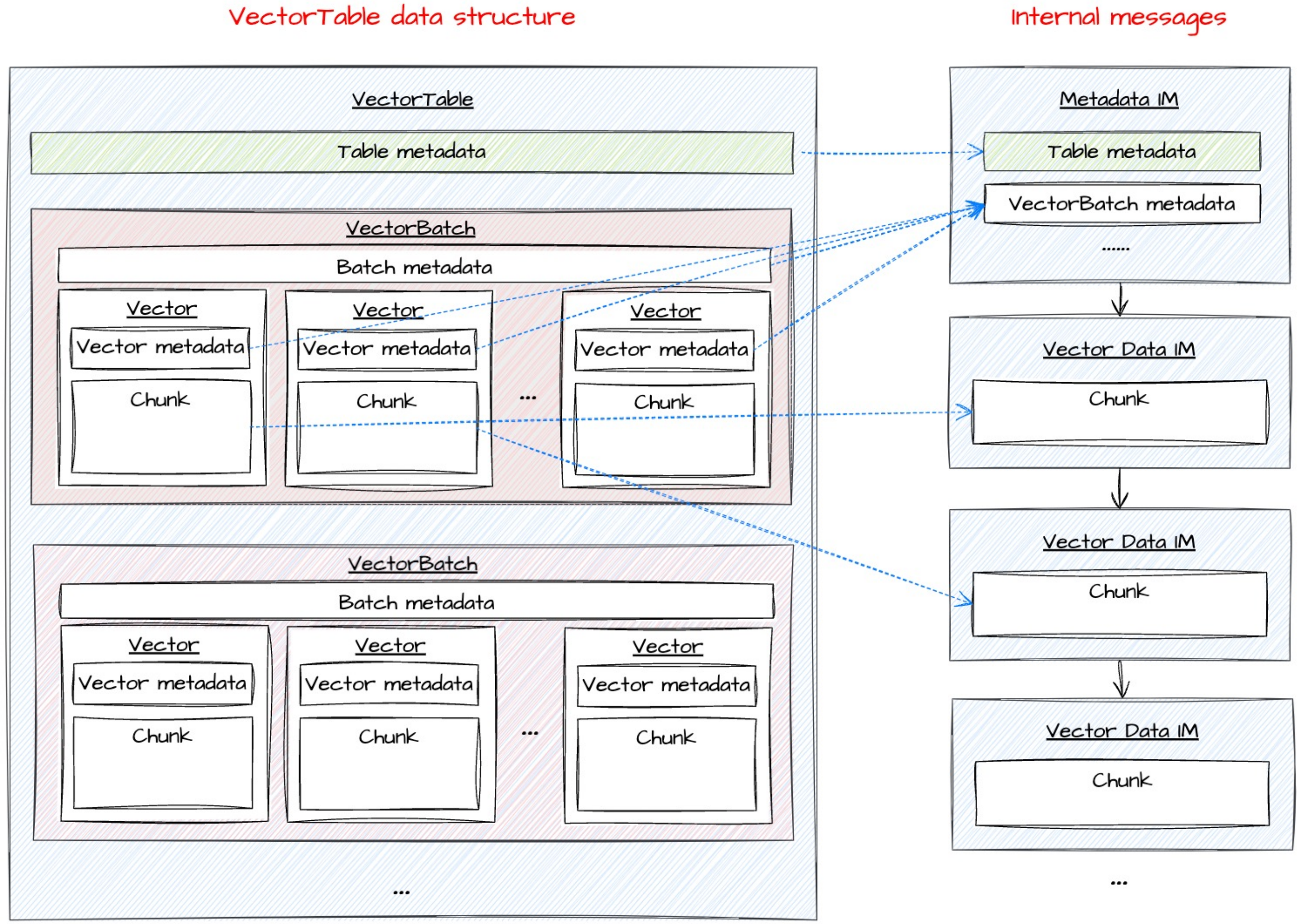


Nebula Vectors

- Arrow-like vectorized layouts for various data types

- Organized vectors in row-group batches for batched processing

- Zero-copy RPC between both the computing/storage nodes as well as the clients and the servers



Some language extensions we’ve made

- Tabular inserts

```
TABLE t {id, u8, f, l} = (1, 2, 1, [1, 2])
USE insert_cast
FOR r IN t
INSERT (@n {id: r.id, u8: r.u8, f: r.f, l: r.l})
```

- MATCH by node/edge type

```
MATCH (a@person) WHERE a.id = r.src
MATCH (b@person) WHERE b.id = r.dst
```

- Graph projection & temporary graphs

```
CREATE GRAPH PROJECTION proj_graph_2 OF job_graph_1 AS
VALUE { USE job_graph_1 MATCH (v:Person)-[e:WORK_FOR]-
>(v1:Company) RETURN GRAPH PROJECTION { v(name),
v1(name, revenue) } }
```

Thanks.