The wonderful life of a SQL query in a streaming database
RisingWave Labs

- Creator of the RisingWave Database
- OLAP streaming queries
- Incremental updates on materialized views
Views and Materialized Views

- What is a view?
- What is a MV?
- How do you calc views traditionally?
  - Full rebuild is expensive
- Incremental updates
  - Run aggregate on diff
  - Run background job that detects changes in base table
  - Triggers that fire if there is a change to the underlying table
- RisingWave: Incremental updates
CREATE_TABLE
CREATE_MATERIALIZED_VIEW
dev=> INSERT INTO stories (id, author, title, url) VALUES (1, 2, 'hacker story', 'some-url.net');
INSERT 0 1
dev=> SELECT * FROM stories;
id | author | title       | url
-----------------------------
1  | 2      | hacker story| some-url.net
(1 row)

dev=> INSERT INTO votes (user, story_id) VALUES (2, 1), (3, 1);
INSERT 0 2
dev=> SELECT * FROM votes;
user | story_id
-------
3  | 1
2  | 1
(2 rows)

dev=> SELECT * FROM StoriesWithVC;
id | author | title       | url     | vcount
-----------------------------
1  | 2      | hacker story| some-url.net| 2
(1 row)
dev=>
dev=>
Streaming graph

CREATE TABLE stories (id int, author int, title text, url text);
CREATE TABLE votes (user int, story_id int);

CREATE MATERIALIZED VIEW StoriesWithVC AS
SELECT id, author, title, url, vcount
FROM stories
JOIN ( SELECT story_id, COUNT(*) AS vcount FROM votes GROUP BY story_id) as VoteCount
ON VoteCount.story_id = stories.id;
EXPLAIN

CREATE MATERIALIZED VIEW StoriesWithVC AS

SELECT id, author, title, url, vcount

FROM stories JOIN (SELECT story_id, COUNT(*) AS vcount FROM votes GROUP BY story_id) AS VoteCount
ON VoteCount.story_id = stories.id;

StreamMaterialize { columns: [id, author, title, url, vcount, ...] }

└─StreamExchange

  └─StreamHashJoin { type: Inner, predicate: stories.id = votes.story_id }

    └─StreamExchange { dist: HashShard(stories.id) }

    └─StreamTableScan { table: stories, columns: [id, author, title, url, _row_id] }

└─StreamHashAgg { group_key: [votes.story_id], aggs: [count] }

    └─StreamExchange { dist: HashShard(votes.story_id) }

    └─StreamTableScan { table: votes, columns: [story_id, _row_id] }
EXPLAIN CREATE MATERIALIZED VIEW StoriesWithVC AS

SELECT id, author, title, url, vcount
FROM stories JOIN (SELECT story_id, COUNT(*) AS vcount FROM votes GROUP BY story_id) AS VoteCount
ON VoteCount.story_id = stories.id;

StreamMaterialize { columns: [id, author, title, url, vcount, ...] }
└─StreamExchange
  └─StreamHashJoin { type: Inner, predicate: stories.id = votes.story_id }
    │─StreamExchange { dist: HashShard(stories.id) }
    |└─StreamTableScan { table: stories, columns: [id, author, title, url, _row_id] }
    └─StreamHashAgg { group_key: [votes.story_id], aggs: [count] }
      └─StreamExchange { dist: HashShard(votes.story_id) }
        └─StreamTableScan { table: votes, columns: [story_id, _row_id] }
Table: Votes

<table>
<thead>
<tr>
<th>user</th>
<th>story_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>1003</td>
</tr>
<tr>
<td>2345</td>
<td>1004</td>
</tr>
<tr>
<td>9999</td>
<td>1003</td>
</tr>
</tbody>
</table>

User story_id count

 Aggregate

story_id -> COUNT(*)

Join

MV

Table: Stories

<table>
<thead>
<tr>
<th>ID</th>
<th>author</th>
<th>title</th>
<th>url</th>
</tr>
</thead>
<tbody>
<tr>
<td>1003</td>
<td>Lu</td>
<td>foo</td>
<td>http...</td>
</tr>
<tr>
<td>1004</td>
<td>Jane</td>
<td>bar</td>
<td>http...</td>
</tr>
<tr>
<td>1005</td>
<td>John</td>
<td>buzz</td>
<td>http...</td>
</tr>
</tbody>
</table>
Table: Votes

<table>
<thead>
<tr>
<th>user</th>
<th>story_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>1003</td>
</tr>
<tr>
<td>2345</td>
<td>1004</td>
</tr>
<tr>
<td>9999</td>
<td>1003</td>
</tr>
</tbody>
</table>

Aggregate

```
<table>
<thead>
<tr>
<th>story_id</th>
<th>COUNT(*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1003</td>
<td>2</td>
</tr>
<tr>
<td>1004</td>
<td>1</td>
</tr>
</tbody>
</table>
```

Join

Table: Stories

<table>
<thead>
<tr>
<th>ID</th>
<th>author</th>
<th>title</th>
<th>url</th>
</tr>
</thead>
<tbody>
<tr>
<td>1003</td>
<td>Lu</td>
<td>foo</td>
<td>http...</td>
</tr>
<tr>
<td>1004</td>
<td>Jane</td>
<td>bar</td>
<td>http...</td>
</tr>
<tr>
<td>1005</td>
<td>John</td>
<td>buzz</td>
<td>http...</td>
</tr>
</tbody>
</table>

MV

VoteCount.story_id = stories.story_id
Table: Votes

<table>
<thead>
<tr>
<th>user</th>
<th>story_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>1003</td>
</tr>
<tr>
<td>2345</td>
<td>1004</td>
</tr>
<tr>
<td>9999</td>
<td>1003</td>
</tr>
<tr>
<td>789</td>
<td>1004</td>
</tr>
</tbody>
</table>

Aggregate

\[ \text{story_id} \rightarrow \text{COUNT(*)} \]

Join

VoteCount.story_id = Stories.story_id

Table: Stories

<table>
<thead>
<tr>
<th>ID</th>
<th>author</th>
<th>title</th>
<th>url</th>
</tr>
</thead>
<tbody>
<tr>
<td>1003</td>
<td>Lu</td>
<td>foo</td>
<td>http</td>
</tr>
<tr>
<td>1004</td>
<td>Jane</td>
<td>bar</td>
<td>http</td>
</tr>
<tr>
<td>1005</td>
<td>John</td>
<td>buzz</td>
<td>http</td>
</tr>
</tbody>
</table>

Table: Stories

<table>
<thead>
<tr>
<th>ID</th>
<th>author</th>
<th>title</th>
<th>url</th>
<th>vcount</th>
</tr>
</thead>
<tbody>
<tr>
<td>1003</td>
<td>Lu</td>
<td>foo</td>
<td>http</td>
<td>2</td>
</tr>
<tr>
<td>1004</td>
<td>Jane</td>
<td>bar</td>
<td>http</td>
<td>1</td>
</tr>
</tbody>
</table>

MV
Table: Votes

\[ + \{(789, 1004)\} \]

Table: Stories

Table: Stories

Join

MV

Table: Votes

\begin{tabular}{|c|c|}
\hline
user & story_id \\
\hline
1234 & 1003 \\
2345 & 1004 \\
9999 & 1003 \\
789 & 1004 \\
\hline
\end{tabular}

\begin{tabular}{|c|c|}
\hline
story_id & COUNT(\#) \\
\hline
1003 & 2 \\
1004 & 1 \\
\hline
\end{tabular}

\begin{tabular}{|c|c|}
\hline
ID & author \\
\hline
1003 & Lu \\
1004 & Jane \\
1005 & John \\
\hline
\end{tabular}

\begin{tabular}{|c|c|c|c|c|}
\hline
ID & author & title & url & vcount \\
\hline
1003 & Lu & foo & http... & 2 \\
1004 & Jane & bar & http... & 1 \\
1005 & John & buzz & http... & 1 \\
\hline
\end{tabular}
<table>
<thead>
<tr>
<th>user</th>
<th>story_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>1003</td>
</tr>
<tr>
<td>2345</td>
<td>1004</td>
</tr>
<tr>
<td>9999</td>
<td>1003</td>
</tr>
<tr>
<td>789</td>
<td>1004</td>
</tr>
</tbody>
</table>

Table: Votes

<table>
<thead>
<tr>
<th>story_id</th>
<th>COUNT((s))</th>
</tr>
</thead>
<tbody>
<tr>
<td>1003</td>
<td>2</td>
</tr>
<tr>
<td>1004</td>
<td>2</td>
</tr>
</tbody>
</table>

Aggregate

<table>
<thead>
<tr>
<th>story_id</th>
<th>COUNT((s))</th>
</tr>
</thead>
<tbody>
<tr>
<td>1003</td>
<td>2</td>
</tr>
<tr>
<td>1004</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>author</th>
<th>title</th>
<th>url</th>
</tr>
</thead>
<tbody>
<tr>
<td>1003</td>
<td>Lu</td>
<td>foo</td>
<td>http...</td>
</tr>
<tr>
<td>1004</td>
<td>Jane</td>
<td>bar</td>
<td>http...</td>
</tr>
<tr>
<td>1005</td>
<td>John</td>
<td>buzz</td>
<td>http...</td>
</tr>
</tbody>
</table>

Table: Stories

<table>
<thead>
<tr>
<th>ID</th>
<th>author</th>
<th>title</th>
<th>url</th>
<th>vcount</th>
</tr>
</thead>
<tbody>
<tr>
<td>1003</td>
<td>Lu</td>
<td>foo</td>
<td>http...</td>
<td>2</td>
</tr>
<tr>
<td>1004</td>
<td>Jane</td>
<td>bar</td>
<td>http...</td>
<td>1</td>
</tr>
</tbody>
</table>

Join

<table>
<thead>
<tr>
<th>ID</th>
<th>author</th>
<th>title</th>
<th>url</th>
</tr>
</thead>
<tbody>
<tr>
<td>1003</td>
<td>Lu</td>
<td>foo</td>
<td>http...</td>
</tr>
<tr>
<td>1004</td>
<td>Jane</td>
<td>bar</td>
<td>http...</td>
</tr>
</tbody>
</table>

MV
Distributed systems
Challenges and opportunities in a distributed setup

- Opportunity:
  - Execute in parallel

- Challenges:
  - Recovery: Trying not to lose data when a node crashes
  - Scalability: Adding/removing nodes if you have more/less workloads
Distributed systems
Parallelism
Table: Votes

Aggregate

story_id -> COUNT(*)

Join

VoteCount.story_id = stories.story_id

Table: Stories

MV
message BarrierMutation {
  oneof mutation {
    StopMutation stop = 4;
    // Update outputs and hash mappings for some dispatchers, used for scaling.
    UpdateMutation update = 5;
    // Pause the dataflow of the whole streaming graph, only used for scaling.
    PauseMutation pause = 7;
    // Resume the dataflow of the whole streaming graph, only used for scaling.
    ResumeMutation resume = 8;
  }
}

message Barrier {
  enum BarrierKind {
    BARRIER_KIND_UNSPECIFIED = 0;
    // The first barrier after a fresh start or recovery.
    BARRIER_KIND_INITIAL = 1;
    // A normal barrier. Data should be flushed locally.
    BARRIER_KIND_BARRIER = 2;
    // A checkpoint barrier. Data should be synchronized to the shared storage.
    BARRIER_KIND_CHECKPOINT = 3;
  }
  BarrierMutation mutation = 3;
  data, Epoch epoch = 1;
  //...
Distributed systems
Recovery
Processing
Time: 0

Event flow direction
Event flow direction

State B

Disk

State M

Processing Time: 1
Event flow direction

State C

Processing
Time: 3

State B

Disk

State N

OPI

OP2
Event flow direction

OPI

State C

State B
State N

Disk

OP2

State N

Persist

Processing
Time: 4
Event flow direction

OPI
State C

State B
State N
Disk

OP2
State O

Processing
Time: 5
Distributed systems
Scaling
Thank you!
Try RisingWave:

Ask questions: