Fault-Tolerance in the Borealis Distributed Stream Processing System

Weihan Wang
November 23, 2005

About this paper

- Magdalena Balazinska, MIT
- Hari Balakrishnan, MIT
- Samuel Madden, MIT
- Michael Stonebraker, MIT
- In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2005.

Agenda

- Background
- System overview
- Upstream failure
- Stabilization
- Evaluation
Agenda

- Background
- System overview
- Upstream failure
- Stabilization
- Evaluation

A piece of history

- Monolithic SPE
- Distributed SPE
- Single adm. domain

- Distributed SPE
- Multiple adm. domains (aka Federation)

Result revision in Borealis

- It's raining
- It's raining

Today is rainy Today is rainy
Result revision in Borealis

- Error fixing from data source
- Load shedding
- Time-travel into the past or future
- Fault tolerance

Common solution for replication-based FT

Common solution for replication-based FT
Comparison with [Hwang05]

- They don’t distinguish between HA & FT.
- They are parallel to each other.
- Compared to [Hwang05]:
  - Approach of this paper is similar to [Hwang05]'s active standby.
  - This paper uses result revision.
  - This paper addresses network failures.
  - This paper avoids inter-replica communications.
  - ....

Agenda

- Background
- System overview
- Upstream failure
- Stabilization
- Evaluation

Design goal

- User’s preference:

<table>
<thead>
<tr>
<th>During failure</th>
<th>After failure</th>
<th>User</th>
</tr>
</thead>
<tbody>
<tr>
<td>No outputs</td>
<td>Correct outputs</td>
<td>😊</td>
</tr>
<tr>
<td>Approximation</td>
<td>-</td>
<td>😕</td>
</tr>
<tr>
<td>Approximation</td>
<td>Error correction</td>
<td>😊😊</td>
</tr>
</tbody>
</table>


Design goal

- Goal: to minimize the number of approximated outputs during failure, subject to a delay constraint, and to revise them after failure.
- For each nodes, the user-defined delay constraint is $X$, and data processing time is $(1-\alpha)X$. So we can hold input tuples up to $\alpha X$ sec.

Data model and node states

- Tuple format: $(type, id, time, a_1, ..., a_m)$
  - 1: STABLE tuples
  - 2: TENTATIVE tuples
  - 3: UNDO tuples
  - 4: BOUNDARY tuples
Node states

- **STABLE**
- **UP_FAILURE**
- **STABILIZATION**

- Missing heartbeats or tentative tuples
- Stabilized
- Upstream healed
- Another upstream failure

Agenda

- Background
- System overview
- **Upstream failure**
- Stabilization
- Evaluation

Scenario 1

Every node monitors all upstream nodes and does stream processing simultaneously.
Scenario 1

Scenario 2

Issues of upstream switching

- All replicas must have consistent outputs.
- No inter- replica communication.
- Solution
  - Use deterministic operators
  - Use $SUnion$ & boundary tuples to sort inputs from multiple streams deterministically.
If all boundaries arrive in time, SUnion sorts & forwards the whole bucket as STABLE tuples.
If boundaries don’t arrive in time, or there are TENTATIVE tuples, SUnion stores & forwards the bucket as TENTATIVE tuples.

Scenario 3
Scenario 3

S

X

A

B

C

U

A

B

C

X

A

B

C

S

X

A

B

C

U

A

B

C

S

X

A

B

C

U

A

B

C
Agenda

- Background
- System overview
- Upstream failure
- **Stabilization**
- Evaluation

Stabilization

- State reconciliation
  - Checkpoint / redo
  - Undo / redo
  - How to satisfy delay constraint if stabilization takes long?
- Output stabilization
- Failed node recovery

State reconciliation: Checkpoint / Redo

![Diagram showing state reconciliation process]
State reconciliation: Checkpoint / Redo

Input

Checkpoint

Query net

Snapshot

State reconciliation: Checkpoint / Redo

Input

Checkpoint

Query net

Snapshot

State reconciliation: Undo / Redo

- The stream markers of tuple \( p \) identify the oldest tuples on each input stream that still contribute to the operator's state when the operator processes \( p \).

- To undo all tuples after \( p \), reset the operator and restart from the markers of \( p \).
Processing new tuples during reconciliation
- A node suspends its outputs for state reconciliation. But it may take longer than $X$.
- Solution:
  - The node requests another replica to postpone its own reconciliation.
  - The downstream nodes turn to that replica for TENTATIVE outputs.
  - They switch back to the original node when reconciliation done.

Agenda
- Background
- System overview
- Upstream failure
- Stabilization
- Evaluation

Evaluation setup
- Single-node evaluation
- Multiple-node evaluation
Evaluation results

- The best approach is to process new tuples without delay in both UP_FAILURE and STABILIZATION states.
- Checkpoint/redo is better than undo/redo.
- Memory overhead is proportional to:
  - \# of SUion
  - SUion's bucket sizes
  - SUion's input rates

Conclusion

- The approach favors availability but guarantees eventual consistency.
- It uses result revision to achieve final consistency.
- It uses SUion to synchronize replicas without inter-replica communication.
- Checkpoint/redo and undo/redo are used for state reconciliation.

Discussion

- Long failures may cause output/input buffers overrun.
- No enough explanation on output buffer truncation strategies.
- No enough explanation on relationship between boundary tuples and SUion bucket size.
- How to recover failed node with divergent operators?
- No evaluations on failed node recovery and replica switching during reconciliation.
References


Backup slides

Output stabilization

- Every node shall propagate UNDO tuples during stabilization.
- Checkpoint/redo nodes use SOutput operators to help produce UNDO tuples.
Query network trees

- Stable tree
- Tentative tree
- Blocking tree
- Not valid

Implementation

- Modified query network

Operator / wrapper interface

- For checkpoint / redo
  - packState()
  - unpackState()
- For undo / redo
  - clear()
  - findOldestTuple(int stream_id)
- For boundary tuple
  - findOldestTimestamp()