Rollback-Recovery without Checkpoints in Distributed Event Processing Systems

Boris Koldehofe, Ruben Mayer, Umakishore Ramachandran, Kurt Rothermel, Marco Völz

DEBS‘13, 2013-07-01
Outline

- Motivation and contribution
- Savepoint recovery approach
- Evaluation
- Summary and outlook
Motivation

- Distributed Complex Event Processing
  - Many operators processing event streams
- Node and communication failures
  - Loss of operator state
  - Events arrive late
- Event streams must still be reliable
  - No false-negatives
  - No false-positives

Diagram:
- Manufacturer
- Billing
- Customer Information
- Source events
- Delivery of a package of 3 artifacts for 300 $
- Delivery of a package of 2 artifacts for 250 $
- false-negative
- false-positive
But Isn’t Reliability Already Solved?

- State-of-the-Art methods provide strong reliability
  - But they induce too much run-time overhead
  - Replication → redundant processing
  - Rollback-Recovery → checkpointing and logs
- In many large-scale CEP systems...
  - High throughput is important
    - Low run-time overhead is crucial!
  - Recovery only required in magnitudes of minutes
  - Failures rarely happen (→ normal case: no failures)
    - Moderate recovery time and recovery costs are acceptable

Our goal is a recovery scheme with low run-time overhead
General Operator Model

- **All** operators $\omega$: Correlation of events is performed in steps
  - Selection of events $\sigma$ from incoming streams gets correlated
  - A set of events $(e_1,...,e_n)$ is deducted from that selection
    - Correlation function $f_\omega: \sigma \rightarrow (e_1,...,e_n)$ describes a correlation step

- **Key observation**: Processing of a selection is independent from processing of other selections
  - Correlation function itself is stateless
Our Contribution

• A rollback-recovery method that significantly reduces run-time overhead
  ◦ Works without persistent checkpoints
  ◦ Utilizes a simple and expressive execution model
    ▪ Allows simple integration of state-of-the-art CEP operators
Approach Idea

- Recovery of
  - Incoming streams
  - Current selection on them
- No internal state of the correlation function $f_\omega$ needs to be recovered
- Incoming streams can be re-streamed from predecessors
- Information on current selection needs to be captured
  - Execution model $\rightarrow$ operator reveals selection information
Execution Model

- Marker defines start event of a selection
- Operator reveals that marker when a new correlation step starts

- Requirement: $M_{n+1} > M_n$
  - Marker always shifts to the future
- Marker shift is signalled by function call
- Expressiveness: All \textit{snoop} expressions can be mapped
  - Snoop event operators
  - Parameter contexts
**Execution Model**

- Marker defines start event of a selection
- Operator reveals that marker when a new correlation step starts

**Requirement:** $M_{n+1} > M_n$
  - Marker always shifts to the future
- Marker shift is signalled by function call
- Expressiveness: All **snoop** expressions can be mapped
  - Snoop event operators
  - Parameter contexts
Savepoint Recovery System

- Current selection marker is tracked
  - Stored locally with produced events in “outlog”
  - Given incoming events > M and marker M,
    - Operator can be recovered to produce same sequence of events $(e_1,\ldots,e_n)$

\[
\begin{align*}
M & \quad \sigma \\
\omega_p & \quad \text{incoming events} \\
\omega & \quad f_\omega \\
\omega_s & \quad \text{outgoing events} \\
\end{align*}
\]

<table>
<thead>
<tr>
<th>event</th>
<th>Marker</th>
</tr>
</thead>
<tbody>
<tr>
<td>$e_1$</td>
<td>$M$</td>
</tr>
<tr>
<td>...</td>
<td>$M$</td>
</tr>
<tr>
<td>$e_n$</td>
<td>$M$</td>
</tr>
</tbody>
</table>
Acknowledgement Algorithm (1)

- Markers are reliably replicated at predecessor operators
  - We call them **savepoints**
  - Savepoints contain a sequence number for each incoming stream
- But which savepoint is relevant for recovery?
  - ACK of event \( \rightarrow \) not need to be re-streamed in the future any more
  - Also all earlier events can be discarded from outlog

\[\begin{align*}
\omega_p & \quad \text{incoming events} \\
\omega & \quad \text{outgoing events} \\
\omega_s & \quad \text{outlog } \omega:
\end{align*}\]

<table>
<thead>
<tr>
<th>event</th>
<th>Marker</th>
</tr>
</thead>
<tbody>
<tr>
<td>(e_1)</td>
<td>M</td>
</tr>
<tr>
<td>...</td>
<td>M</td>
</tr>
<tr>
<td>(e_n)</td>
<td>M</td>
</tr>
</tbody>
</table>

ACK(\(e_n\))
Acknowledgement Algorithm (2)

- When recovering $\omega$, ACKed events do not need to be reproduced
  - Events earlier than $M$ will not need to be re-streamed in the future
  - They are ACKed at the predecessors of $\omega$
- ACKs flow upstream, until they reach the basic event sources
  - Sources need reliable implementation
- Piggybacking savepoints at ACKs for replication
Acknowledgement Algorithm (3)

- To survive multiple simultaneous operator failures, a savepoint is replicated several times
  - At predecessors’ predecessors, and so on...
- Each operator stores several savepoints
  - A savepoint tree

\[ \omega \]
Recovery From Multiple Failures

- Coordinator with weak failure detection
- Suspect operator
  → init replacement
- Recovery Request
- Stream events and savepoint
  ◦ Operator recovers
- Recovery Notification
- Parallel execution of operator and replacement
- Progress condition: Savepoint update
- Recall of operator
  ◦ Just in case that suspicion was wrong
Evaluation: Setup

• Simulation: Omnet++

• Goals
  ◦ Evaluate inherent run-time overhead of the recovery scheme
    ▪ Communication costs for ACKs / savepoint transmissions
    ▪ Memory footprint for log of events at sources
  ◦ Evaluate the influence of ACK frequency

• Setting: Parametrized tree topology
  ◦ Tree depth
  ◦ Node degree
Evaluation: Run-time overhead

- Tree depth was 3
- Comparison of communication overhead to active replication
  - And we don't have redundant processing
Evaluation: ACK frequency

- Depth = 3, degree = 3
- Communication overhead can be traded against memory footprint
Summary and Outlook

• Our contribution is a novel rollback-recovery method
  ◦ It works without heavy checkpoints
    ▪ Instead, we use lightweight savepoints
  ◦ It works without persistent storage
    ▪ Instead, we use predecessors for replication of savepoints
    ▪ Events get re-streamed and reproduced
  ◦ Execution model builds the interface to savepoint recovery system
    ▪ Paper shows expressiveness of the model (→ snoop)

• We have shown that only little run-time overhead is induced

• Outlook: Improve shortcomings
  ◦ Limit recovery time (→ real-time requirements)
End of Presentation

Thank you for your attention. It’s now time for questions and discussions.

This work has been carried out in the context of the “CEP in the Large” project. CEPiL is a joint project of the University of Stuttgart and Georgia Tech that is funded by the BW foundation. http://www.ipvs.uni-stuttgart.de/abteilungen/vs/forschung/projekte/CEP-in-the-Large?__locale=en

Advertisement: Other interesting talks of our project group at the DEBS’13

Efficient Content-based Routing with Network Topology Inference
MigCEP: Operator Migration for Mobility Driven Distributed Complex Event Processing
Opportunistic Spatio-temporal Event Processing for Mobile Situation Awareness