S-Store: Streaming meets Transaction Processing

Nesime Tatbul (Intel Labs & MIT)  

joint work with

John Meehan, Stan Zdonik, Cansu Aslantas, Uğur Çetintemel, Tim Kraska (Brown)
Mike Stonebraker, Sam Madden, Hao Wang (MIT)
Kristin Tufte, Dave Maier (PSU)
Andy Pavlo (CMU)
ISTC for Big Data

• One of Intel’s 4 current Science and Technology Centers in the US (+6 similar ones world-wide)
• MIT as main hub + 8 other universities
• Launched in 2012, 3+2 years of funding
• Research themes:
  – Data analytics & processing platforms
  – Scalable math & algorithms
  – Visualization
  – Architecture
  – Benchmarks & testbeds
  – **Integration across multiple data processing systems**
S-Store: BigDAWG’s Streaming Data Store

- Reliable, real-time ingest of streaming data
- In-memory processing of all streaming analytics workloads
- Support for transactional state management and relational OLTP workloads
- Real-time ETL of new data into other BigDAWG stores
- Critical enabler for joining current data with past data
The Big Velocity Challenge

• Data is coming too fast!
  – Sensors, Smart phones, Internet of Things, Web clicks, Stock tickers, Social media feeds, News feeds, ...

• Applications need:
  – scalable data ingest, processing, and storage
  – real-time, complex data analytics
  – high-throughput, transactional processing
  – data-driven, continuous, incremental processing models
State of the Art & Recent Trends

• Stream processing
  – in-memory, low-latency processing
  – fine-grained batching of inputs, complex dataflow computations
  – scalability and fault-tolerance over large clusters

What about streams + transactions?

• Transaction processing
  – disk-based OLTP -> main-memory OLTP
  – multi-core, shared-nothing clusters
  – NewSQL architectures (scalable SQL and ACID)
Shared Mutable State in Streaming
A Real-World Example: Financial Order Routing

Q: Streaming or OLTP?
A: Both!

[Source: StreamBase, Inc.]
S-Store in a Nutshell

• A single system for transaction & stream processing

• A novel computational model for supporting hybrid workloads with well-defined correctness guarantees
  – ACID guarantees for individual transactions (OLTP + streaming)
  – ordered execution guarantees for dataflow graphs of streaming transactions
  – exactly-once processing guarantees for streams (no loss or duplication)

• A flexible and expressive programming interface
  – transactions as user-defined stored procedures (Java) w/ SQL-based data access
  – support for dataflow graphs and nested transactions

• Scalable software architecture and implementation
  – distributed main-memory OLTP system as foundation (H-Store)
  – clean and general architectural extensions (e.g., triggers, windowing)
Hybrid Computational Model

**Batch-id’s** are used to track lineage and order

**Dataflow Graphs** of Streaming Transactions

Push outputs to an external sink (or store in a Table)

Stream as a sequence of Atomic Batches

Each batch leads to a Transaction Execution (TE)

<table>
<thead>
<tr>
<th>Stream s₁</th>
<th>Window w₁</th>
<th>Stream s₂</th>
<th>Table for s₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>T₁₁(s₁.b₁,w₁)</td>
<td>T₁₂(s₁.b₂,w₁)</td>
<td>T₂₁(s₂.b₁)</td>
<td>alternative for output s₃</td>
</tr>
</tbody>
</table>

Three kinds of state: **Streams, Windows, and Tables**
- All physically kept as in-memory tables
- Tables can be publicly shared among all transactions (**OLTP or Streaming**) 
- Streams & Windows are not publicly shareable

Nested Transactions for coarse-grained isolation
Example Uses for Nested Transactions

**Use 1:** To protect parts of a dataflow graph from other OLTP or Streaming transactions

**Use 2:** To protect one instance of a dataflow graph from its subsequent instances (e.g., Leaderboard Benchmark)
Triple Correctness Guarantees

- **ACID** from traditional OLTP
  - Failure recovery (Atomicity and Durability)
  - Concurrency control (Consistency and Isolation)

- **Ordered execution** from Streaming
  - Atomic batches of a stream must be processed in order (stream order constraint)
  - For a given atomic batch, transactions in a dataflow graph must be processed in topological order (dataflow order constraint)
  - Nested transactions require strict serial ordering

- **Exactly-once processing** from Streaming
  - Recovering from failures (i.e., replay of streams) should not cause lost or duplicated data

> S-Store provides efficient scheduling and recovery mechanisms to ensure these guarantees.
H-Store as System Foundation

- main-memory OLTP system developed at Brown & MIT
- base design for the VoltDB NewSQL database system
- programming model: stored procedures (Java + SQL)
- database partitioned across multiple sites in a way to minimize the number of distributed transactions
- single-threaded transaction execution per partition
- recovery via checkpointing + command-logging
- anti-caching to disk if all data does not fit in memory

\[ \text{S-Store} = \text{H-Store} + \text{Streaming} \]
S-Store’s Extended Architecture

- **Client**
  - S-Store Engine
    - Stored Procedure (Java)
      - Query (SQL)
      - Query (SQL)
      - Query (SQL)
    - Stored Procedure (Java)
      - Query (SQL)
      - Query (SQL)
      - Query (SQL)
  - Partition Engine (PE)
  - Execution Engine (EE)

- **Stream Ingestion Module**
- **In-memory Partition Data**
  - Tables
  - Windows
  - Streams

- **Additional Functions**
  - Transaction management
  - Query planning
  - Statistics management
  - Input management
  - Dataflow graph management
  - PE triggers
  - Storage management
  - Query processing
  - Window management
  - EE triggers

- **Participating Institutions**
  - Portland State University
  - MIT
  - Brown University
  - Carnegie Mellon University
S-Store vs. H-Store: EE Triggers
S-Store vs. H-Store: EE Triggers

Max Throughput (batches/sec) vs. Number of EE Triggers

- S-Store
- H-Store
S-Store vs. H-Store: PE Triggers

- PE trigger (S-Store)
- Client-PE round-trip (H-Store)
- Client-PE round-trip (both)
S-Store vs. H-Store: PE Triggers

Max Throughput (batches/sec)

Number of PE Triggers

S-Store
H-Store
Fault Tolerance in S-Store
Check-pointing + Command-logging + Upstream backup

• Periodic check-pointing of in-memory tables to disk
• Strong recovery
  – Log all committed transactions (OLTP + streaming)
  – Upon failure, log replay reproduces the exact pre-failure state
  – To avoid redundancy, must turn off triggers during recovery
• Weak recovery
  – Log transactions selectively (all OLTP + “border” streaming)
  – Upon failure, log replay may lead to a different, but correct state
  – No need to turn off triggers
• Upstream backup for streaming inputs that have not yet been accounted for in downstream logs
Weak Recovery vs. Strong Recovery

Max. Throughput (batches/sec)

Recovery Time (sec)

per 5000 input batches
## S-Store vs. State of the Art

Better Correctness Guarantees & Better Performance

<table>
<thead>
<tr>
<th>System</th>
<th>ACID</th>
<th>Order</th>
<th>Exactly-Once</th>
<th>Max Tput (batches/sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>H-Store (async)</td>
<td>✓</td>
<td>×</td>
<td>×</td>
<td>5300</td>
</tr>
<tr>
<td>H-Store (sync)</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
<td>210</td>
</tr>
<tr>
<td>Esper+ VoltDB</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
<td>570</td>
</tr>
<tr>
<td>Storm+ VoltDB</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>600</td>
</tr>
<tr>
<td>S-Store</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>2200</td>
</tr>
</tbody>
</table>

Leaderboard Benchmark on a single-node Intel® Xeon® E7-4830 at 2.13 GHz

~ 10 x OLTP  
~ 4 x SPE

Current Work in Progress
Scaling to Multiple Nodes

• Three basic primitives to partition a streaming workload:
  – **Move**: Move a stream from one node to another (distributed transaction)
  – **Demux**: Split a stream into multiple partitions
  – **Mux**: Merge multiple streams into one

• Both pipelined (Move) & partitioned parallelism (Demux+Move)

• Research question #1: Given a dataflow graph and a set of processing nodes, where to place Move/Demux/Mux + how to partition public Tables in order to maximize performance and load balance?

• Research question #2: How to ensure correct and efficient scheduling and recovery at all nodes?
Future Directions

• Extend our support for streaming analytics
• Tighter integration with BigDAWG (e.g., optimizing cross-system workloads)
• Hardware-aware S-Store (NVM, many-core, fast networks)
• Handling mixed and dynamic workloads
• Building novel and challenging use cases
S-Store in Action
The MIMIC Demo
S-Store in Action
The MIMIC Demo

S-Store
S-Store in Action
The Canadian Dreamboat Demo
S-Store in Action
The Canadian Dreamboat Demo

---

### S-Store

<table>
<thead>
<tr>
<th>#</th>
<th>Top 3</th>
<th>Votes</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>#</th>
<th>Bottom 3</th>
<th>Votes</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>#</th>
<th>Trending 3</th>
<th>Votes</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### H-Store

<table>
<thead>
<tr>
<th>#</th>
<th>Top 3</th>
<th>Votes</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>#</th>
<th>Bottom 3</th>
<th>Votes</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>#</th>
<th>Trending 3</th>
<th>Votes</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

Votes until next delete: 0
Invalid Votes: 0

---

Intel | MIT | Brown | Carnegie Mellon University | W | Oregon State University | Portland State University
S-Store in Action
The BikeShare Demo

BikeStatus Stream

NearByStations read
NearByDiscounts read

Stations read
write
NearByStations
write
StationStatus

read/write
CheckOutBike
CheckInBike
Accept-Discount

Rider Status Stream
write

Bikeshare App & Workload Generator
S-Store in Action
The BikeShare Demo