Sub-millisecond Stateful Stream Querying over Fast-evolving Linked Data

Yunhao Zhang, Rong Chen, Haibo Chen

Institute of Parallel and Distributed Systems (IPADS)
Shanghai Jiao Tong University
Stream Query is Important

Multiple data sources are continuously generating **streaming data in high velocity**
A stateful stream query needs to integrate streaming data with stored data.
Stateful Stream Query

A stateful stream query needs to integrate streaming data with stored data.

Streaming Data:
high velocity

Stored Data:
large & evolving
Example Dataset for Stateful Stream Query

Great experience at SOSP'17! 😊

<Rong, creates, Feed>. 12:30
<Feed, hash_tag, SOSP>. 12:30
<Yunhao, likes, Feed>. 12:31
<Haibo, likes, Feed>. 12:40

Streaming Data

Stored Data
Connectivity Property of Data

Linked data represents information as entities and relations between the entities.

Stored Data
- Haibo
  member_of
  IPADS
- Yunhao
  member_of
  Cornell

Streaming Data
- Rong
  member_of
  IPADS
- Yunhao
- Haibo
  like
  Feed
  #SOSP#
  create
Example Continuous Query

In the last 30 minutes, which IPADS members created feeds that are liked by other IPADS members?
Example Continuous Query

In the last 30 minutes, which IPADS members created feeds that are liked by other IPADS members?
Workload Characteristics

Connectivity property

Stateful queries integrate Stored and Streaming Data

Stored data evolves by absorbing streaming data
Conventional Approach

Streaming Data

Stream Processing System

Continuous Query

Stored Data

Graph Store System

One-shot Query
Composite Design

**Example**

- **Apache Storm**
  - Stream Processing System
- **Wukong**
  - Graph Store System

**Composite Design**

- Streaming Data
- Continuous Query
- Graph Store System
- One-shot Query

**Composite Design Example**
Composite Design Observations

1. Cross-system Cost

~40% execution time wasted due to data transformation and transmission

2. Inefficient Query Plan

Semantic gap between the two systems impair query optimization

3. Limited Scalability

Stream processing systems dedicate all resources to the improve performance of a single job
Composite Design Observations

1. Cross-system Cost

~40% execution time wasted due to data transformation and transmission

2. Inefficient Query Plan

Semantic gap between the two systems impairs query optimization

3. Limited Scalability

Stream processing systems dedicate all resources to improve performance of a single job

Composite Design:
high latency
low throughput
Wukong+S uses a novel integrated design for stateful stream query over fast-evolving linked data.

Integrated Design manages streaming data and stored data in a single system:
- Eliminate cross-system cost
- Global semantics for query optimization
- Better scalability by sharing data between the queries
Implementing integrated design is not trivial

Decisions for efficient integrated design:

► **Hybrid Store:** efficiently handle streaming data and fast-evolving stored data
► **Stream Index:** fast path to access streaming data in a certain time interval
► **Consistent Data View:** through decentralized vector timestamps and bounded snapshot scalarization
Data is **partitioned and stored** on multiple servers.
Hybrid Store

**How to gracefully integrate streaming data and stored data?**

**Strawman:** using different graph stores according to “where from”, namely *streaming* and *stored* data

**Hybrid Store:** using different graph stores according to “how to use”, namely *timeless* and *timed* data
Explicit Separation of Streaming Data

Streaming Data

user-defined predicate

Timed

Continuous Query

One-shot Query

Timeless

Stored Data
Benefit of Hybrid Store

No interference between timeless data and timed data

Design data stores separately and optimize for different operation pattern:

- **Timeless Data**: continuous persistent store
- **Timed Data**: time-based transient store
Hybrid Store

Continuous Persistent Store

- Continuously absorb the **timeless** portion of streams
- **Goal**: support **stateful** continuous query and **up-to-date** one-shot query
Hybrid Store

**Time-based Transient Store**

- Timed data will only be accessed by relevant continuous queries in a time interval.
- **Goal**: support fast garbage collection (GC) for the timed portion of streams.
Consistent Data Snapshot

How to provide consistent view over dynamic data with memory efficiency?

- Streaming data contains *order* information
- Early output from a stream source should always be *visible before* later output
- No *order* relation across data sources
Decentralized Vector Timestamp (VTS)

Data is partitioned and stored on multiple servers.
Snapshot Scalarization

One-shot Query

SN: Snapshot Number
VTS: Vector Timestamp

SN-VTS Plan

Visible snapshot

SN=2: [4, 10]
SN=3: [5, 12]
SN=4: [7, 14]
Benefit of Snapshot Scalarization

- Memory Efficiency
  - bound number of visible snapshot
- Injection Speed
  - decouple data sources from underlying store

Staleness of Stored Data

- control staleness by SN_VTS Plan

stream query scenario
Other Designs of Wukong+S

- Stream index & locality-aware partitioning
- Data-driven query trigger
- One-shot query execution
- Fault tolerance
- Leveraging RDMA
Evaluation

Baseline: 6 state-of-the-art systems
- CSPARQL-engine, Heron+Wukong, Storm+Wukong
- Spark Streaming, Spark Structured Streaming, Wukong/ext

Platforms: a rack-scale 8-machine cluster
- Each: two 12-core Intel Xeon, 128GB DRAM, w/ RDMA Mellanox 56Gbps InfiniBand NIC, 40Gbps IB Switch

Benchmarks:
- LSBench: Social Networking Benchmark w/ 3.75B initial stored data & 5 streams totally 134K tuple/second stream
- CityBench: Smart City Benchmark w/ 11 real-world data streams
Single Query Latency

Outperform: state-of-the-art systems

- Wukong+S: sub-millisecond
- 13.7X speedup vs. Storm+Wukong
- 3 orders of magnitude speedup vs. Spark Streaming

![Graph showing latency comparison between Wukong+S, Storm+Wukong, and Spark Streaming across different load levels (L1 to L6). The graph indicates significantly lower latency for Wukong+S compared to the other systems.]
Single Query Latency

Unavoidable reason for high latency

- Cross-system Cost for Storm+Wukong
- Joining large stored data (3.75B) for Spark Streaming
Throughput of Mixed Workloads

- **Wukong+S**: ~1M queries/second on 8 nodes
- **Mixture of 3 queries**: 1.08M queries/second
- **Add complex queries**: 802K queries/second

![Graphs showing throughput vs. number of machines for different workloads.](image-url)
Other Evaluations

- Influence of different stream rate
- Data insertion latency
- Performance of one-shot queries
- Memory consumption
- Fault-tolerance overhead
Conclusion

Existing systems cannot satisfy demands of stateful stream query over fast-evolving linked data

**Wukong+S**: A distributed stream querying engine adopting a novel integrated design for stateful stream queries over fast-evolving linked data

Achieving sub-millisecond latency and throughput exceeding one million queries per second

http://ipads.se.sjtu.edu.cn/projects/wukong
Thanks

Wukong+S

http://ipads.se.sjtu.edu.cn/projects/wukong
Institute of Parallel and Distributed Systems

Questions
## Table 5: The performance impact of RDMA on Wukong+S.

<table>
<thead>
<tr>
<th>LSBench</th>
<th>L1</th>
<th>L2</th>
<th>L3</th>
<th>L4</th>
<th>L5</th>
<th>L6</th>
<th>Geo.M</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wukong+S</td>
<td>0.10</td>
<td>0.08</td>
<td>0.11</td>
<td>1.78</td>
<td>3.50</td>
<td>1.68</td>
<td>0.46</td>
</tr>
<tr>
<td>Non-RDMA</td>
<td>0.11</td>
<td>0.08</td>
<td>0.12</td>
<td>6.22</td>
<td>6.14</td>
<td>4.90</td>
<td>0.75</td>
</tr>
<tr>
<td>Slowdown</td>
<td>1.1X</td>
<td>1.0X</td>
<td>1.1X</td>
<td>3.5X</td>
<td>1.8X</td>
<td>2.9X</td>
<td>1.6X</td>
</tr>
</tbody>
</table>