CS 744: BIG DATA SYSTEMS

Shivaram Venkataraman
Fall 2018
- Assignment 2 grades
- Midterm review session on Nov 2 at 5pm at 1221 CS
- Course Project Proposal feedback
STREAM PROCESSING
Sales Dashboard

Total Sales: $3,256.8M
Number of Deals: 17,164
Avg Deal Size: $189,545
Rev. per Salesperson: $20.5M

Week of Date Closed:
- December 6, 200 - December 25, 200

Region
- (All)

Country
- (All)

Sales Team
- (All)
- Small and Midmarket
- Enterprise

Avg Deal Size/Salesperson
- $147,043
- $336,519

Week of September 4, 2016
- Revenue: $14.6M
- Running Sum of Revenue: $798.4M
REAL-TIME ANALYSIS
STREAMING + ITERATIVE COMPUTATION

User queries are received

Low-latency query responses are delivered

Queries are joined with processed data

Updates to data arrive

Complex processing incrementally re-executes to reflect changed data
TIMELY DATAFLOW

Timestamp: \( (e \in \mathbb{N}, \langle c_1, \ldots, c_k \rangle \in \mathbb{N}^k) \)

<table>
<thead>
<tr>
<th>Vertex</th>
<th>Input timestamp</th>
<th>Output timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ingress</td>
<td>( (e, \langle c_1, \ldots, c_k \rangle) )</td>
<td>( (e, \langle c_1, \ldots, c_k, 0 \rangle) )</td>
</tr>
<tr>
<td>Egress</td>
<td>( (e, \langle c_1, \ldots, c_k, c_{k+1} \rangle) )</td>
<td>( (e, \langle c_1, \ldots, c_k \rangle) )</td>
</tr>
<tr>
<td>Feedback</td>
<td>( (e, \langle c_1, \ldots, c_k \rangle) )</td>
<td>( (e, \langle c_1, \ldots, c_k + 1 \rangle) )</td>
</tr>
</tbody>
</table>
Receiving Messages

\( v.\text{OnRecv}(e : \text{Edge}, m : \text{Msg}, t : \text{Time}) \)

\( v.\text{OnNotify}(t : \text{Timestamp}) \)

Sending Messages

\( \text{this.SendBy}(e : \text{Edge}, m : \text{Msg}, t : \text{Time}) \)

\( \text{this.NotifyAt}(t : \text{Timestamp}) \)

**Conditions**

OnNotify\( (t) \) invoked after all OnRecv\( (e, r, t') \) for all \( t' \leq t \)

SendBy or NotifyAt only called with \( t' \geq t \)
IMPLEMENTING TIMELY DATAFLOW

Need to track when it is safe to notify

Path Summary
Check if \((t_1, l_1)\) could-result-in \((t_2, l_2)\)

Scheduler

Occurrence and Precursor count
Precursor count = 0 \(\rightarrow\) Frontier

<table>
<thead>
<tr>
<th>Operation</th>
<th>Update</th>
</tr>
</thead>
<tbody>
<tr>
<td>v.SENDBY(e, m, t)</td>
<td>(OC[(t, e)] \leftarrow OC[(t, e)] + 1)</td>
</tr>
<tr>
<td>v.ONRECV(e, m, t)</td>
<td>(OC[(t, e)] \leftarrow OC[(t, e)] - 1)</td>
</tr>
<tr>
<td>v.NONOTIFYAT(t)</td>
<td>(OC[(t, v)] \leftarrow OC[(t, v)] + 1)</td>
</tr>
<tr>
<td>v.ONNOTIFY(t)</td>
<td>(OC[(t, v)] \leftarrow OC[(t, v)] - 1)</td>
</tr>
</tbody>
</table>
Workers communicate using Shared Queue

Batch messages delivered
Account for cycles

Vertex single threaded
DISTRIBUTED PROGRESS TRACKING

Broadcast-based approach

Maintain local precursor count, occurrence count
Send progress update \((p \in \text{Pointstamp}, \delta \in \mathbb{Z})\)
Local frontier tracks global frontier

Optimizations

Batch updates and broadcast
Use projected timestamps from logical graph
FAULT TOLERANCE

Checkpoint
- Log data as computation goes on
- Write a full checkpoint on demand
- Pause worker threads
- Flush message queues OnRecv

Restore
- Reset all workers to checkpoint
- Reconstruct state
- Resume execution

Trade-off between mutable updates and recovery time!
**MICRO STRAGGLERS**

Networking
- Disable Nagle’s algorithm
- Reduce TCP retransmission window

Concurrency
- Reduce clock granularity to avoid spin lock delay

Garbage Collection
- Arrays of value types (similar to Plain-Old Java Objects)
- Buffer pool
Differential DATAFLOW

// 1a. Define input stages for the dataflow.
var input = controller.NewInput<string>();

// 1b. Define the timely dataflow graph.
// Here, we use LINQ to implement MapReduce.
var result = input.SelectMany(y => map(y))
    .GroupBy(y => key(y),
            (k, vs) => reduce(k, vs));

// 1c. Define output callbacks for each epoch
result.Subscribe(result => {
    ...);

// 2. Supply input data to the query.
input.OnNext(/* 1st epoch data */);
input.OnCompleted();
**END-TO-END EXAMPLE**

Stream of incoming tweets: username, raw tweet
Incremental connected components, top hashtag computation
Queries: username \(\rightarrow\) return top hashtag from their component
Stream processing → Increasingly important workload trend

Timely dataflow: Principled approach to model batch, streaming together

Vertex message model
- Compute frontier
- Distributed progress tracking