CS 744: BIG DATA SYSTEMS

Shivaram Venkataraman

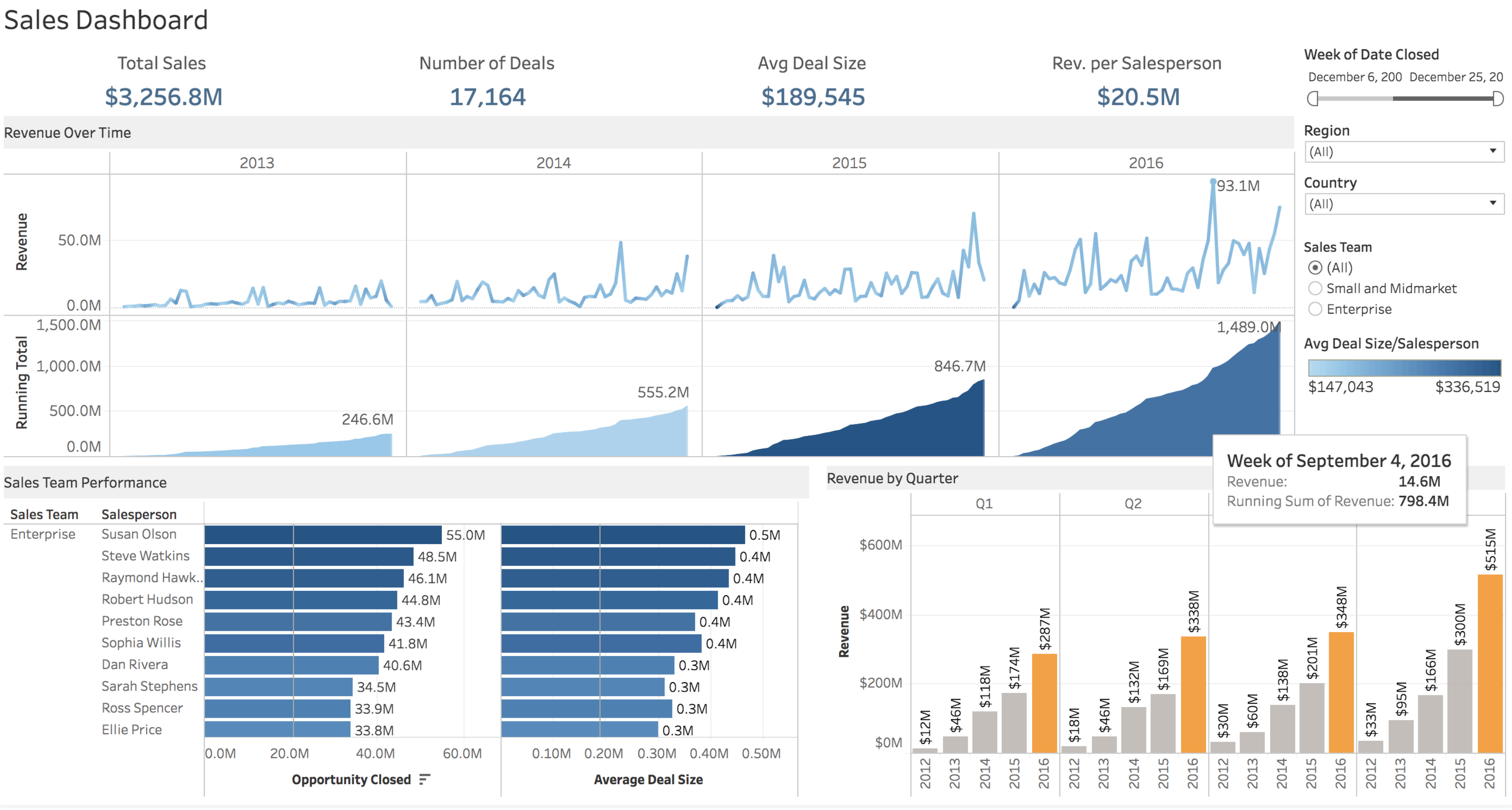
Fall 2018

ADMINISTRIVIA

- Assignment 2 grades
- Midterm review session on Nov 2 at 5pm at 1221 CS
- Course Project Proposal feedback

STREAM PROCESSING

DASHBOARDS



Sales Dashboard

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

Week of Date Closed: December 6, 200 – December 25, 20

Region: (All)

Country: (All)

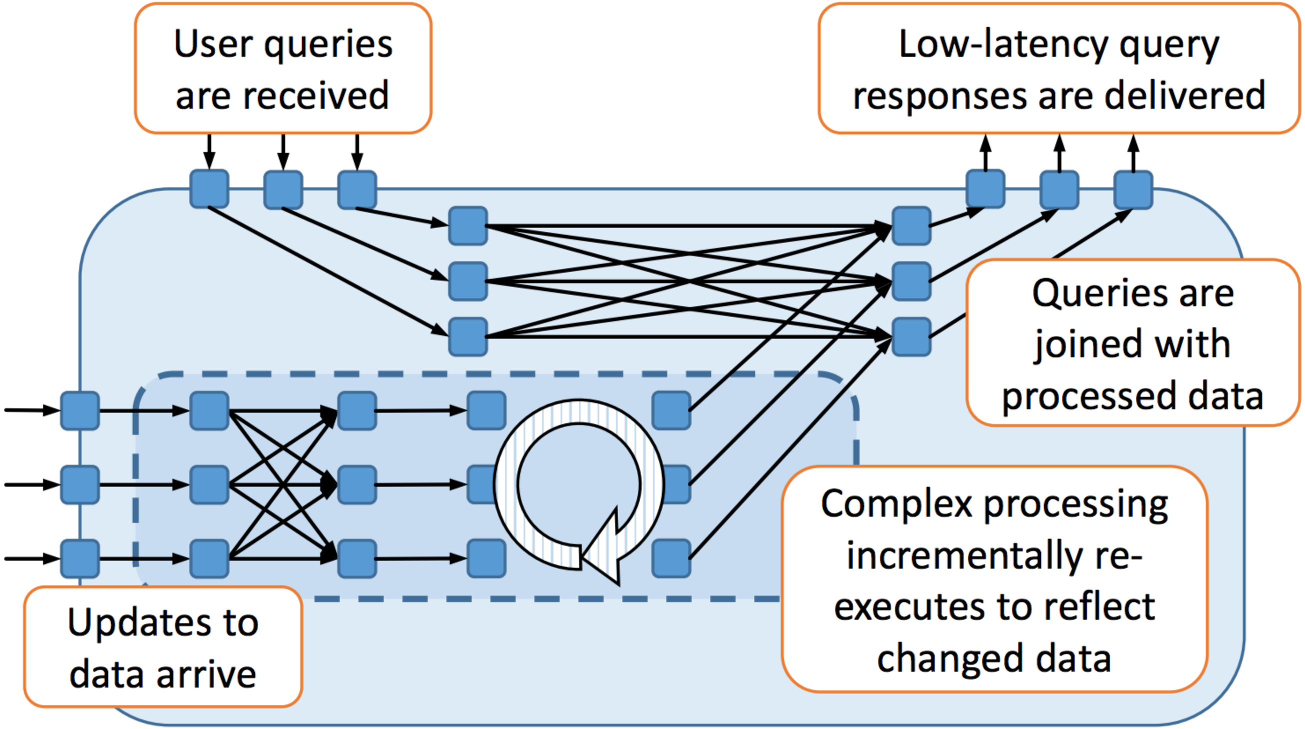
Sales Team: ● (All) ○ Small and Midmarket ○ Enterprise

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

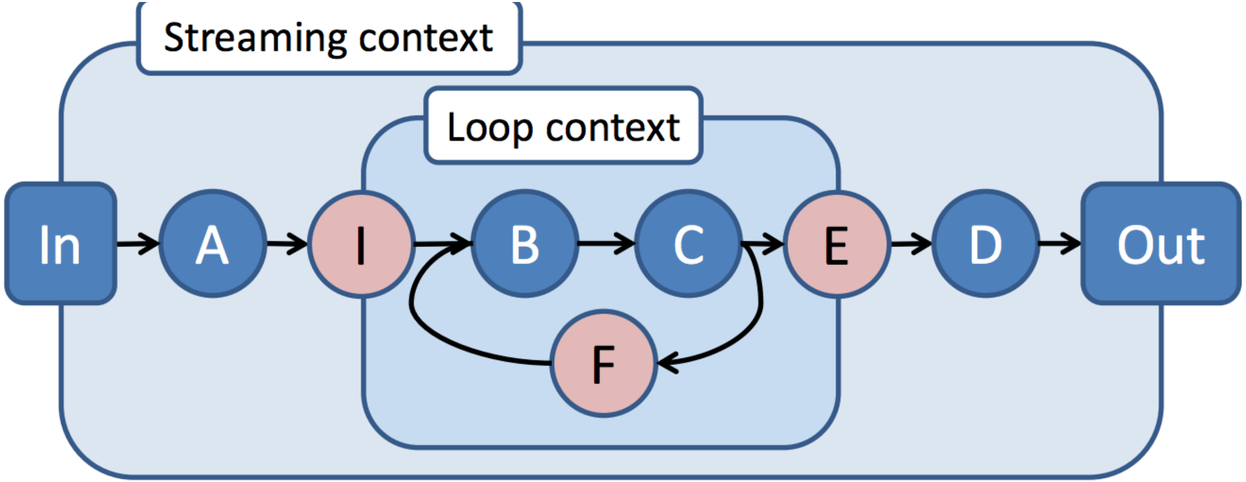
Low-latency query responses are delivered

Queries are joined with processed data

Complex processing incrementally re-executes to reflect changed data

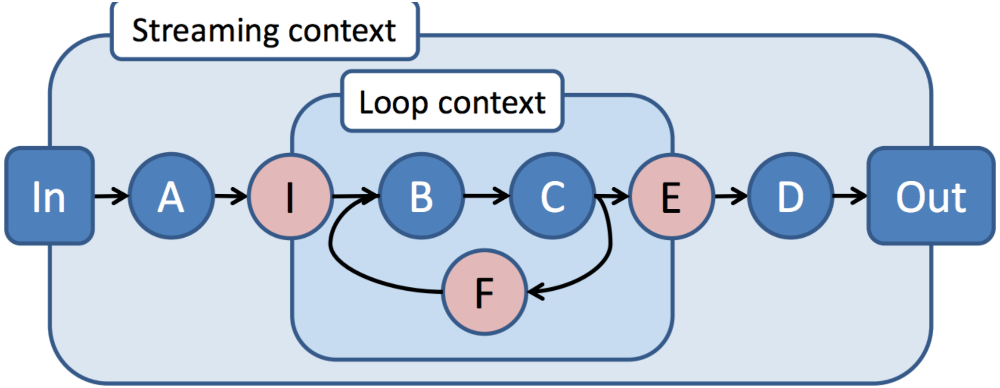
Updates to data arrive

TIMELY DATAFLOW



TIMELY DATAFLOW

$$\text{Timestamp} : (\overbrace{e \in \mathbb{N}}^{\text{epoch}}, \overbrace{\langle c_1, \ldots, c_k \rangle \in \mathbb{N}^k}^{\text{loop counters}})$$



Vertex	Input timestamp	Output timestamp
Ingress	$(e, \langle c_1, \ldots, c_k \rangle)$	$(e, \langle c_1, \ldots, c_k, 0 \rangle)$
Egress	$(e, \langle c_1, \ldots, c_k, c_{k+1} \rangle)$	$(e, \langle c_1, \ldots, c_k \rangle)$
Feedback	$(e, \langle c_1, \ldots, c_k \rangle)$	$(e, \langle c_1, \ldots, c_k + 1 \rangle)$

VERTEX API

Receiving Messages

v.OnRecv(e : Edge, m : Msg, t :Time)

v.OnNotify(t :Timestamp)

Conditions

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

Sending Messages

this.SendBy(e : Edge, m : Msg, t :Time)

this.NotifyAt(t :Timestamp)

SendBy or NotifyAt only called
with t' >= t

IMPLEMENTING TIMELY DATAFLOW

Need to track when it is safe to notify

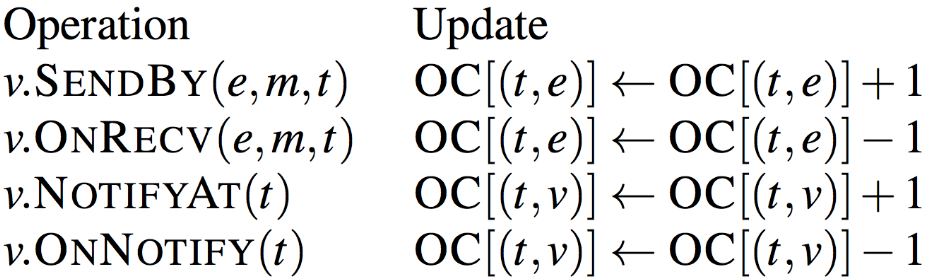
Path Summary

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

Scheduler

Occurrence and Precursor count

Precursor count = 0 → Frontier



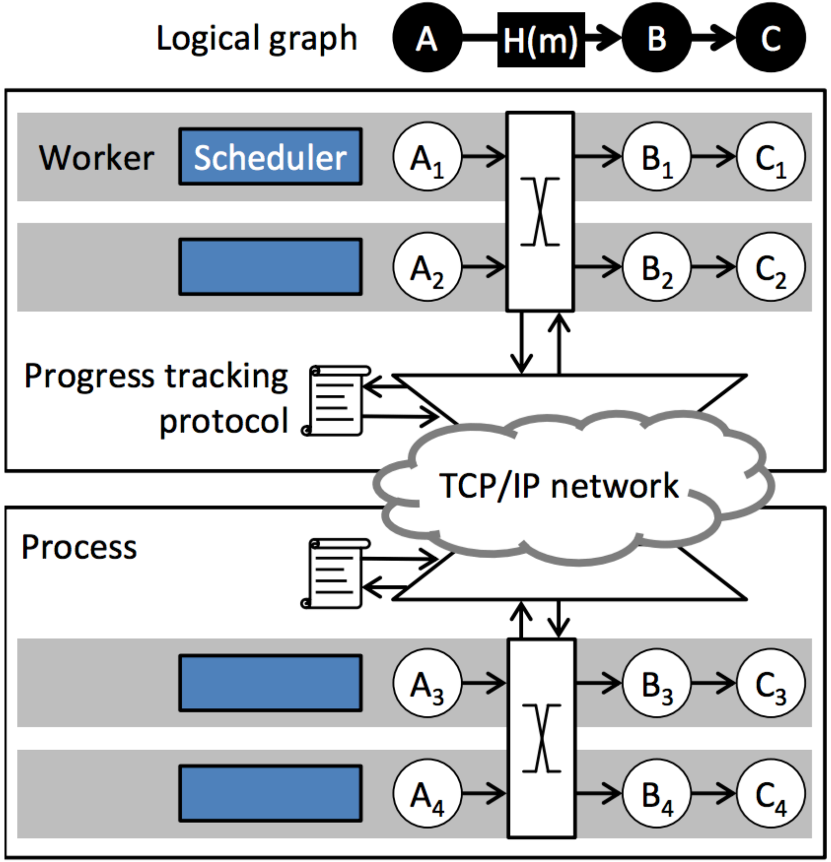
Operation	Update
$v.\textsc{SendBy}(e,m,t)$	$\mathrm{OC}[(t,e)] \leftarrow \mathrm{OC}[(t,e)] + 1$
$v.\textsc{OnRecv}(e,m,t)$	$\mathrm{OC}[(t,e)] \leftarrow \mathrm{OC}[(t,e)] - 1$
$v.\textsc{NotifyAt}(t)$	$\mathrm{OC}[(t,v)] \leftarrow \mathrm{OC}[(t,v)] + 1$
$v.\textsc{OnNotify}(t)$	$\mathrm{OC}[(t,v)] \leftarrow \mathrm{OC}[(t,v)] - 1$

ARCHITECHTURE

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 ($p \in$ Pointstamp, $\delta \in Z$)
- 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

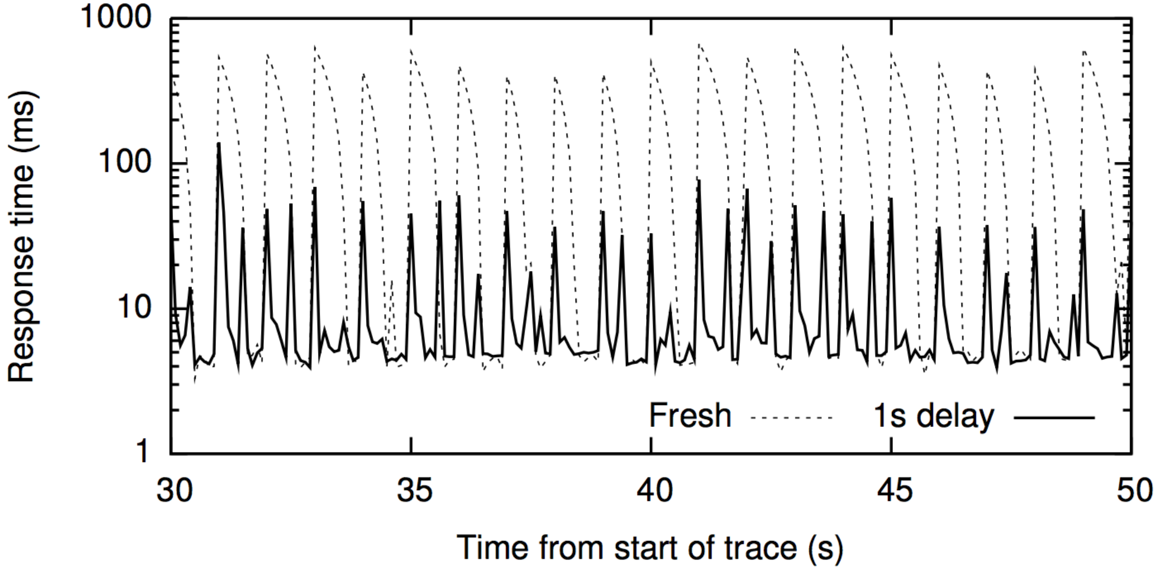
Garbage Collection

- 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 → return top hashtag from their component

SUMMARY

Stream processing → Increasingly important workload trend

Timely dataflow: Principled approach to model batch, streaming together

Vertex message model

- Compute frontier
- Distributed progress tracking