- Course Project Proposal feedback
- Midterm grades
- Checkins?
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

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

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

Diagram:
- **In**: Input vertex
- **A**: Process vertex
- **B**: Process vertex
- **C**: Process vertex
- **E**: Feedback vertex
- **D**: Process vertex
- **Out**: Output vertex
- **F**: Loop context
- **I**: Streaming context
VERTEX API

Receiving Messages
v.OnRecv(e : Edge, m : Msg, t : Time)
v.OnNotify(t : Timestamp)

Sending Messages
this.SendBy(e : Edge, m : Msg, t : Time)
this.NotifyAt(t : Timestamp)
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
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
MICRO STRAGGLERS

What is different from stragglers in MapReduce?

Sources of stragglers
Network
Concurrency
Garbage Collection
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();
Stream processing → Increasingly important workload trend

Timely dataflow: Principled approach to model batch, streaming together

Vertex message model
  - Compute frontier
  - Distributed progress tracking
DISCUSSION

https://forms.gle/v3YsW1HvnqszCuPu5
What are some example scenarios discussed in the dataflow paper that are NOT a good fit for implementation using Naiad?
Consider you are implementing a micro-batch streaming API on top of Apache Spark. What are some of the bottlenecks/challenges you might have in building such a system?