In Progress
- Course Project Proposal feedback
- Midterm grading
Scalable Storage Systems

Datacenter Architecture

Resource Management

Computational Engines

Applications

Machine Learning  SQL  Streaming  Graph

SQL  Streaming  Graph

Applications

Machine Learning  SQL  Streaming  Graph

Computational Engines

Scalable Storage Systems

Resource Management

Datacenter Architecture
STREAMING + ITERATIVE COMPUTATION

- User queries are received
- Updates to data arrive
- Low-latency query responses are delivered
- Queries are joined with processed data
- Complex processing incrementally re-executes to reflect changed data
TIMELY DATAFLOW
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:
- Vertex: In, A, B, C, E, D, Out
- Edges: In -> A -> I -> B -> C -> E -> D -> Out
- Loop context: F
- Streaming context
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)
class DistinctCount<S,T> : Vertex<T> {
    Dictionary<T, Dictionary<S,int>> counts;
    void OnRecv(Edge e, S msg, T time) {
        if (!counts.ContainsKey(time)) {
            counts[time] = new Dictionary<S,int>();
            this.NotifyAt(time);
        }
        if (!counts[time].ContainsKey(msg)) {
            counts[time][msg] = 0;
            this.SendBy(output1, msg, time);
        }
        counts[time][msg]++;;
    }

    void OnNotify(T time) {
        foreach (var pair in counts[time])
            this.SendBy(output2, pair, time);
        counts.Remove(time);
    }
}
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
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 ($p \in \text{Pointstamp}, \delta \in \mathbb{Z}$)
   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
// 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/cYYy14JfX71qBzY66
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?
Next class: Spark Streaming
Course project peer feedback due tonight!