

CS 744: SPARK STREAMING

Shivaram Venkataraman

Spring 2025

ADMINISTRIVIA

- Course Projects feedback
- Midterm grades
- Shivaram travel week after Spring break
- Cloudlab reservations: Per-user from now
- Google credits!
- Reading group interest?

Applications

Machine Learning

SQL

Streaming

Graph

Computational Engines

Scalable Storage Systems

Resource Management



Datacenter Architecture



DASHBOARDS

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, 20



Region

(All)

Country

(All)

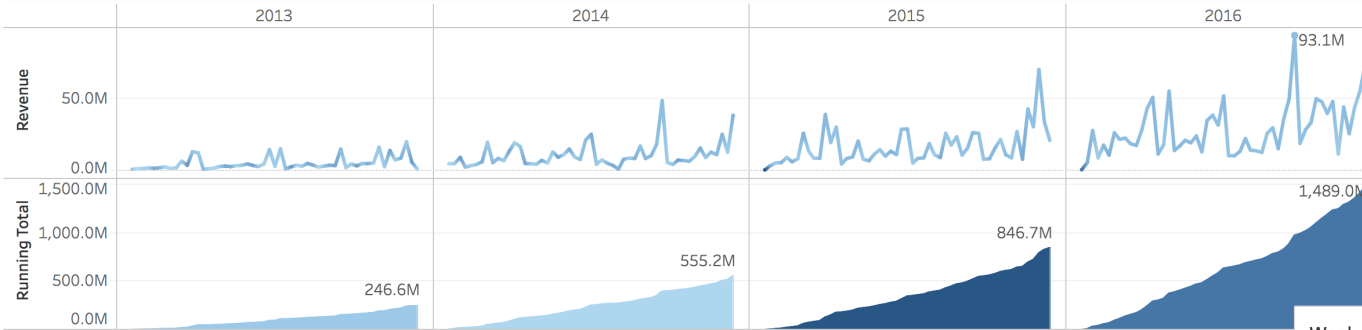
Sales Team

- (All)
- Small and Midmarket
- Enterprise

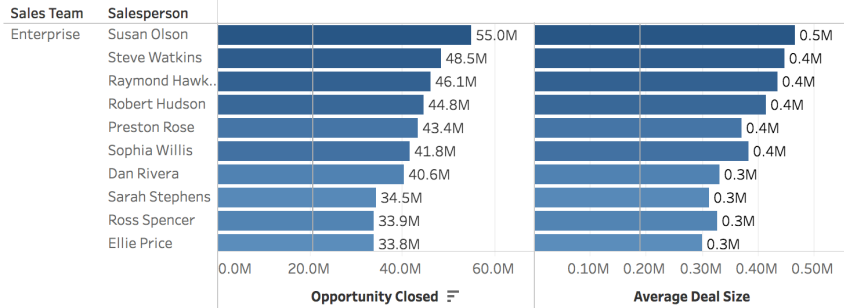
Avg Deal Size/Salesperson



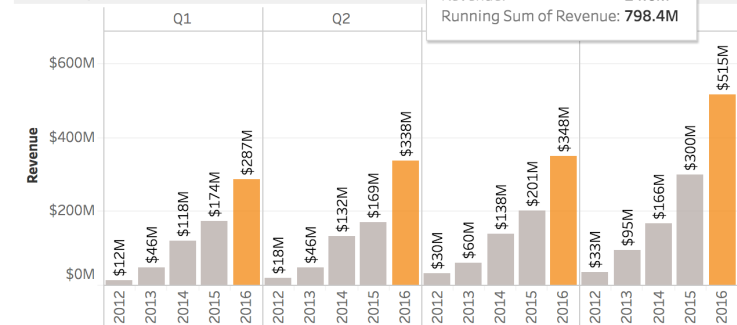
Revenue Over Time



Sales Team Performance



Revenue by Quarter

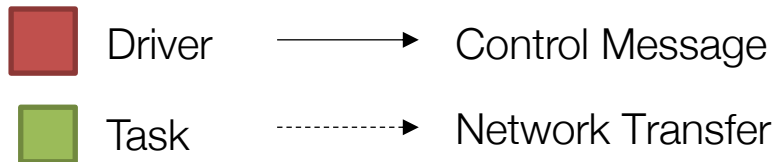
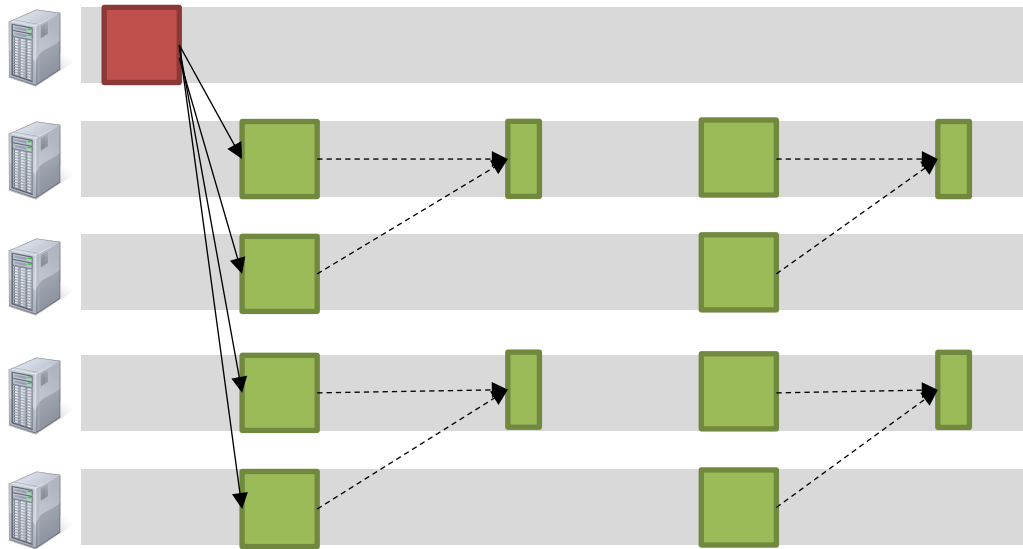


Week of September 4, 2016

Revenue: **14.6M**

Running Sum of Revenue: **798.4M**

CONTINUOUS OPERATOR MODEL

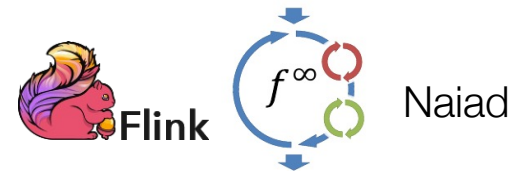


Long-lived operators

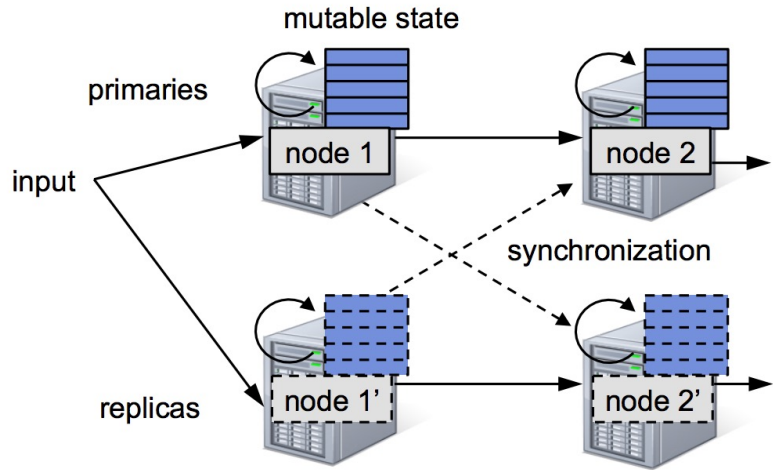
Mutable State

Distributed Checkpoints
for Fault Recovery

Stragglers ?



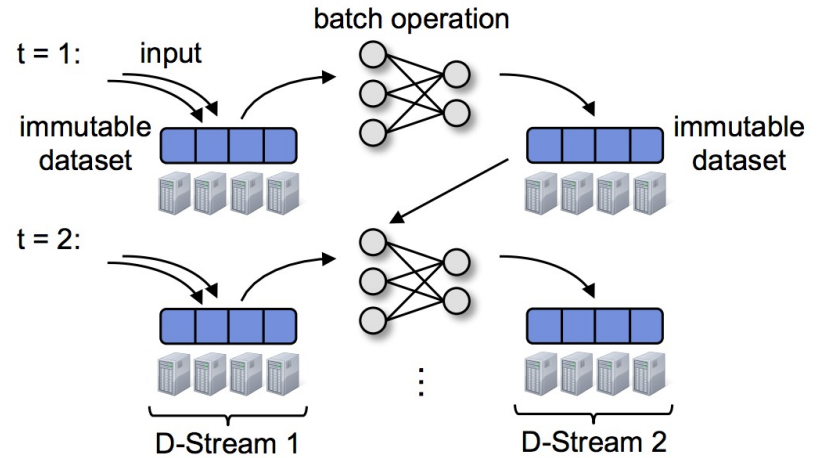
CONTINUOUS OPERATORS



SPARK STREAMING: GOALS

1. Scalability to hundreds of nodes
2. Minimal cost beyond base processing (no replication)
3. Second-scale latency
4. Second-scale recovery from faults and stragglers

DISCRETIZED STREAMS (DSTREAMS)

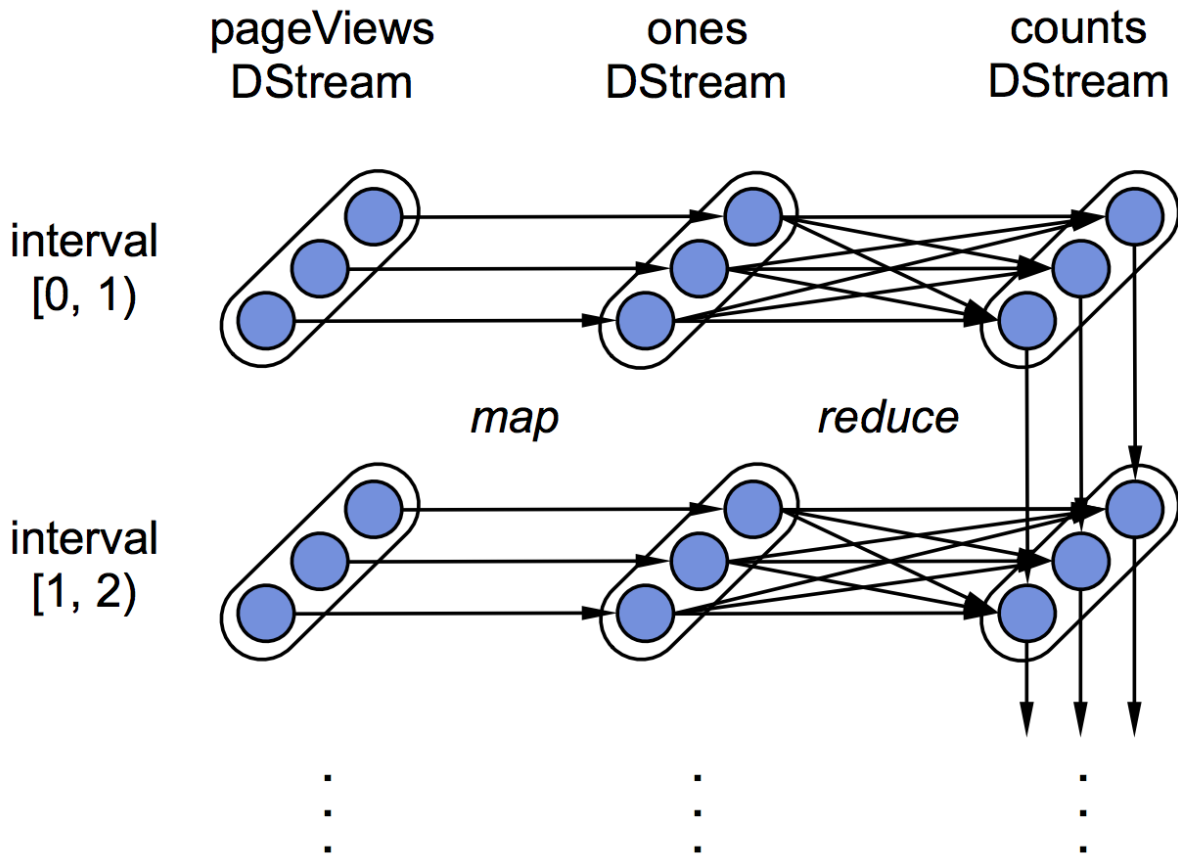


EXAMPLE

```
pageViews =  
  readStream(http://...,  
            "1s")
```

```
ones = pageViews.map(  
  event =>(event.url, 1))
```

```
counts =  
  ones.runningReduce(  
    (a, b) => a + b)
```



DSTREAM API

Transformations

Stateless: map, reduce, groupBy, join

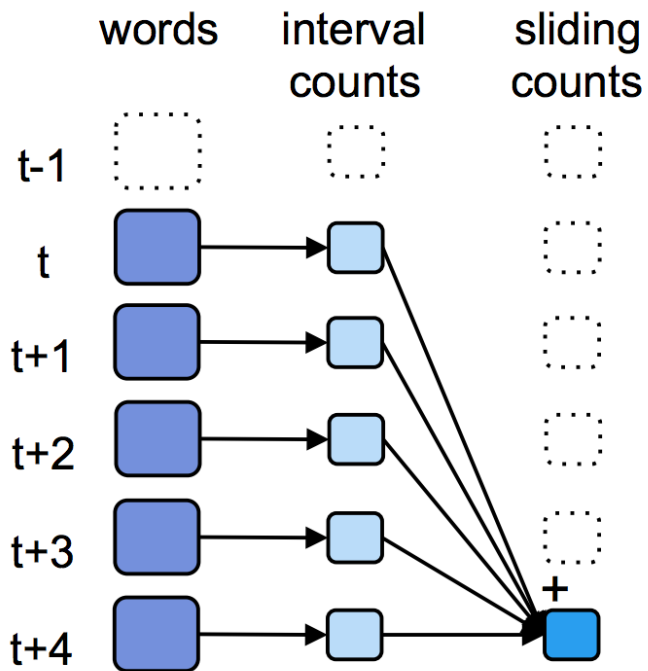
Stateful:

Sliding window("5s") → RDDs with data in [0,5), [1,6), [2,7)

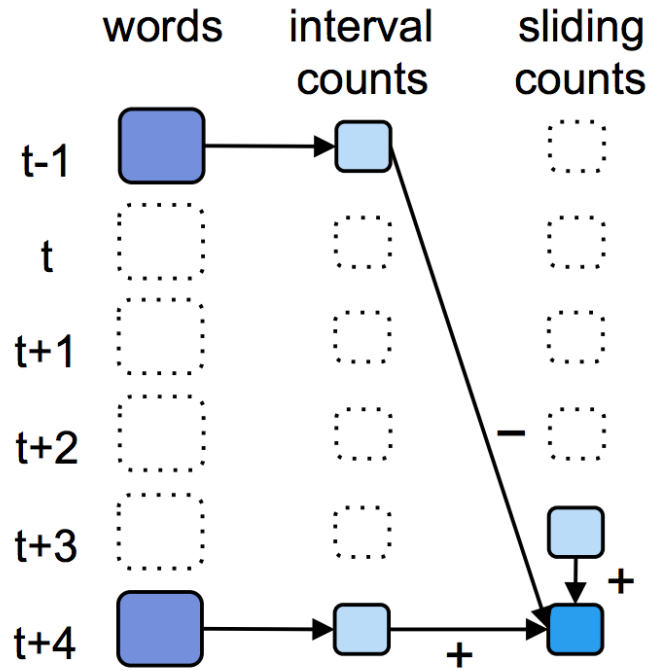
reduceByWindow("5s", (a, b) => a + b)

SLIDING WINDOW

Add
previous 5
each time



(a) Associative only



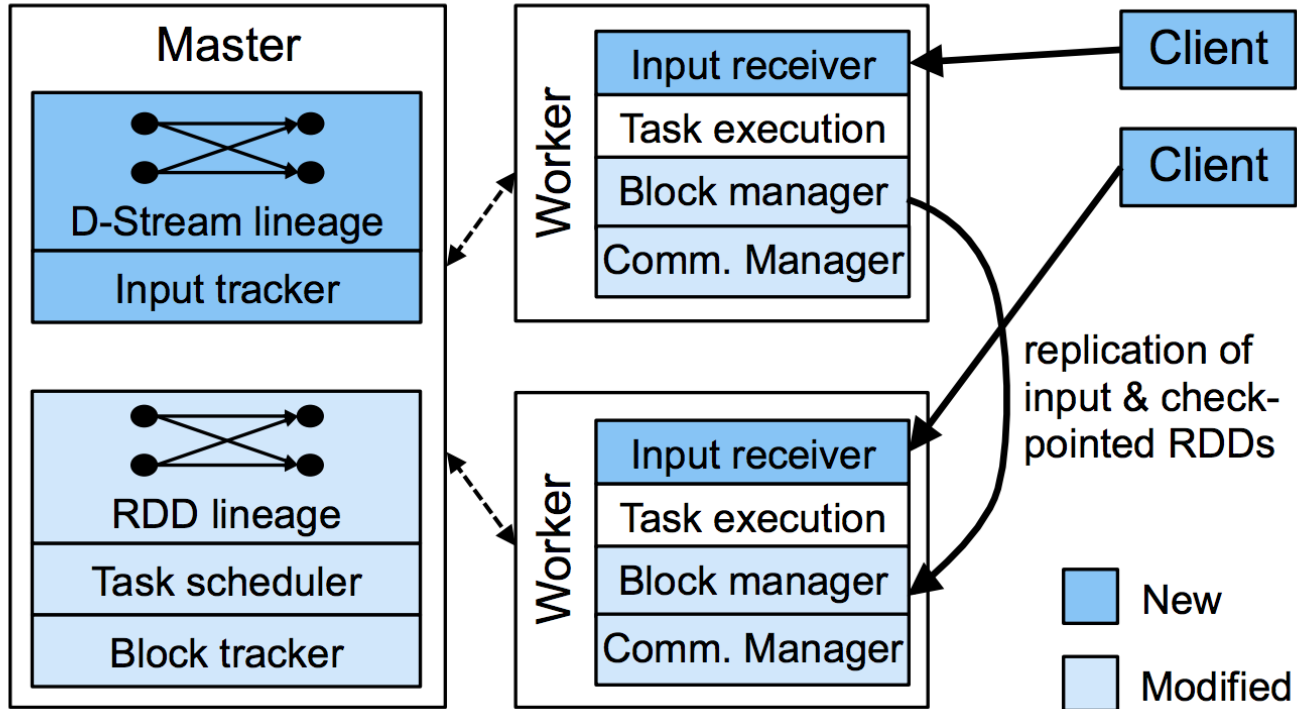
(b) Associative & invertible

STATE MANAGEMENT

Tracking State: streams of (Key, Event) \rightarrow (Key, State)

```
events.track(  
  (key, ev) => 1,  
  
  (key, st, ev) => ev == Exit ? null : 1,  
  
  "30s")
```

SYSTEM IMPLEMENTATION



OPTIMIZATIONS

Timestep Pipelining

- No barrier across timesteps unless needed

- Tasks from the next timestep scheduled before current finishes

Checkpointing

- Async I/O, as RDDs are immutable

- Truncate lineage after checkpoint

FAULT TOLERANCE: PARALLEL RECOVERY

Worker failure

- Need to recompute state RDDs stored on worker
- Re-execute tasks running on the worker

Strategy

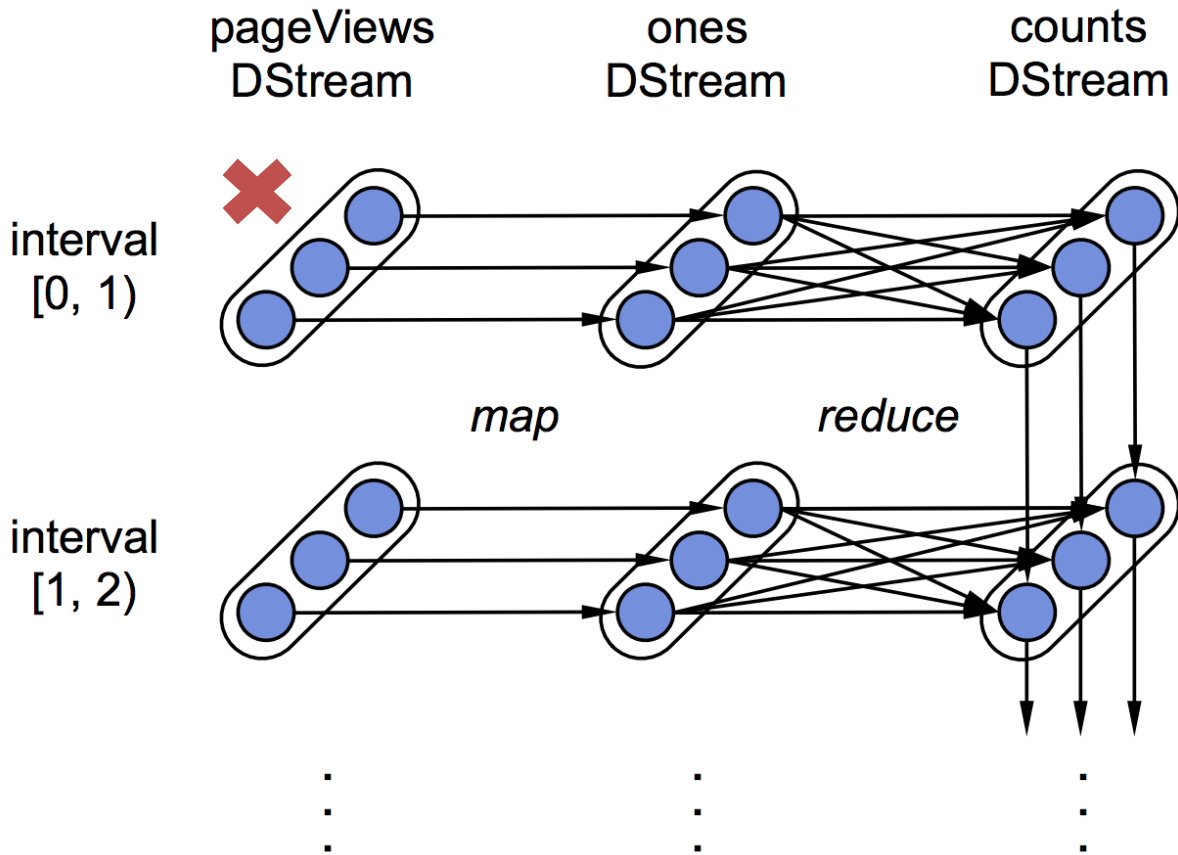
- Run all independent recovery tasks in **parallel**
- Parallelism from partitions *in timestep* and *across timesteps*

EXAMPLE

```
pageViews =  
  readStream(http://...,  
            "1s")
```

```
ones = pageViews.map(  
  event =>(event.url, 1))
```

```
counts =  
  ones.runningReduce(  
    (a, b) => a + b)
```



FAULT TOLERANCE

Straggler Mitigation: Use speculative execution

Driver Recovery

- At each timestep, save graph of DStreams and Scala function objects
- Workers connect to a new driver and report their RDD partitions
- Note: No problem if a given RDD is computed twice (determinism).

SUMMARY

Micro-batches: New approach to stream processing

Simplifies fault tolerance, straggler mitigation

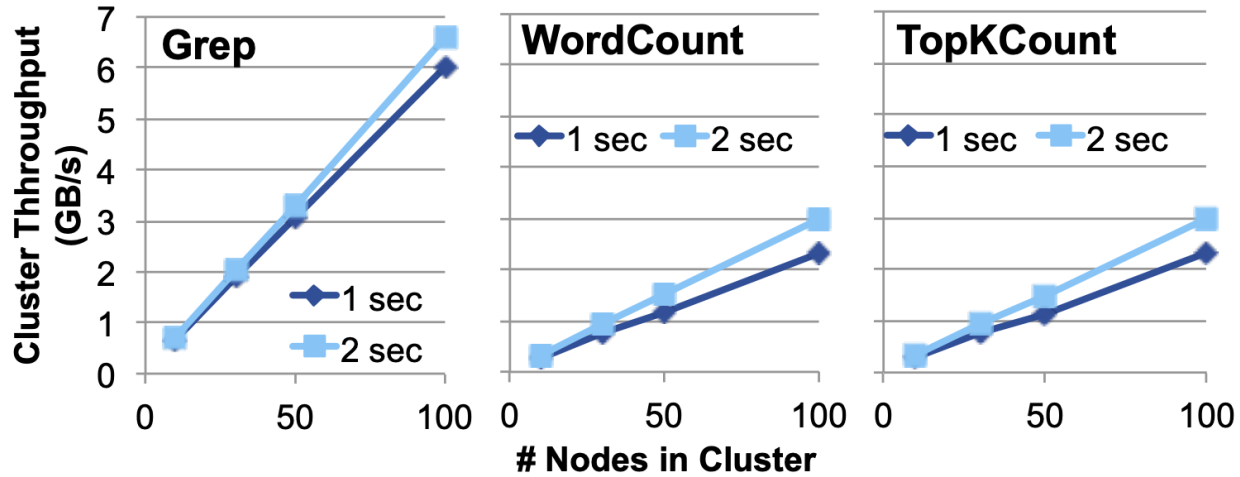
Unifying batch, streaming analytics



DISCUSSION

<https://forms.gle/6j3PCKV6AG7WVGqIw9>

If the latency bound was made to 100ms, how do you think the above figure would change? What could be the reasons for it?



Discuss what part of queries considered by the Dataflow paper could use the "track" operation in Spark Streaming.

NEXT STEPS

Next week: Spring break!

Next class: Graph processing!

Midterm grades soon!