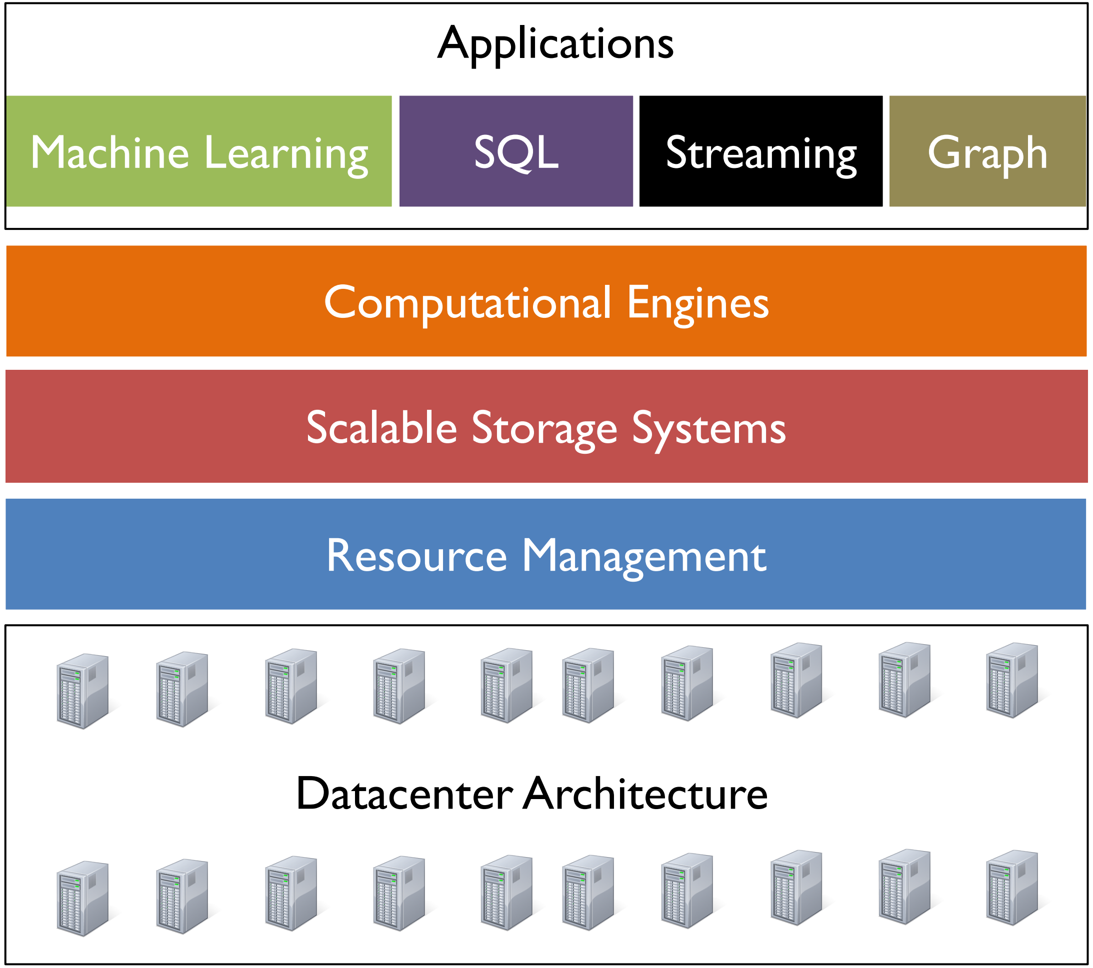
CS 744: SPARK STREAMING

Shivaram Venkataraman

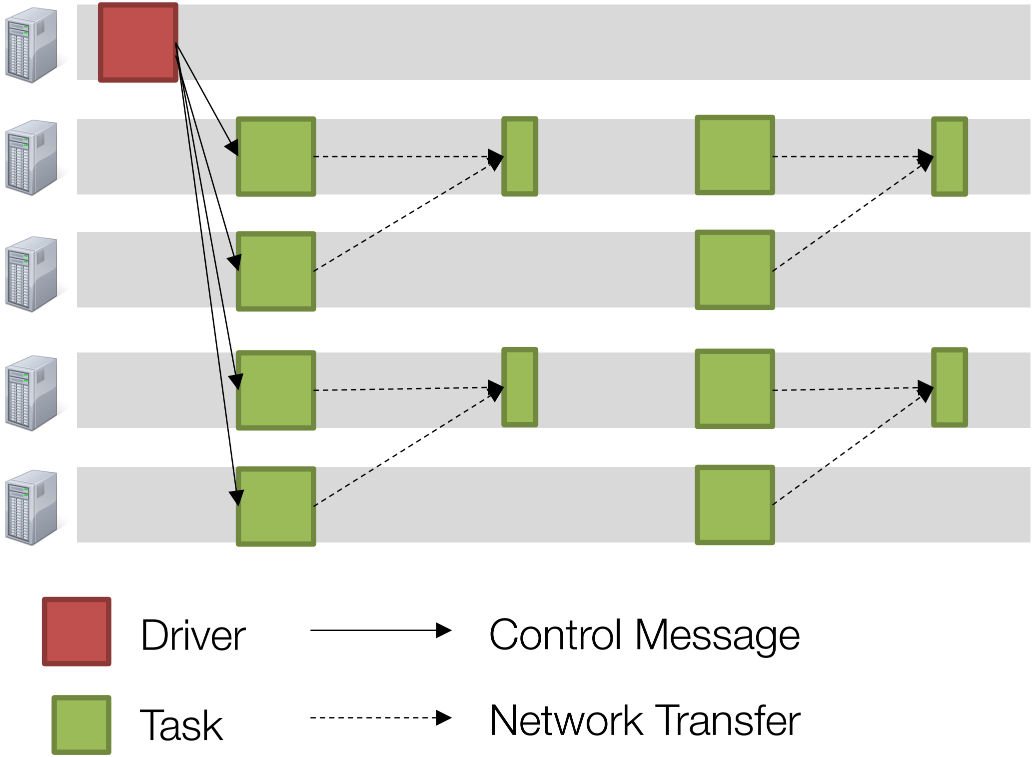
Fall 2019

ADMINISTRIVIA

- Midterm grades this week
- Course Projects sign up for meetings


Applications
Machine Learning
SQL
Streaming
Graph
Computational Engines
Scalable Storage Systems
Resource Management
Datacenter Architecture

CONTINUOUS OPERATOR MODEL



Driver

Task

Control Message

Network Transfer

Long-lived operators

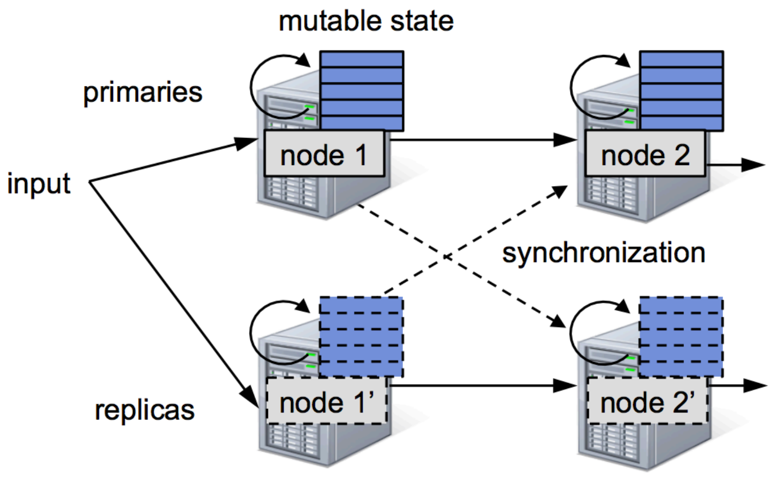
Mutable State

Distributed Checkpoints for Fault Recovery

Stragglers ?

Naiad

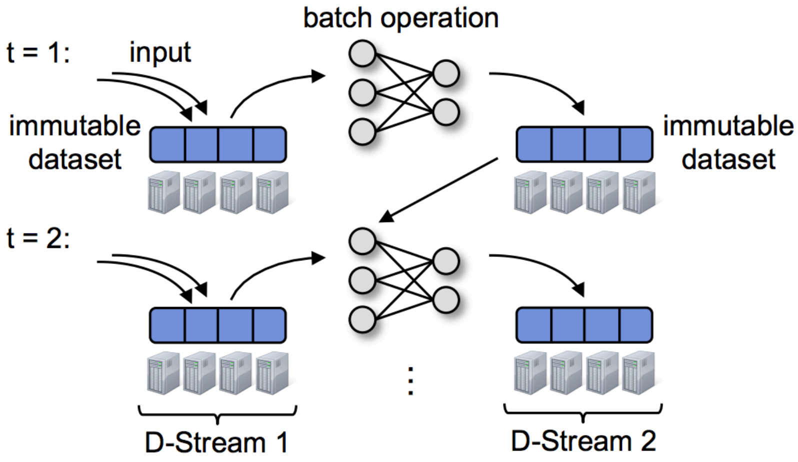
CONTINUOUS OPERATORS



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)

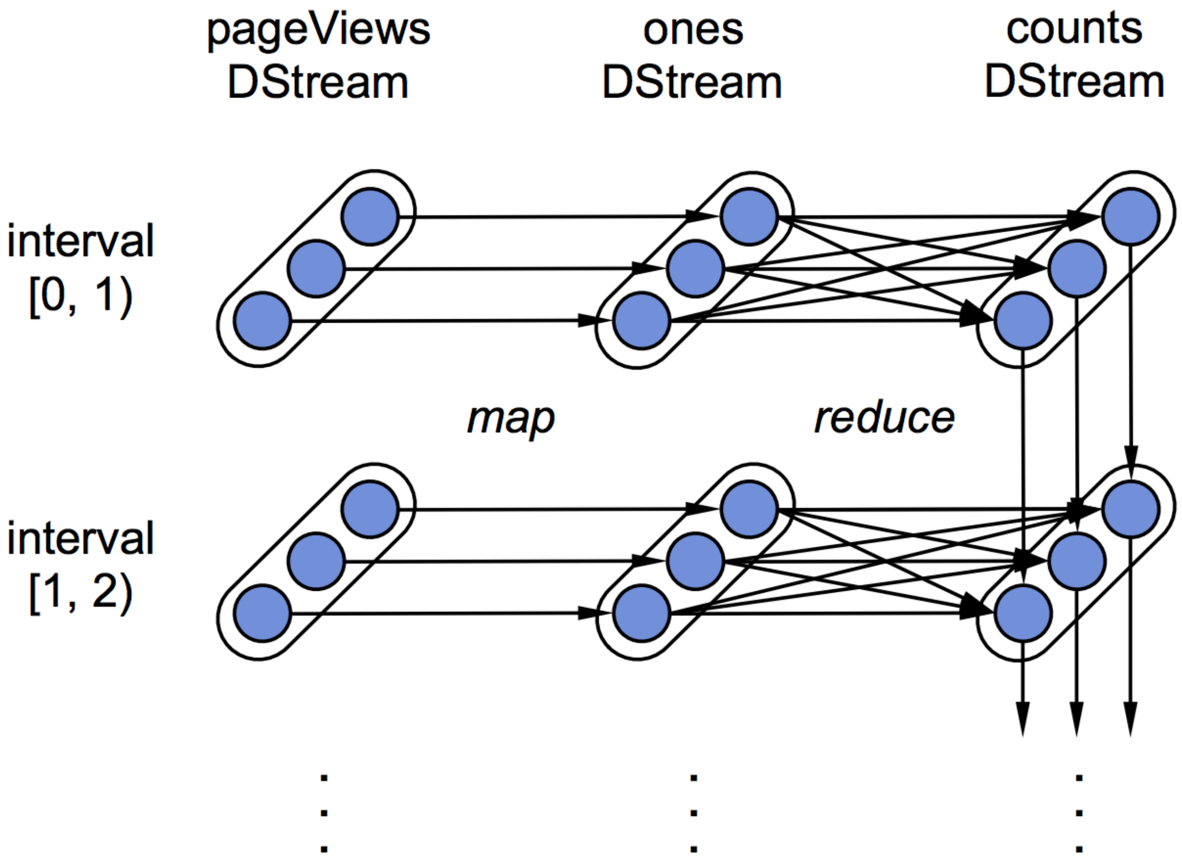


EXAMPLE

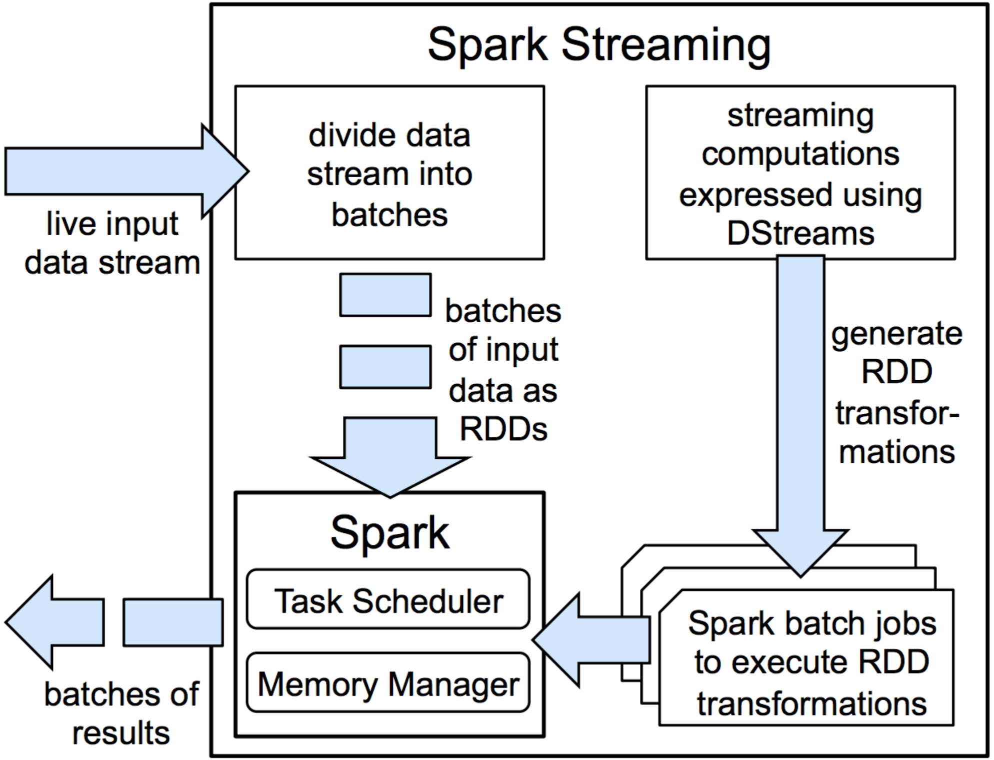
```
pageViews =
  readStream(http://...,
             "1s")

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

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



ARCHITECHTURE



DSTREAM API

Transformations

Stateless: map, reduce, groupBy, join

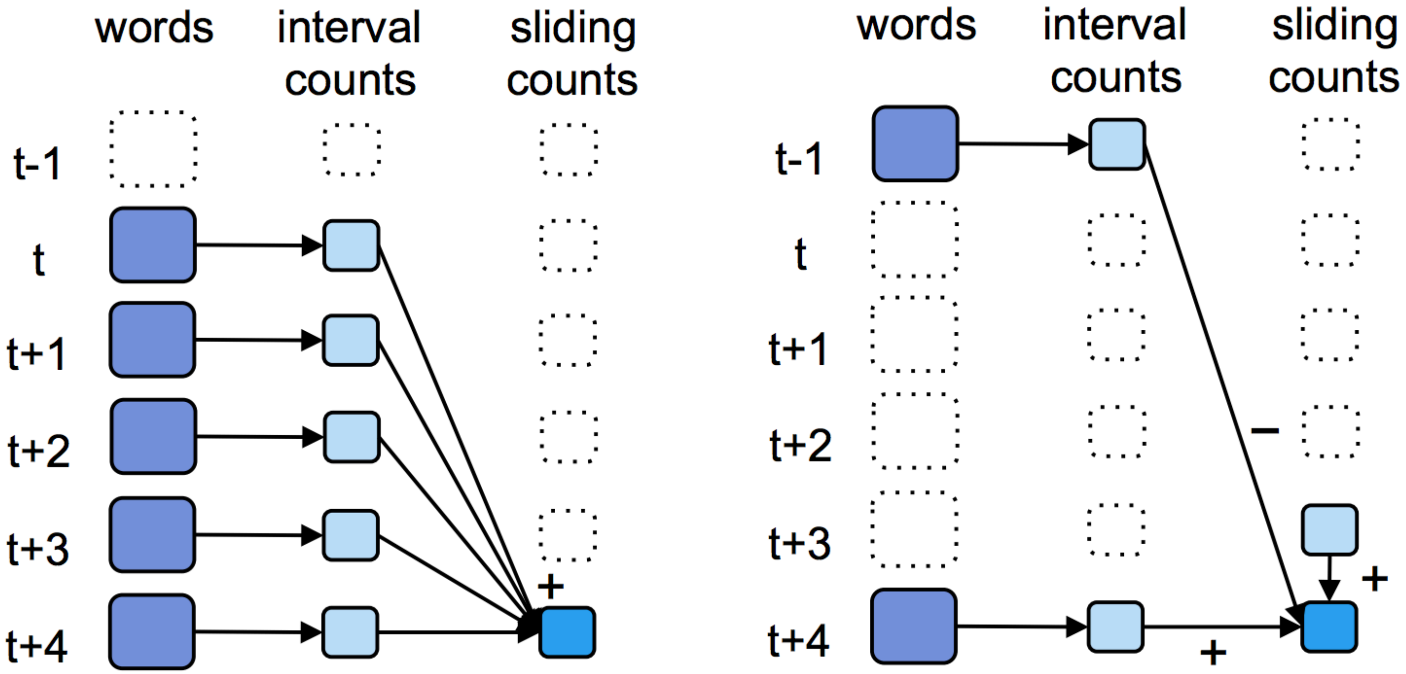
Stateful:

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

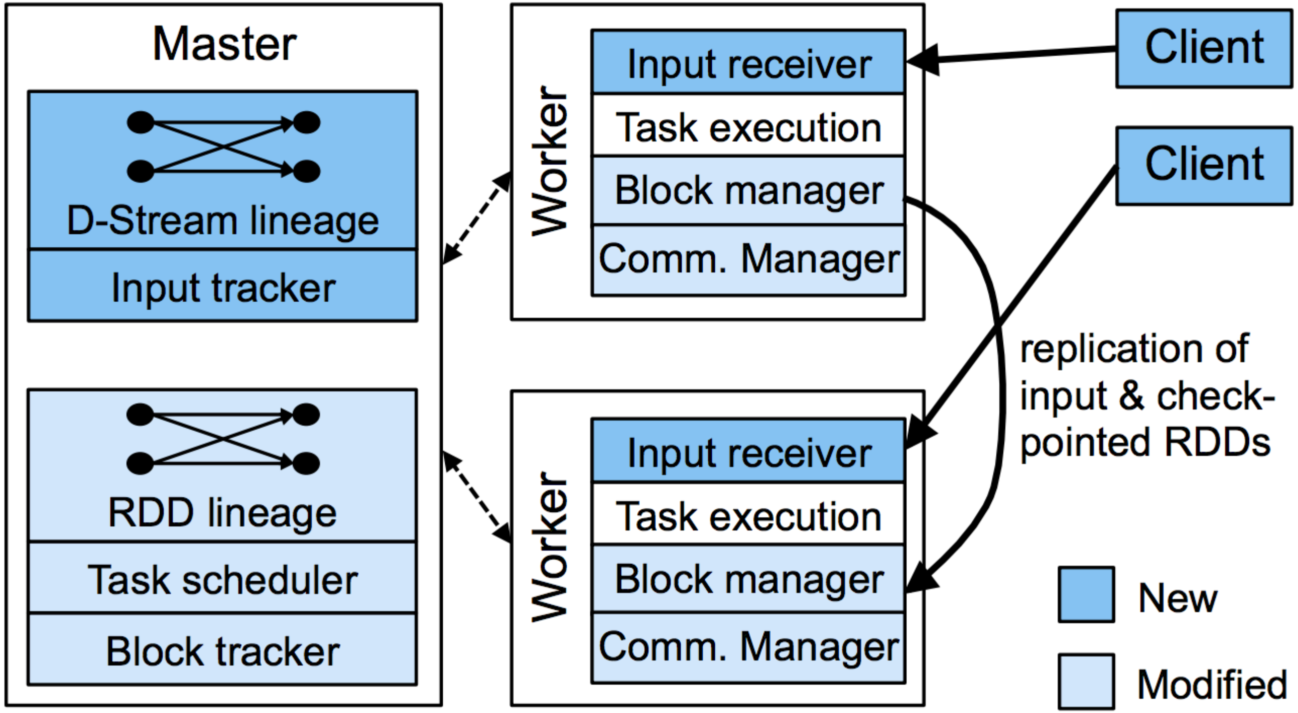
Tracking State: streams of (Key, Event) → (Key, State)

```
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

Forget 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

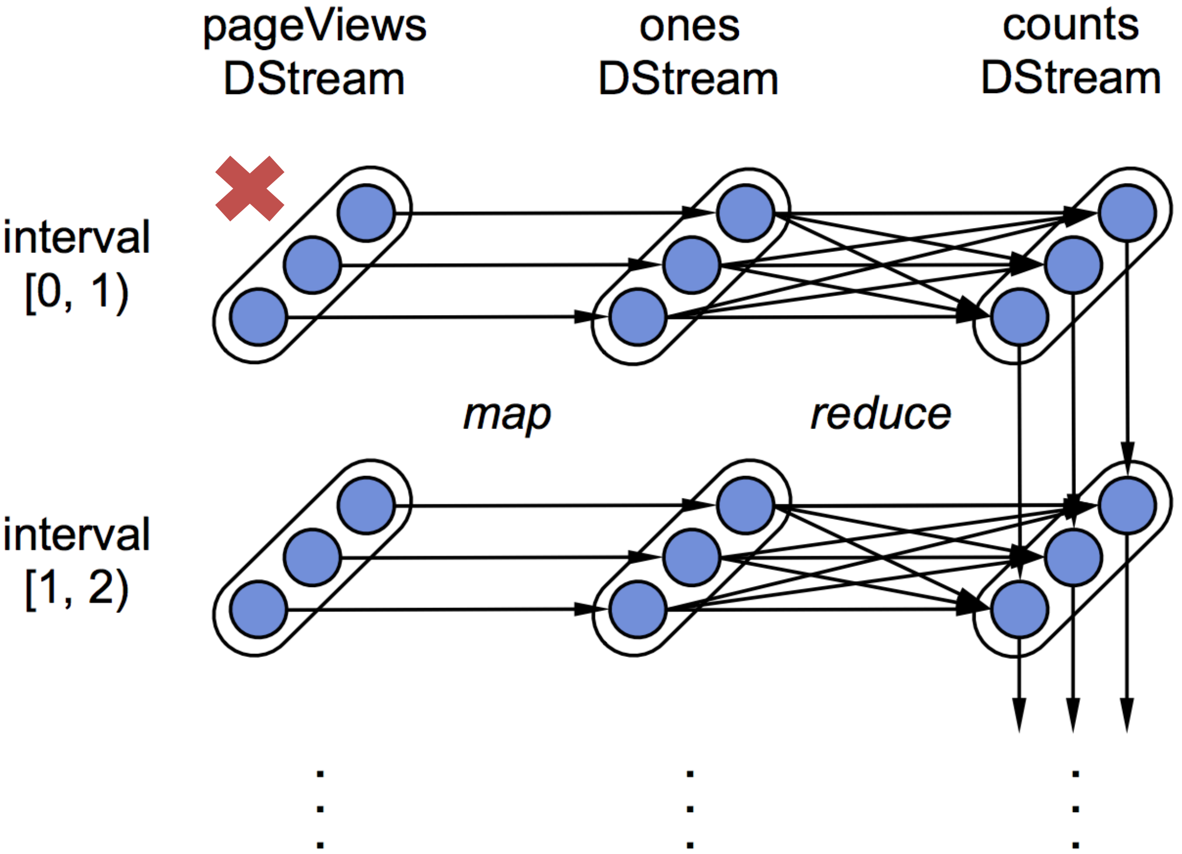
- Run all independent recovery tasks in parallel
- Parallelism from partitions *in timestep* and *across timesteps*

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

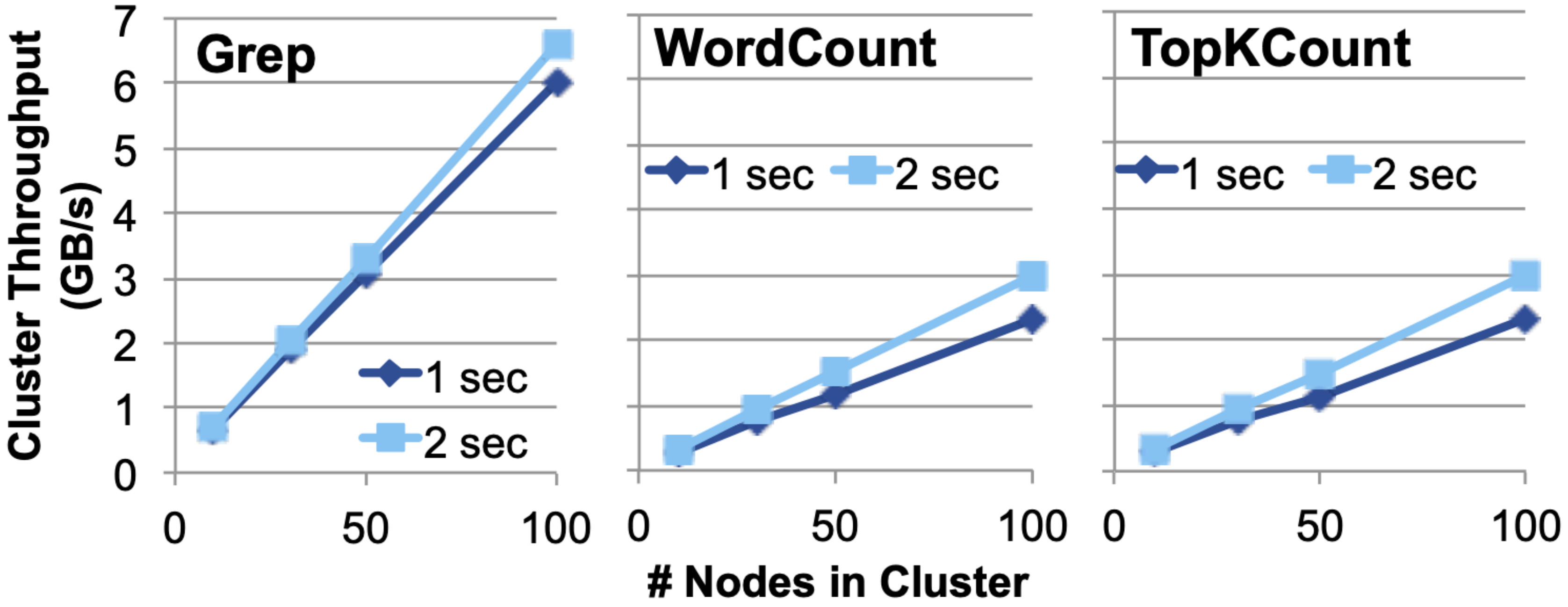
Task runs more than 1.4x longer than median task → straggler

Master Recovery

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

DISCUSSION

https://forms.gle/xUvzC1bdV7H48mTM8


Grep
WordCount
TopKCount
1 sec
2 sec
Cluster Thhroughput (GB/s)
0
1
2
3
4
5
6
7
0
50
100
Nodes in Cluster

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

Consider the pros and cons of approaches in Naiad vs Spark Streaming. What application properties would you use to decide which system to choose?

NEXT STEPS

Next class: Graph processing

Sign up for project check-ins!

SHORTCOMINGS?

Expressiveness

- Current API requires users to “think” in micro-batches

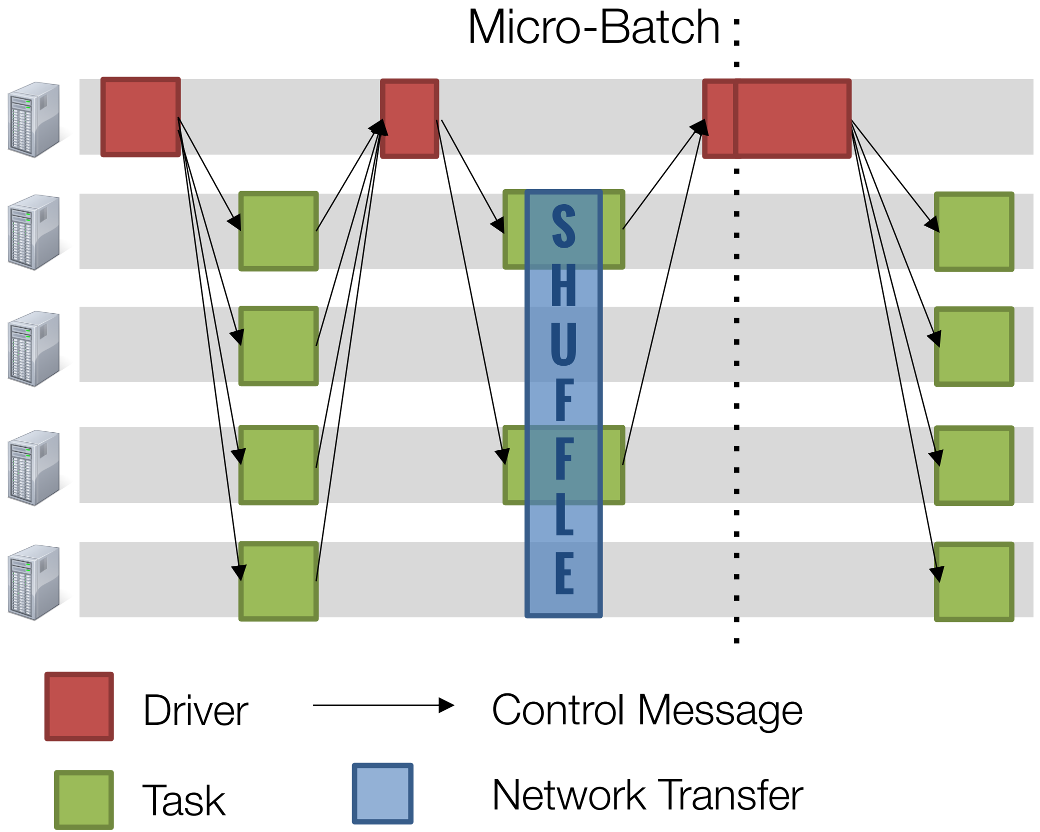
Setting batch interval

- Manual tuning. Higher batch → better throughput but worse latency

Memory usage

- LRU cache stores state RDDs in memory

COMPUTATION MODEL: MICRO-BATCHES



SUMMARY

Micro-batches: New approach to stream processing

Higher latency for fault tolerance, straggler mitigation

Unifying batch, streaming analytics