Discretized Streams: Fault-Tolerant Streaming Computation at Scale
Matei Zaharia, Tathagata Das, Haoyuan Li, Timothy Hunter, Scott Shenker, Ion Stoica

Motivation and Goal

- Many important applications must process large data streams at second-scale latencies
- Need a system for building large-scale apps that
  - Scales to 10s - 100s of nodes with second-scale latencies
  - Efficiently recovers from node failures and slow nodes

Existing Systems

- Traditional streaming systems use continuous operator model with mutable state
- Do not have fast AND efficient fault-tolerance
  - Node replication: not cost-efficient
  - Upstream backup: slow recovery
- Do not handle slow nodes well
  - Can process 6 GB/sec (60M records/sec) from 100 data streams on 100 nodes at 1 second latency
- Higher throughput than Storm
  - Spark Streaming: 670k records/sec/node
  - Storm: 115k records/sec/node
- Fast recovery from node failures and stragglers

Benefits

- Parallel Recovery on Failure: Rebuild lost partitions of datasets in parallel on multiple existing nodes; faster than upstream backup and more cost-efficient than node replication
- Straggler Mitigation: Detect slower tasks and speculatively run them on other nodes
- Integration with batch+interactive processing: Combine live data with historic data, make ad-hoc queries on live streams

Evaluation

- Can process 6 GB/sec (60M records/sec) from 100 data streams on 100 nodes at 1 second latency
- Higher throughput than Storm
  - Spark Streaming: 670k records/sec/node
  - Storm: 115k records/sec/node
- Fast recovery from node failures and stragglers

Discretized Stream Processing

- Observation: Batch processing like MapReduce have efficient techniques for fault-tolerance and straggler mitigation
- Key Idea: Treat streaming computations as series of batch jobs with small batches of data
  - Apply known recovery techniques at smaller timer intervals
  - Try to make batch sizes as small as possible
- Processing Model:
  - Input data and intermediate state data stored in cluster memory as immutable, partitioned datasets
  - Previous state dataset + next input dataset = new state dataset
- Programming Abstraction: Discretized Stream (DStream) is a series of partitioned datasets representing a stream of data
- Example: Running count of page views using DStreams

```scala
views = readStream("http://...", "1s")
ones = views.map(ev => (ev.url, 1))
counts = ones.runningReduce(_ + _)
```

- `runningReduce(_ + _)` combines the page view counts of previous interval with that of the current interval to generate the running count

---

Discretized Stream Processing (DStream)

- Observation: Batch processing like MapReduce have efficient techniques for fault-tolerance and straggler mitigation
- Key Idea: Treat streaming computations as series of batch jobs with small batches of data
  - Apply known recovery techniques at smaller timer intervals
  - Try to make batch sizes as small as possible
- Processing Model:
  - Input data and intermediate state data stored in cluster memory as immutable, partitioned datasets
  - Previous state dataset + next input dataset = new state dataset
- Programming Abstraction: Discretized Stream (DStream) is a series of partitioned datasets representing a stream of data
- Example: Running count of page views using DStreams

```scala
views = readStream("http://...", "1s")
ones = views.map(ev => (ev.url, 1))
counts = ones.runningReduce(_ + _)
```

- `runningReduce(_ + _)` combines the page view counts of previous interval with that of the current interval to generate the running count

---

Discretized Stream Processing (DStream)

- Observation: Batch processing like MapReduce have efficient techniques for fault-tolerance and straggler mitigation
- Key Idea: Treat streaming computations as series of batch jobs with small batches of data
  - Apply known recovery techniques at smaller timer intervals
  - Try to make batch sizes as small as possible
- Processing Model:
  - Input data and intermediate state data stored in cluster memory as immutable, partitioned datasets
  - Previous state dataset + next input dataset = new state dataset
- Programming Abstraction: Discretized Stream (DStream) is a series of partitioned datasets representing a stream of data
- Example: Running count of page views using DStreams

```scala
views = readStream("http://...", "1s")
ones = views.map(ev => (ev.url, 1))
counts = ones.runningReduce(_ + _)
```