Discretized Streams: Fault-Tolerant Streaming Computation at Scale

Matei Zaharia, Tathagata Das, Haoyuan Li, Timothy Hunter, Scott Shenker, Ion Stoica

University of California, Berkeley

Published in SOSP '13

Motivation

- Faults and stragglers inevitable in large clusters running "big data" applications.
- Streaming applications must recover from these quickly.
- Current distributed streaming systems, including Storm, TimeStream, MapReduce Online provide fault recovery in an expensive manner.
  - Involves hot replication which requires 2x hardware or upstream backup which has long recovery time.

Previous Methods

- Hot replication
  - two copies of each node, 2x hardware.
  - straggler will slow down both replicas.
- Upstream backup
  - nodes buffer sent messages and replay them to new node.
  - stragglers are treated as failures resulting in long recovery step.

- Conclusion : need for a system which overcomes these challenges

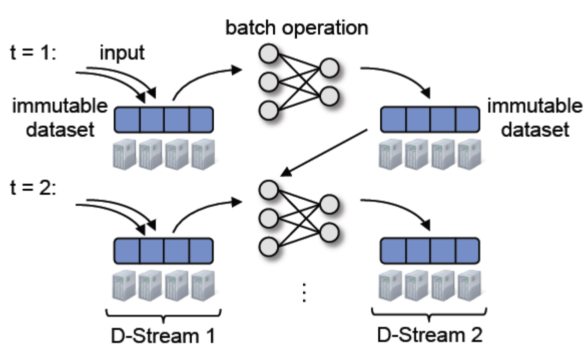
- Voila ! D-Streams

Computation Model

- Streaming computations treated as a series of deterministic batch computations on small time intervals.
- Data received in each interval is stored reliably across the cluster to form input datatsets
- At the end of each interval dataset is subjected to deterministic parallel operations and two things can happen
  - new dataset representing program output which is pushed out to stable storage
  - intermediate state stored as resilient distributed datasets (RDDs)

PURDUE UNIVERSITY

D-Stream processing model

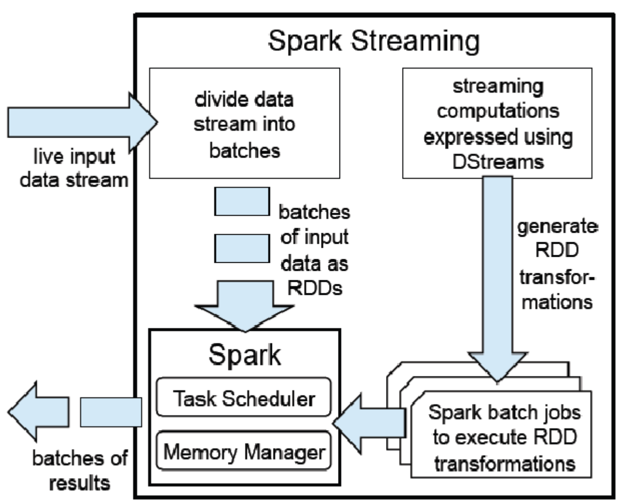


PURDUE UNIVERSITY

What are D-Streams ?

- sequence of immutable, partitioned datasets (RDDs) that can be acted on by deterministic transformations
- transformations yield new D-Streams, and may create intermediate state in the form of RDDs
- Example :-
  - pageViews = readStream("http://...", "1s")
  - ones = pageViews.map(event => (event.url, 1))
  - counts = ones.runningReduce((a, b) => a + b)

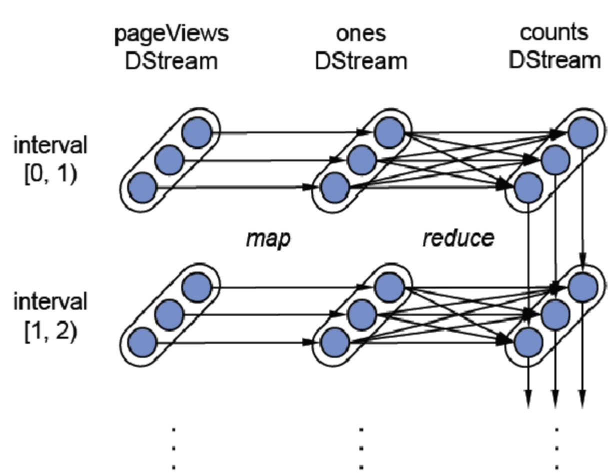
High-level overview of Spark Streaming system



Recovery

- D-Streams & RDDs track their lineage, that is, the graph of deterministic operations used to build them.
- When a node fails, it recomputes the RDD partitions that were on it by re-running the tasks that built them from the original input data stored reliably in the cluster.
- Checkpointing of state RDDs is done periodically

Lineage graph for RDDs



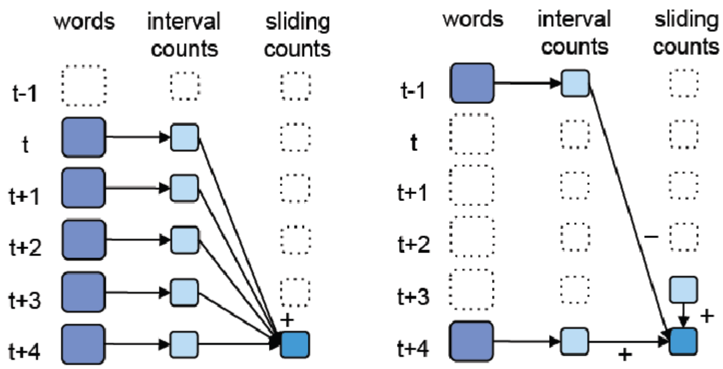
D-Stream API

- Users register one or more streams using a functional API
- Input streams can either be read by listening on a port or periodically loading from secondary storage
- Two types of operations can be performed on these streams :
  - Transformations – which create a new D-Stream from one or more parent streams
  - Output operations – which let the program write data to external systems.

D-Stream API

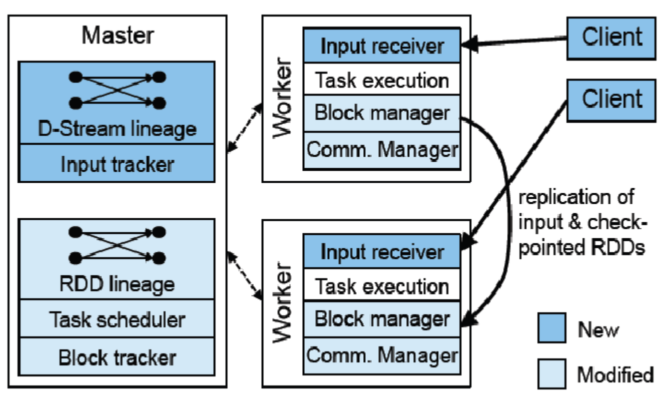
- D-Streams also provides several stateful transformations for computations spanning multiple intervals.
- Windowing : groups all the records from a sliding window of past time intervals into one RDD.
  - e.g. words.window("5s")
- Incremental aggregation : several variants of an incremental reduceByWindow operation
  - pairs.reduceByWindow("5s", (a, b) => a + b)

reduceByWindow execution



(a) Associative only (b) Associative & invertible

Components of Spark Streaming



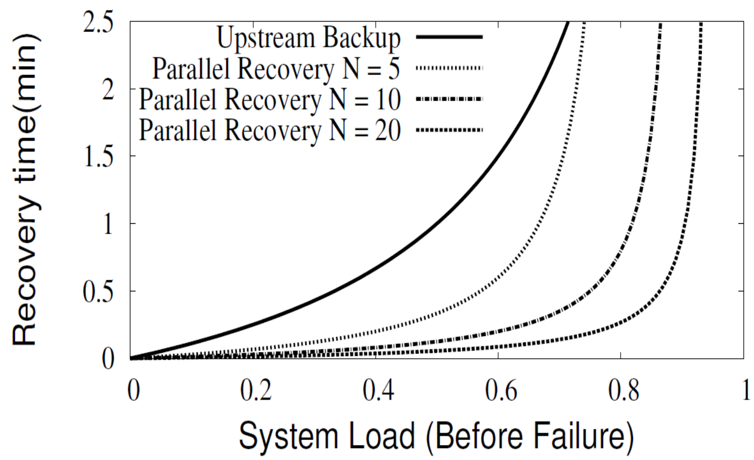
System Architecture

- D-Streams is implemented in a system called Spark Streaming
- This is based on a modified version of Spark processing engine from the same group (NSDI '12)
- Spark Streaming consists of three components
  - A **master** that tracks the D-Stream lineage graph and schedules tasks to compute new RDD partitions.
  - **Worker nodes** that receive data, store the partitions of input and computed RDDs, and execute tasks.
  - A **client library** used to send data into the system.

Fault and Straggler Recovery

- Parallel Recovery
  - All tasks which were running on a failed node are recomputed in parallel on other nodes
  - Motivation behind this : upstream backup takes long time to recover when the load is high
  - Parallel recovery catches up with the arriving stream much faster than upstream backup

Parallel recovery vs upstream backup



Fault and Straggler Recovery

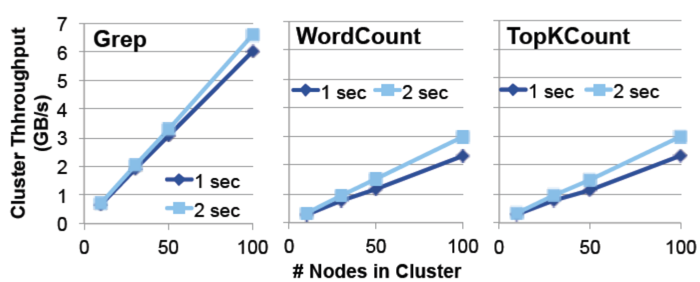
- Straggler Mitigation
  - a task runs more than 1.4x longer than the median task in its job stage is marked as slow
  - They show that this method works well enough to recover from stragglers within a second.
- Master Recovery
  - At the start of each interval the current state of computation is written into stable storage.
  - Workers connect to the new master when it comes up and inform it of their RDD partitions

Evaluation

- Spark streaming was evaluated using three applications :
  - Grep, which finds the number of input strings matching a pattern
  - Word- Count, which performs a sliding window count over 30s
  - TopKCount, which finds the k most frequent words over the past 30s
- These applications were run on “m1.xlarge” nodes on Amazon EC2, each with 4 cores and 15 GB RAM

PURDUE UNIVERSITY

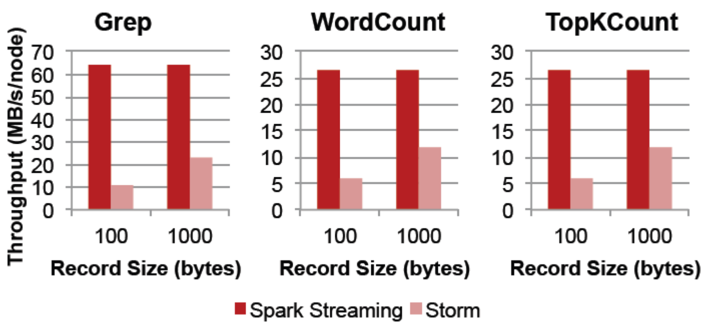
Results



Maximum throughput attainable under a given latency bound (1 s or 2 s) by Spark Streaming

PURDUE UNIVERSITY

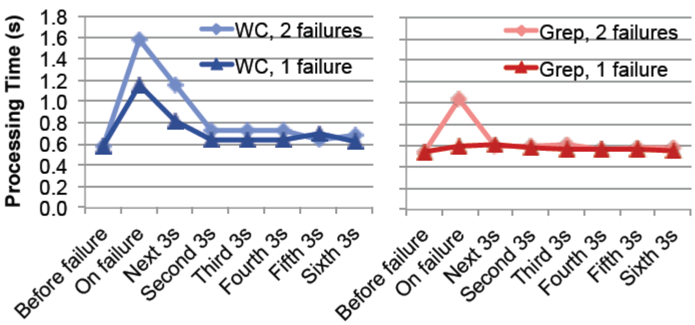
Results



Throughput vs Storm on 30 nodes

PURDUE UNIVERSITY

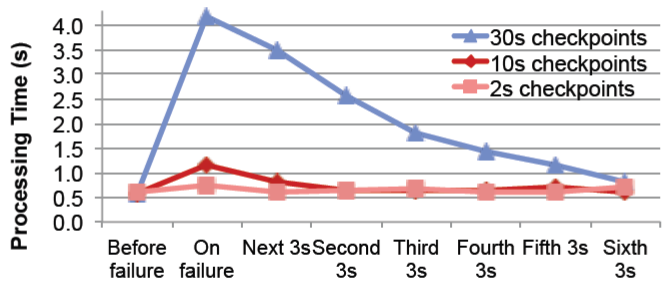
Results



Interval processing times for WordCount(WC) and Grep under failures

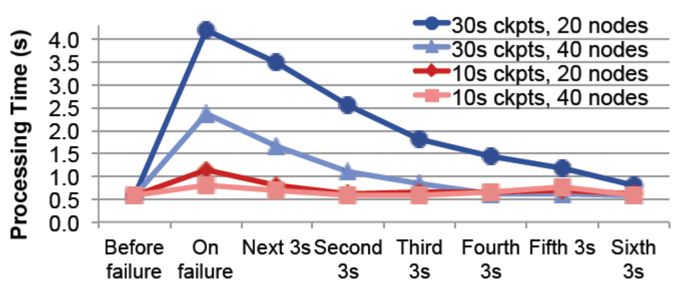
PURDUE UNIVERSITY

Results



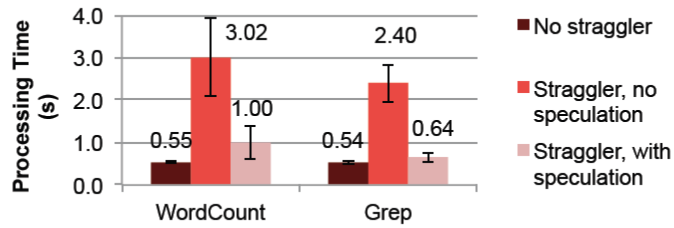
Effect of checkpoint in WordCount

Results



Recovery of WordCount on 20 & 40 nodes

Results



Processing time of intervals in Grep & WordCount in normal operation as well as in the presence of a straggler, with and without speculation

Conclusion

- By breaking computations into short, deterministic tasks and storing state in lineage-based data structures (RDDs), Dstreams can use powerful recovery mechanisms.
- D-Streams has a fixed minimum latency due to batching data. However the show that the total delay of 1-2 seconds is still tolerable for many real world uses

Thanks

Questions?