A Presentation About:
Discretized Streams: A Fault-Tolerant Model for Scalable Stream Processing

Siyuan (Jack) He

9/16/2015

Motivations

There is a demand for “real time” big data applications with the following requirements

- Data arrives in real time (online)
- High throughput
- Low Latency
- Fault tolerance
- Stagger mitigation
- Highly efficient

Current solutions are bad!

Current solutions

Apache Storm (developed by Twitter)

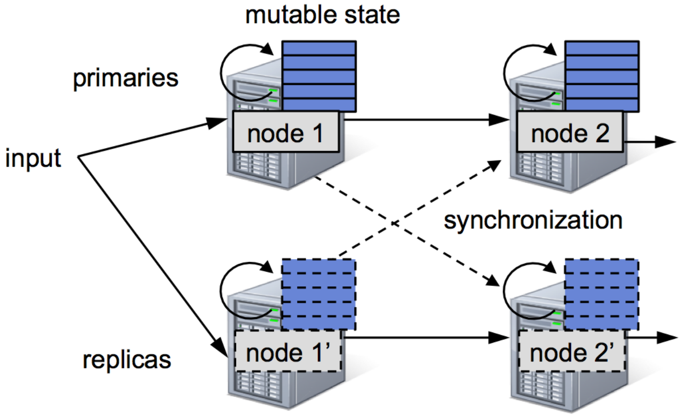
Apache S4 (developed by Yahoo)

Streaming databases (Borealis, Incoop, etc)

- Record-at-a-time processing model
- Long running stateful operators

Their problems

- Faults and Staggers
  - Normally uses replication, 2X the resources
  - Minimum or none stagger handling
- Consistency
  - Hard to reason about as different node might be processing data arrived at different times
- Unification with batch processing
  - These are event-driven systems, totally different from batch
  - Hard to combine streaming data with historical data

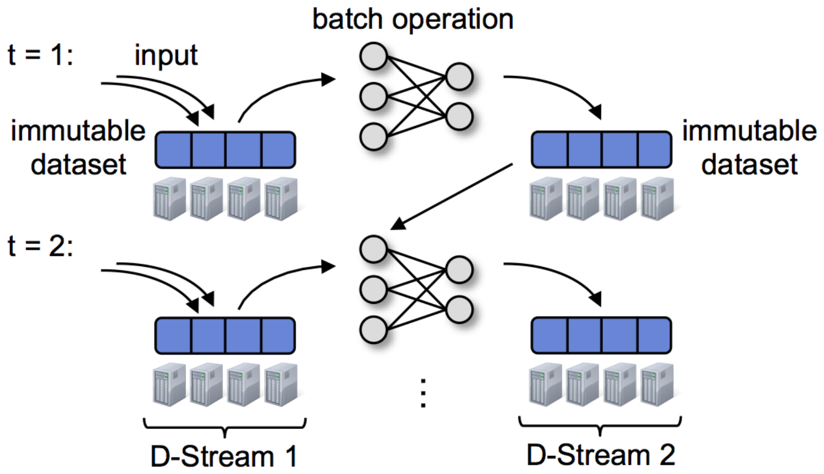


Discretized Streams - The new solution

A “Pseudo” streaming framework

Accumulate events over a short period of time and compute them in batches

Use existing batch processing frameworks (Mapreduce, Hadoop, Spark, etc) to solve the Fault Tolerance, Staggers, Consistency Problem, etc



Why this is good?

Adapt the new streaming problem into the existing batch processing problem

- Each batch is deterministic within - Consistency Solved!
- Fault tolerance and stagger handling are provided by existing framework - Solved (sort of)!
- High throughput in nature - It’s batch … - Solved!
- Easy integration between offline and online batch applications - Solved!

Wait, what about latency? - Why nobody did it before

Existing frameworks (Mapreduce, Hadoop) have high disk I/O overhead.

The minimum time (fixed cost) for running the smallest batch (say 1 record), is high (10-15 minutes!)

The “Pseudo” streaming implemented using these frameworks will be so fake.

Here comes the RDD! - Why it is possible

- In memory datasets eliminates the disk I/O overhead
- Super low start-up cost
- With some optimization, sub second latency can be achieved!
- And all the goodness of RDD (fault tolerance, stagger handling, etc)

The **Buffer** and **Compute** “streaming” pattern

- Incoming data is buffered at an RDD worker partition node
- Each node’s clock is synced with a Master
- Master schedules the compute action every fixed period of time
- During each “Compute” batch, we can view the system as a normal batch processing pipeline.

Optimizations

- Block placement
  - Pick RDD replica (partitions) based on load - Load balancing?
- Network communication
  - Asynchronous I/O
- Timestamp pipelining
  - New data can arrive while previous batch is running (so that there is no “no-serve” period), (Jack: otherwise we really can’t call this streaming...)
- Lineage cutoff
  - Forget the lineage after an RDD has been checkpointed.

Fault Tolerance and Stagger Handling

Fault Tolerance

- Parallel Recovery
- Recompute failed RDD's data in other partitions in parallel
- Catch up time (t_par) scales inversely proportional w.r.t number of machines (N), where (lambda) is the failed load

$$t_{\text{par}} = \frac{\lambda/N}{1 - \frac{N}{N-1}\lambda} \approx \frac{\lambda}{N(1-\lambda)}.$$

Stagger Mitiage

- RDD already have it
- Speculative backup copies of slow tasks
- 1.4x is the threshold (why????)

Evaluation

- Generally much faster than existing ones
- 60M records/second on 100 nodes at sub-second latency
- Good fault tolerance and stagger mitigation

What it is good for

- Streaming applications that require second level latency (e.g. not for high frequency trading)
- Interactive programming (same as Spark)
- Easy to use and easy to maintain

My thoughts

- Very clean design, use existing work, very good abstraction!
- An upgrade to the underlying batch processing framework can improve the performance of the streaming system as well
  - Say if we have a batch framework with millisecond start-up cost, then we might be able to achieve millisecond latency.
- Made possible because of RDD and Spark
- But, what about dependencies of one event over previous events? Those computation where data arrival sequence matther. Seems this one can treats each batch as a parallel dataset

Thank You!

Questions?