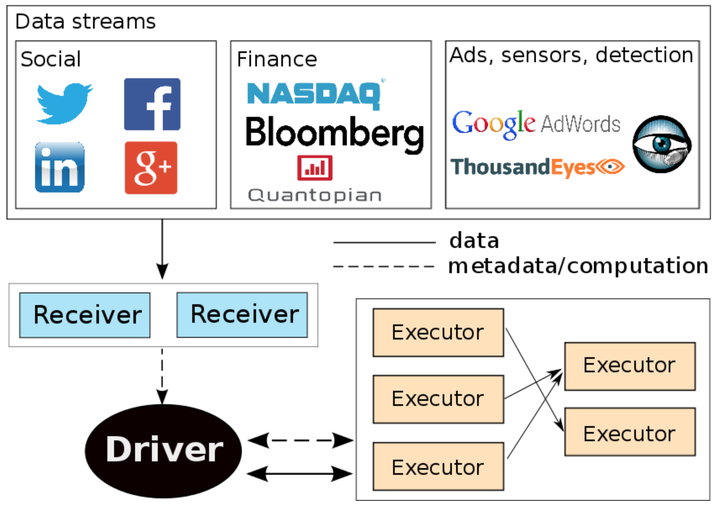
Optimizing Latency and Throughput Trade-offs in a Stream Processing System

João Carreira (joao@berkeley.edu) Jianneng Li (jiannengli@berkeley.edu)

Motivation and Objective

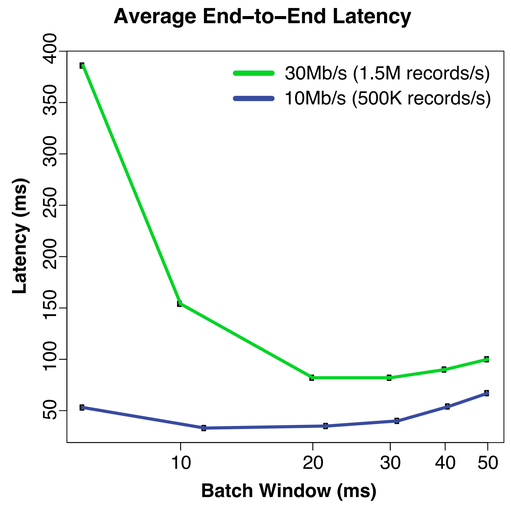
- Streaming systems can process data in two ways: record-by-record or micro-batch, favoring latency and throughput respectively
- Spark Streaming is a micro-batch stream processor built on top of Spark, and offers high throughput while targeting 0.5 to 2 seconds of latency
- Because Spark Streaming also offers several desirable properties such as fault tolerance, we explore what it takes for a system with this architecture to provide low latency while maintaining reasonable throughput

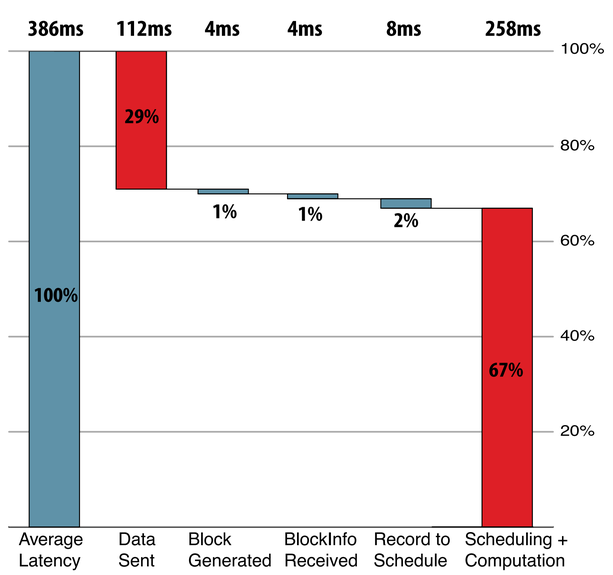
Spark Streaming Architecture

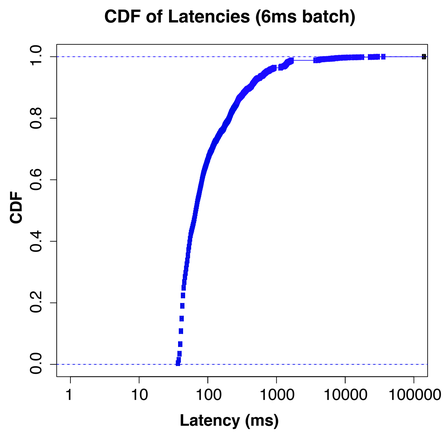


Experiments

- One node
  - Given a fixed throughput, 1) find end-to-end latencies of data for various batch windows and 2) find average breakdown as well as cumulative distribution functions (CDFs) of those latencies
  - For a configuration with stable end-to-end latencies, find average breakdown of times for running tasks
- Multiple nodes
  - Given a throughput and batch window, compare latencies and breakdowns per-node to that of one-node deployment

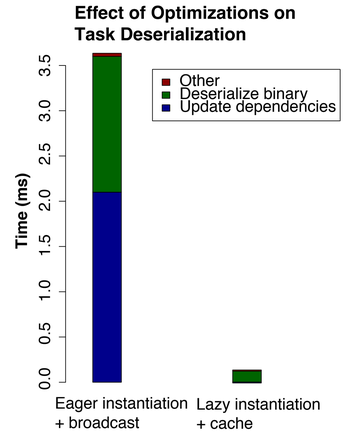


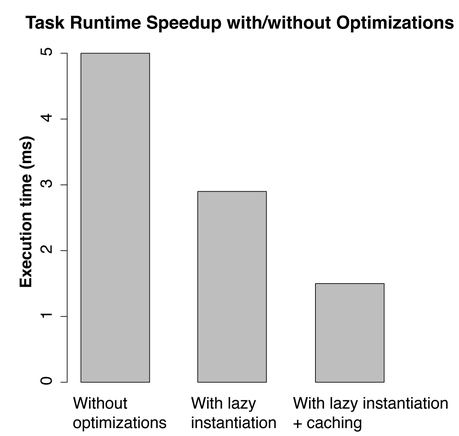




Implementation and Evaluation

- Reduce overhead in task deserialization
  - Lazily instantiate configuration object while updating dependencies
  - Cache task binary instead of sending same information every time
- Experimental decentralized task scheduler





Contributions

- We provide an evaluation of the performance of Spark Streaming across two major metrics: throughput and latency
- For a stream processor following the architecture of Spark Streaming, we analyze the performance bottlenecks and suggest potential areas of improvements
  - *Data ingest*: to receive input at a greater rate, use more receivers
  - *Task overhead*: we reduced the overhead of running a task in Spark Streaming through lazy instantiation of objects and caching
  - *Scheduling overhead*: we did not encounter scheduler bottlenecks in our experiments, but as the system scales out, a decentralized scheduler can be potentially useful
  - *Network overhead*: we did not find significant network bottleneck either in our experiments, but as the number of tasks per second increases, hardware technology such as InfiniBand can be used to decrease communication time between the driver and executors
- We show that for the best latency in Spark Streaming, the batch window should be slightly larger than the time it takes to process a batch