Building Robust, Adaptive Streaming Apps with Spark Streaming

Tathagata "TD" Das
@tathadas

Spark Summit East 2016

databricks

Who am I?

Project Management Committee (PMC) member of Spark

Started Spark Streaming in grad school - AMPLab, UC Berkeley

Current technical lead of Spark Streaming

Software engineer at Databricks

Streaming Apps in the Real World

Building a high volume stream processing system in production has many challenges

Fast and Scalable

Spark Streaming is fast, distributed and scalable by design!

Running in production at many places with large clusters and high data volumes

Streaming Apps in the Real World

Building a high volume stream processing system in production has many challenges

Fast and Scalable

Easy to program

Spark Streaming makes it easy to express complex streaming business logic

Interoperates with Spark RDDs, Spark SQL DataFrames/Datasets and MLlib

Streaming Apps in the Real World

Building a high volume stream processing system in production has many challenges

Fast and Scalable

Easy to program

Fault-tolerant

Spark Streaming is fully fault-tolerant and can provide end-to-end semantic guarantees

See my previous Spark Summit talk for more details

Streaming Apps in the Real World

Building a high volume stream processing system in production has many challenges

Fast and Scalable

Easy to program

Fault-tolerant

Adaptive

Focus of this talk

Adaptive Streaming Apps

Processing conditions can change dynamically

- Sudden surges in data rates
- Diurnal variations in processing load
- Unexpected slowdowns in downstream data stores

Streaming apps should be able to adapt accordingly

Backpressure

Make apps robust against data surges

Elastic Scaling

Make apps scale with load variations

Backpressure

Make apps robust
against data surges

Motivation

Stability condition for any streaming app

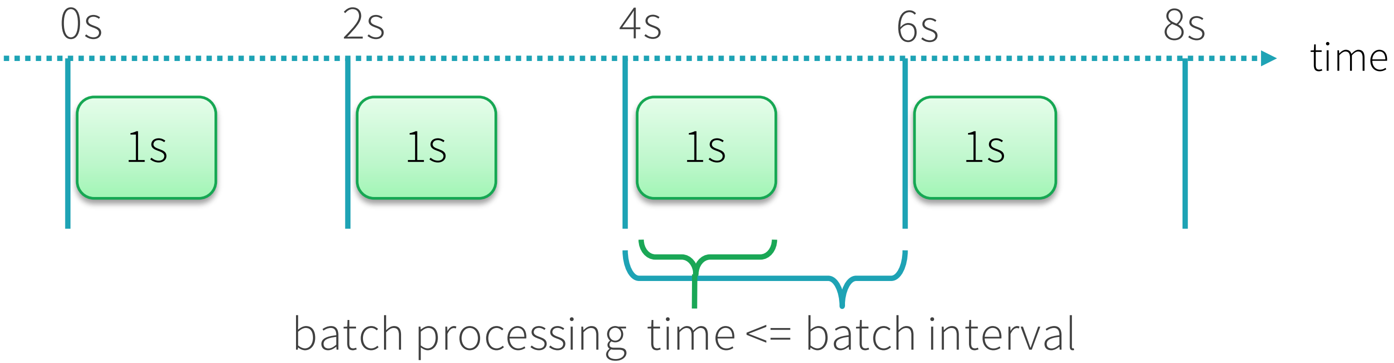
Receive data only as fast as the system can process it

Stability condition for Spark Streaming's "micro-batch" model

Finish processing previous batch before next one arrives

Stable micro-batch operation

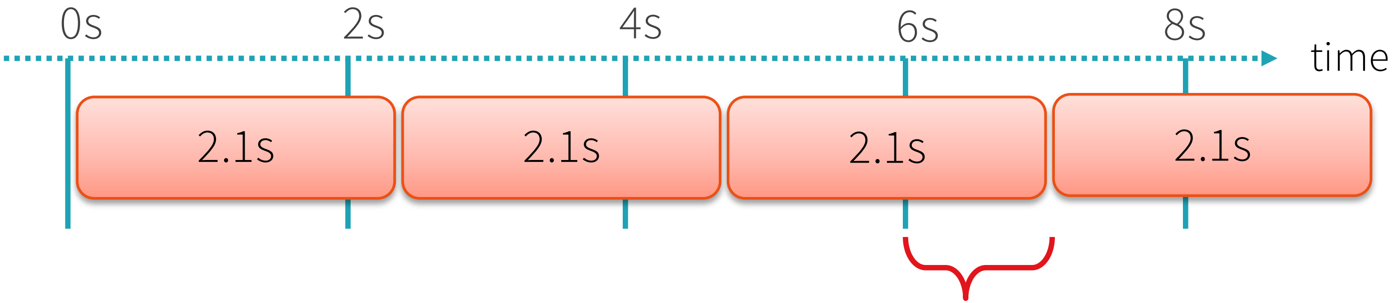
Spark Streaming runs micro-batches at fixed *batch intervals*



Previous batch is processed before next one arrives => stable

Unstable micro-batch operation

Spark Streaming runs micro-batches at fixed *batch intervals*



batch processing time > batch interval

scheduling delay builds up

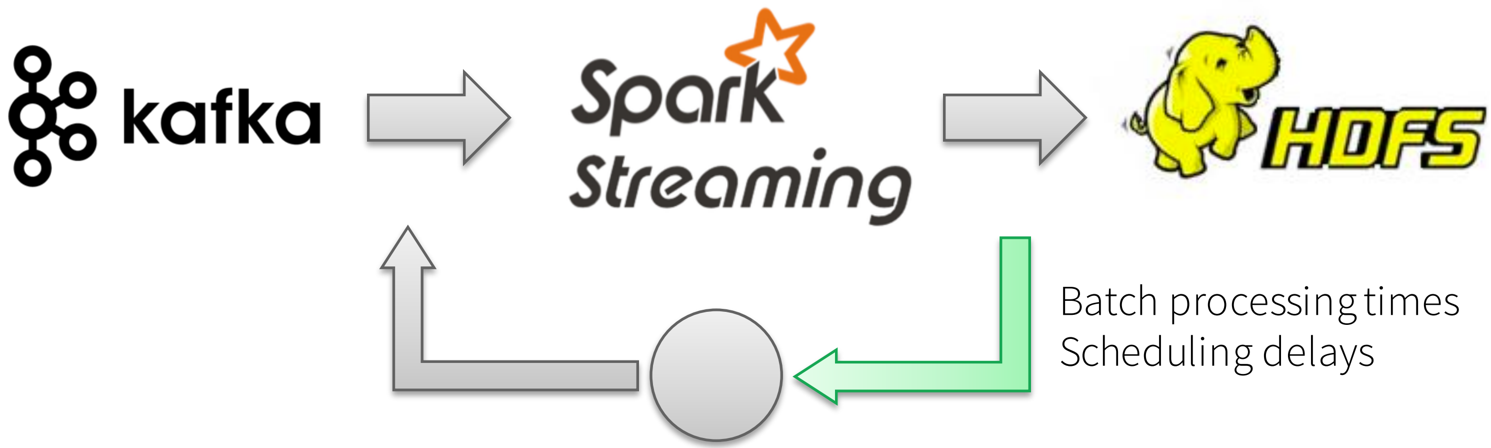
Batches continuously gets delayed and backlogged => unstable

databricks

Backpressure: Feedback Loop

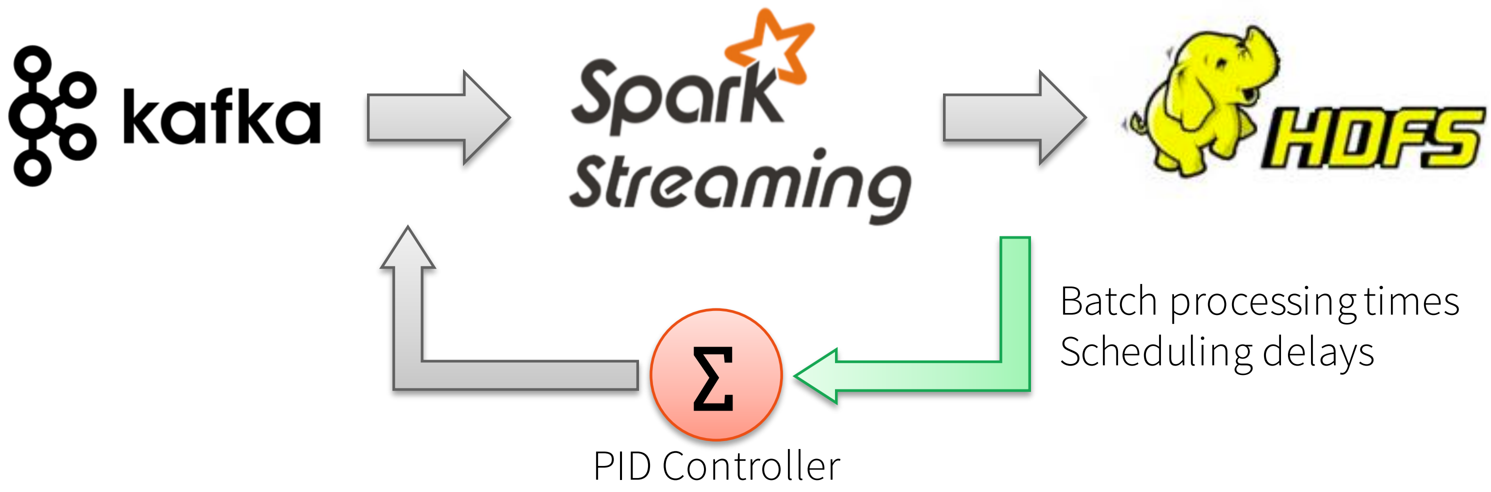
Backpressure introduces a feedback loop to dynamically adapt the system and avoid instability

Backpressure: Dynamic rate limiting



Batch processing times and scheduling delays used to continuously estimate current processing rates

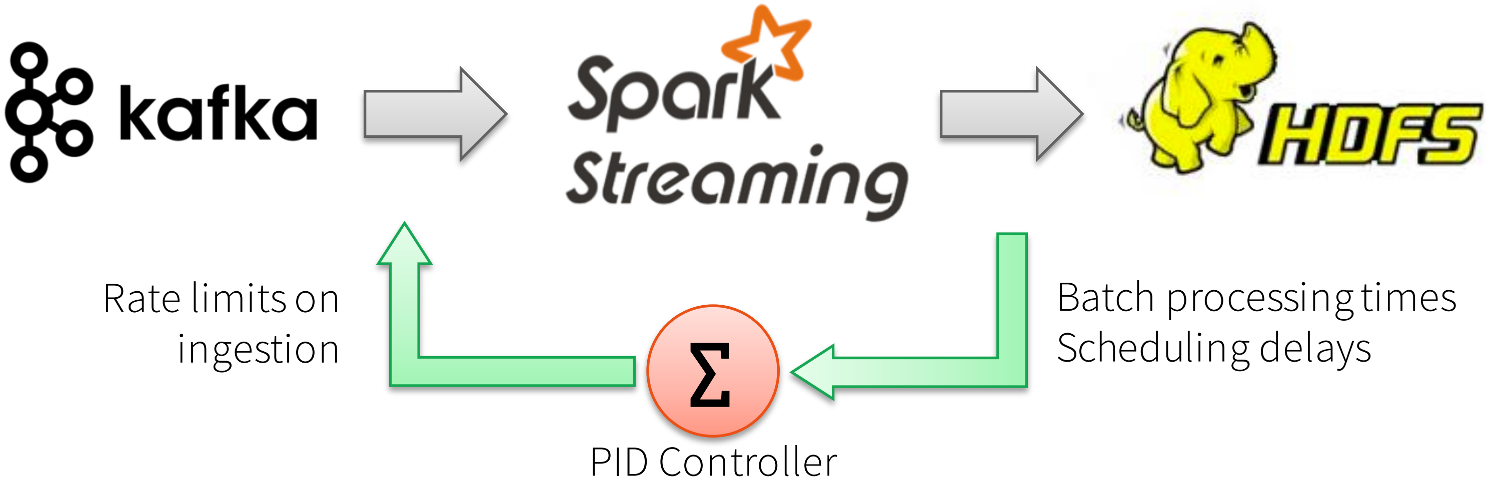
Backpressure: Dynamic rate limiting



Max stable processing rate estimated with PID Controller theory

Well known feedback mechanism used in industrial control systems

Backpressure: Dynamic rate limiting



Accordingly, the system dynamically adapts the limits on the data ingestion rates

Backpressure: Dynamic rate limiting

If HDFS ingestion slows down, processing times increase → SS lowers rate limits to slow down receiving from Kafka

Data buffered in Kafka, ensures Spark Streaming stays stable

Backpressure: Configuration

Available since Spark 1.5

Enabled through SparkConf, set

spark.streaming.backpressure.enabled = true

Elastic Scaling

Make apps scale with load variations

Elastic Scaling (aka Dynamic Allocation)

Scaling the number of Spark executors according to the load

Spark already supports Dynamic Allocation for batch jobs

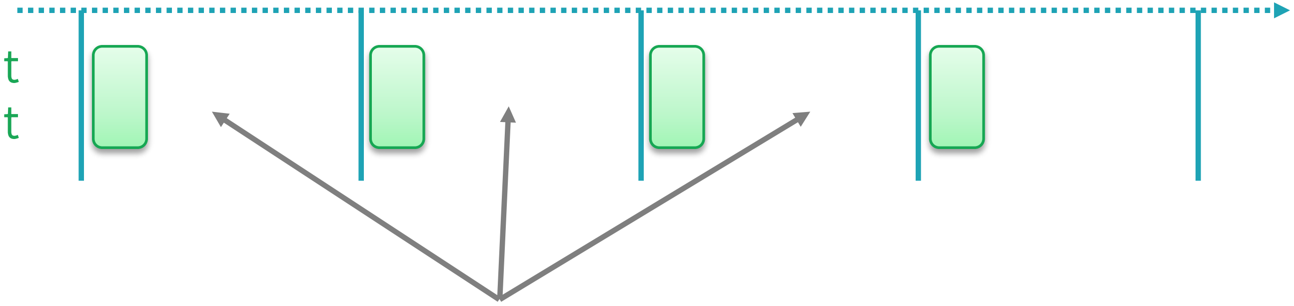
- Scale down if executors are idle
- Scale up if tasks are queueing up

Streaming "micro-batch" jobs need different scaling policy

- No executor is idle for a long time!

Scaling policy with Streaming

stable but inefficient

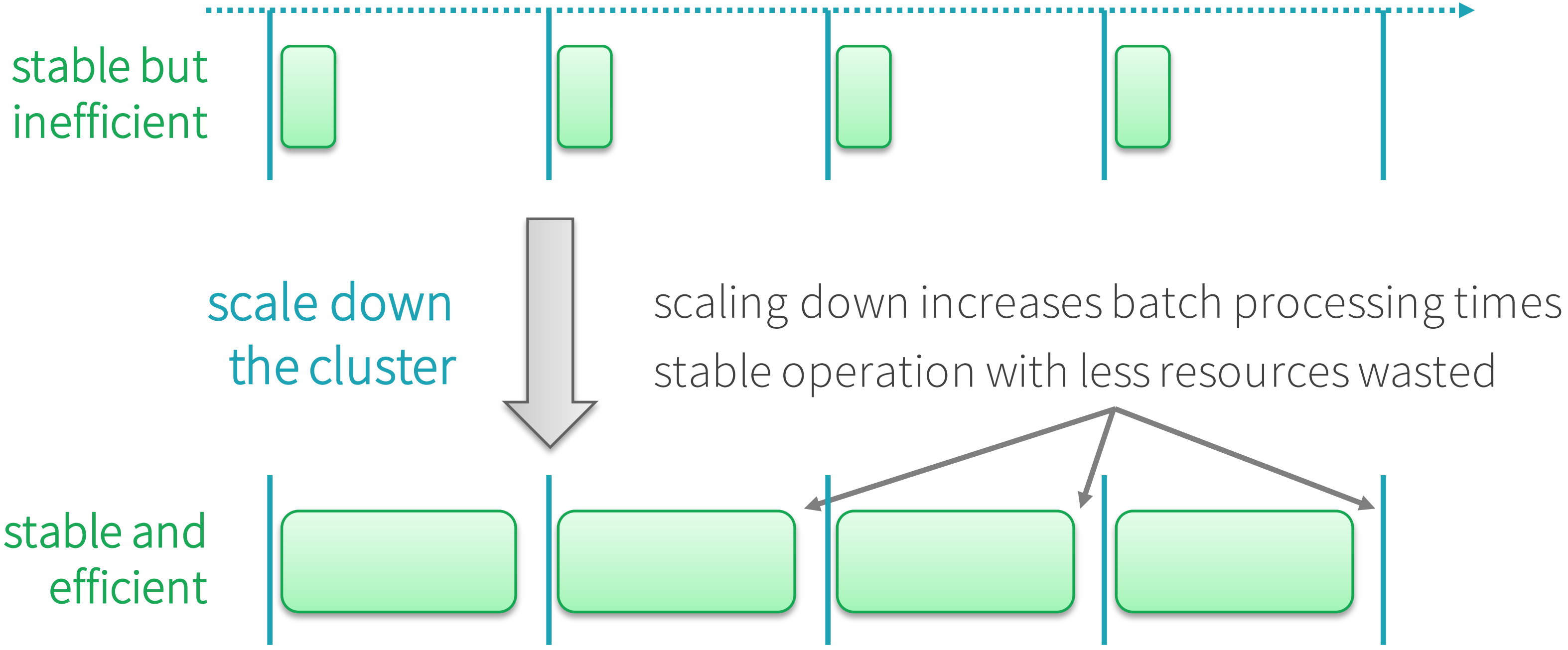


Load is very low

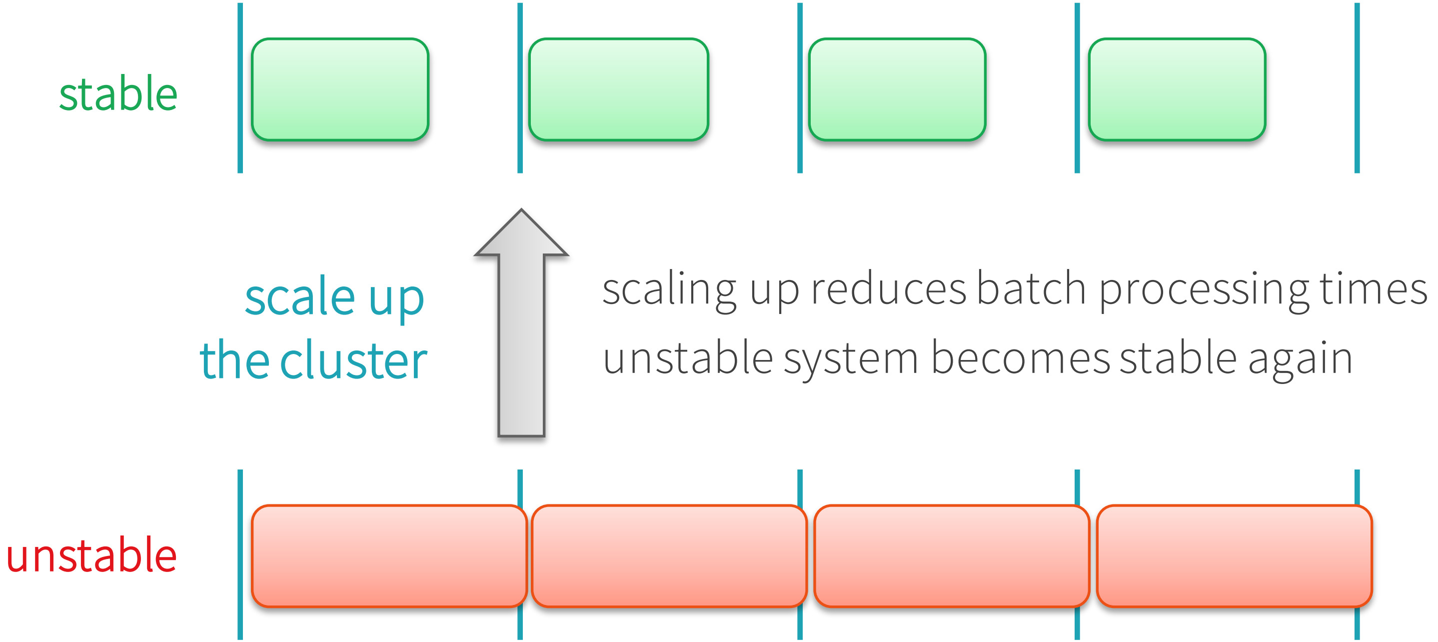
Lots idle time between batches

Cluster resources wasted

Scaling policy with Streaming



Scaling policy with Streaming



Elastic Scaling

If Kafka gets data faster than what backpressure allows → SS scales up cluster to increase processing rate

Data buffered in Kafka starts draining, allowing app adapt to any data rate

Elastic Scaling: Configuration

Will be available in Spark 2.0

Enabled through SparkConf, set

spark.streaming.dynamicAllocation.enabled = true

More parameters will be in the online programming guide

Elastic Scaling: Configuration

Make sure there is enough parallelism to take advantage of max cluster size

- # of partitions in reduce, join, etc.
- # of Kafka partitions
- # of receivers

Gives usual fault-tolerance guarantees with files, Kafka Direct, Kinesis, and receiver-based sources with WAL enabled

Backpressure

Elastic Scaling


Awesome
Adaptive
Apps

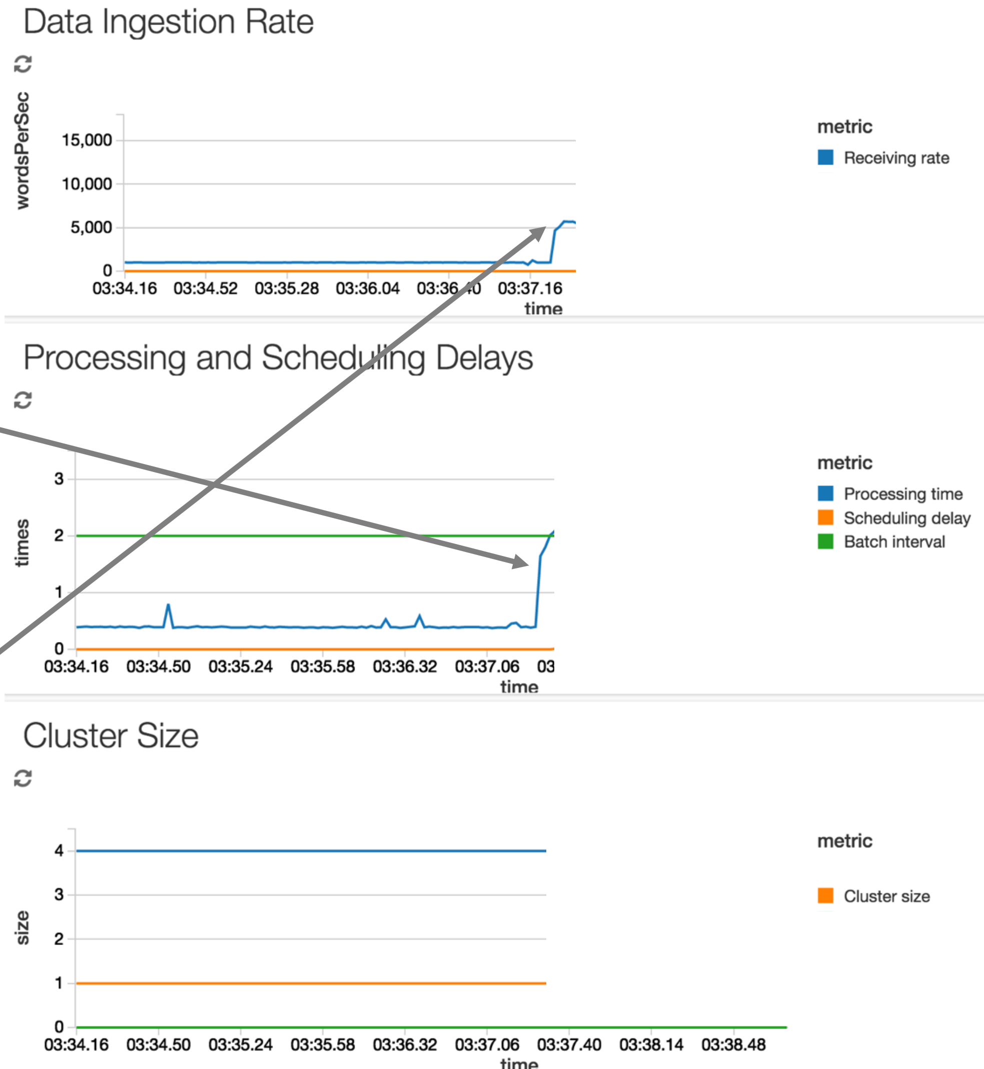
Demo

How app behaves when the data rate suddenly increases 20x

Demo

Processing time increases with data rate, until equal to batch interval

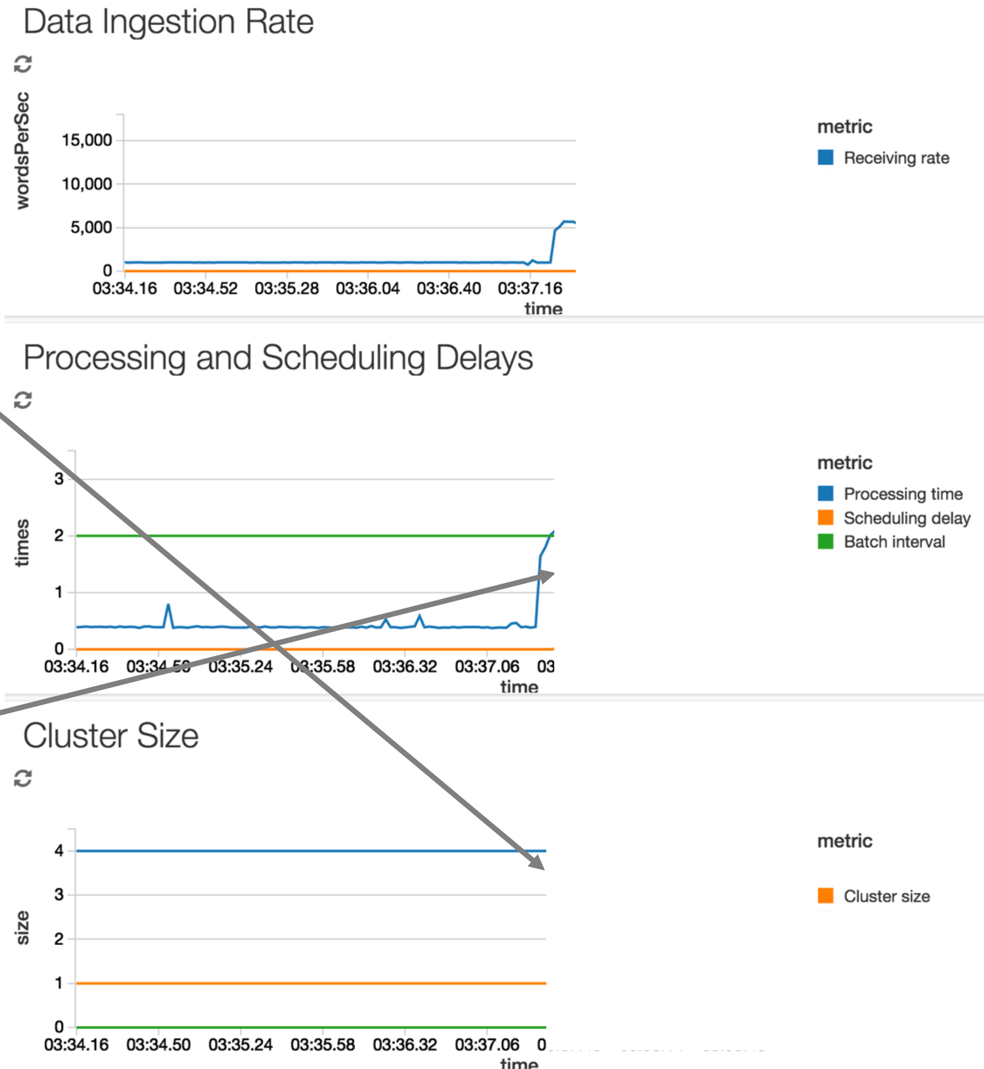
 limits the ingestion rate lower than 20k recs/sec to keep app stable



Demo

Elastic Scaling detects heavy load and increases cluster size

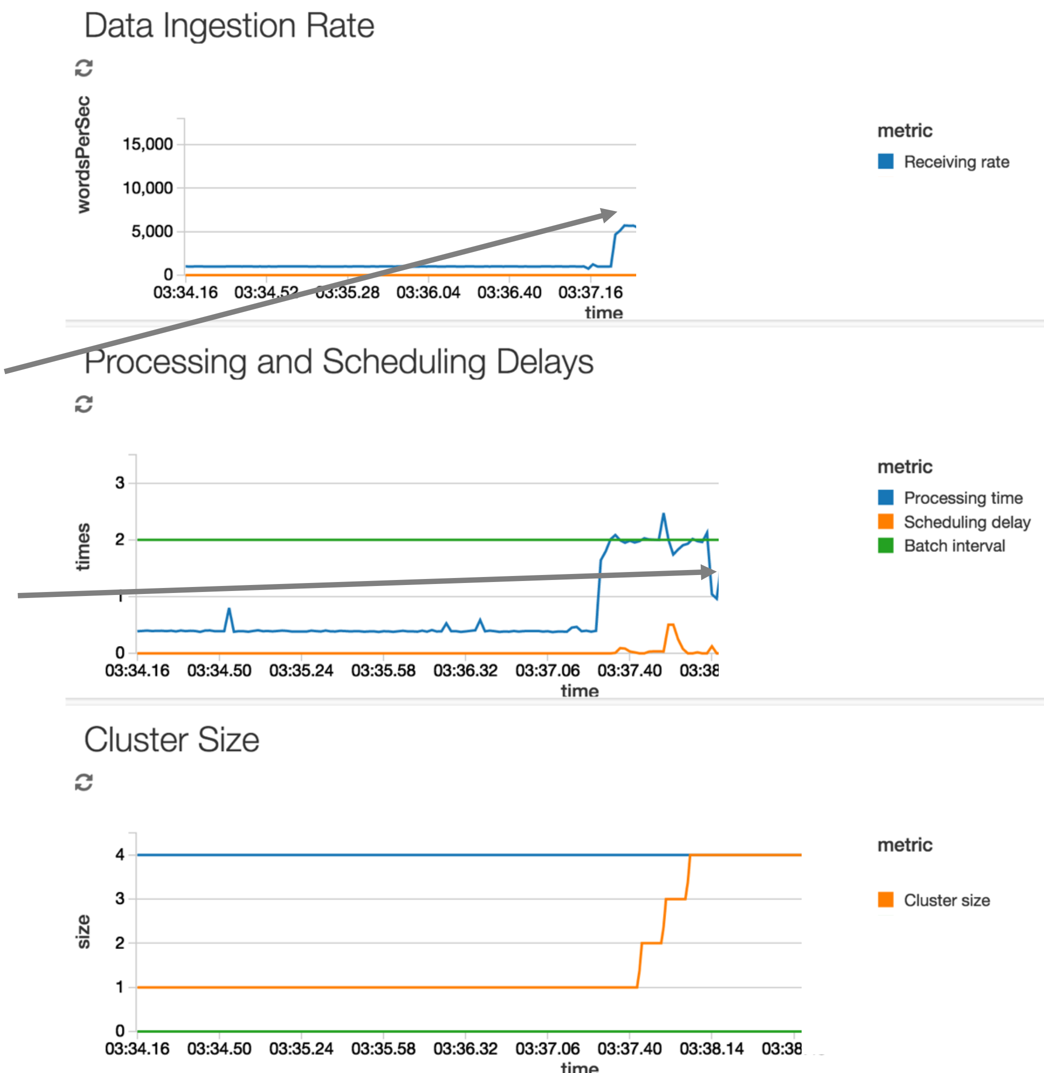
Processing times reduces as more resources available



Demo

Backpressure relaxes limits to allow higher ingestion rate

But still less than 20x as cluster is fully utilized



Takeaways

Backpressure: makes apps robust to sudden changes

Elastic Scaling: makes apps adapt to slower changes

Backpressure + Elastic Scaling =

Awesome Adaptive Apps