From pipelines to refineries: scaling big data applications

Tim Hunter
@timjhunter
Spark Summit Dublin 2017

databricks

About Me

- Tim Hunter
- Software engineer @ Databricks
- Ph.D. from UC Berkeley in Machine Learning
- Very early Spark user
- Contributor to MLlib
- Author of TensorFrames, GraphFrames, Deep Learning Pipelines

Introduction

- Spark 2.3 in the release process
- Spark 2.4 being thought about
- This is the time to discuss Spark 3
- This presentation is a personal perspective on a future Spark

Introduction

There is nothing more practical than a good theory.

James Maxwell

As Spark applications grow in complexity, what challenges lie ahead?

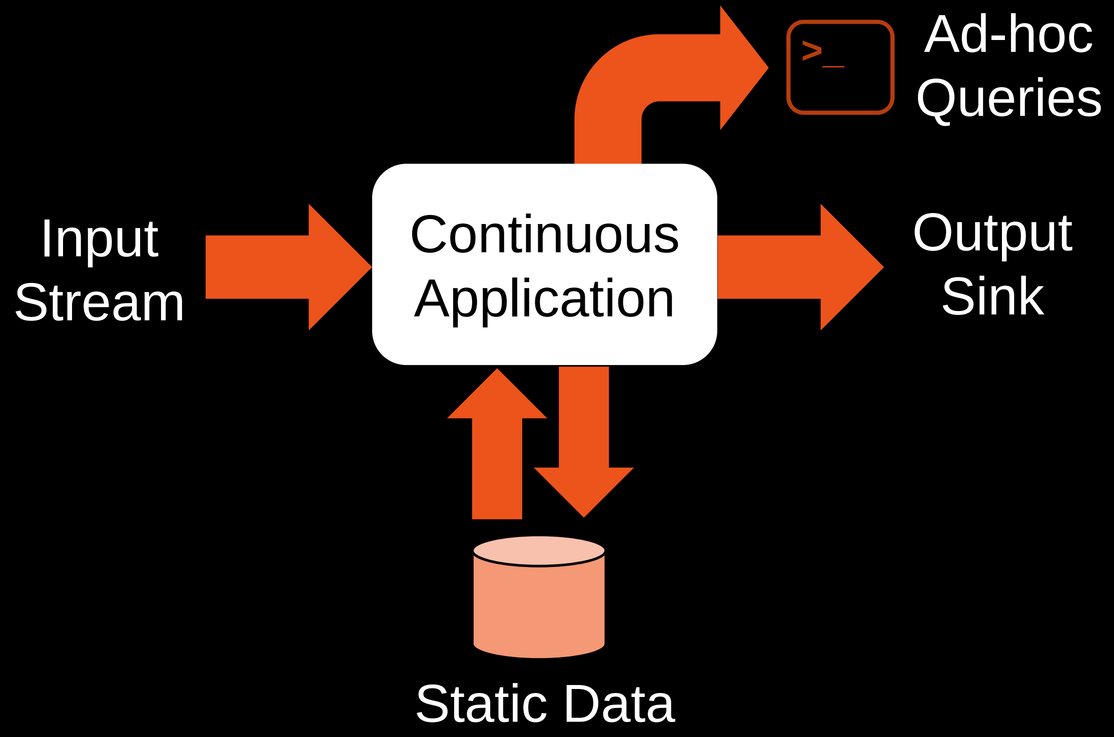
What are some good foundations for building big data frameworks?

Outline

- State of the union
  - What is good about Spark?
  - What are the trends?
- Classics to the rescue
  - Fighting the four horsemen of the datapocalypse
  - Laziness to the rescue
- From theory to practice
  - Making data processing great again

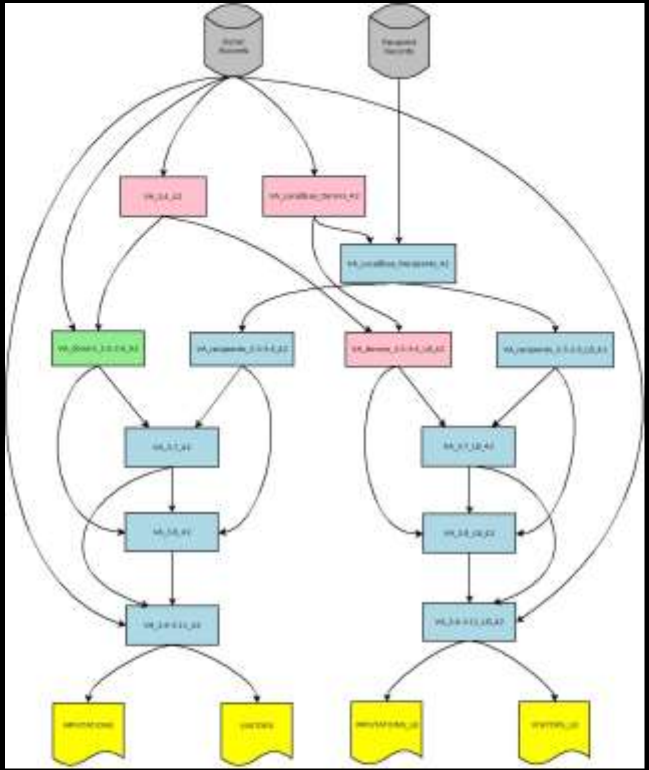
State of the union

- What we strive for



State of the union

- What we deal with:
  - Coordinating a few tasks



State of the union

- The (rapidly approaching) future
  - Hundreds of input sources
  - Thousands of concurrent requests
  - Mixing interactive, batch, streaming

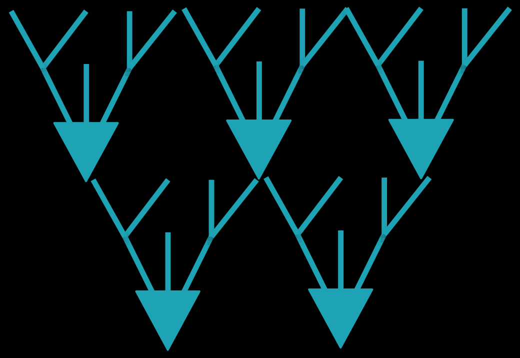
- How do we enable this?

The state of the union

- The image of a pipeline gives you the illusion of simplicity
  - One input and one output

- Current big data systems: the tree paradigm
  - Combine multiple inputs into a single output
  - The SQL paradigm
  - Followed by Spark

- A forest is more than a group of trees
  - Multiple inputs, multiple outputs
  - The DAG paradigm



databricks

The ideal big data processing system:

- *Scalability*
  - in quantity (big data) and diversity (lots of sources)
- *Chaining*
  - express the dependencies between the datasets
- *Composition*
  - assemble more complex programs out of simpler ones

- *Determinism*
  - given a set of input data, the output should be unique*

databricks

How is Spark faring so far?

- You *can* do it, but it is not easy

What can go wrong with this program?

```
all_clicks = session.read.json("/tables/clicks/year=2017")
all_clicks.cache()
max_session_duration = all_clicks("session_duration").max()
top_sessions = all_clicks.filter(
        all_clicks("session_duration") >= 0.9 * max_session_duration)
top_ad_served = top_sessions("ad_idd")
top_ad_served.write.parquet("/output_tables/top_ads")
```

leak

a few hours…

typo

missing directory

The 4 horsemen of the datapocalypse

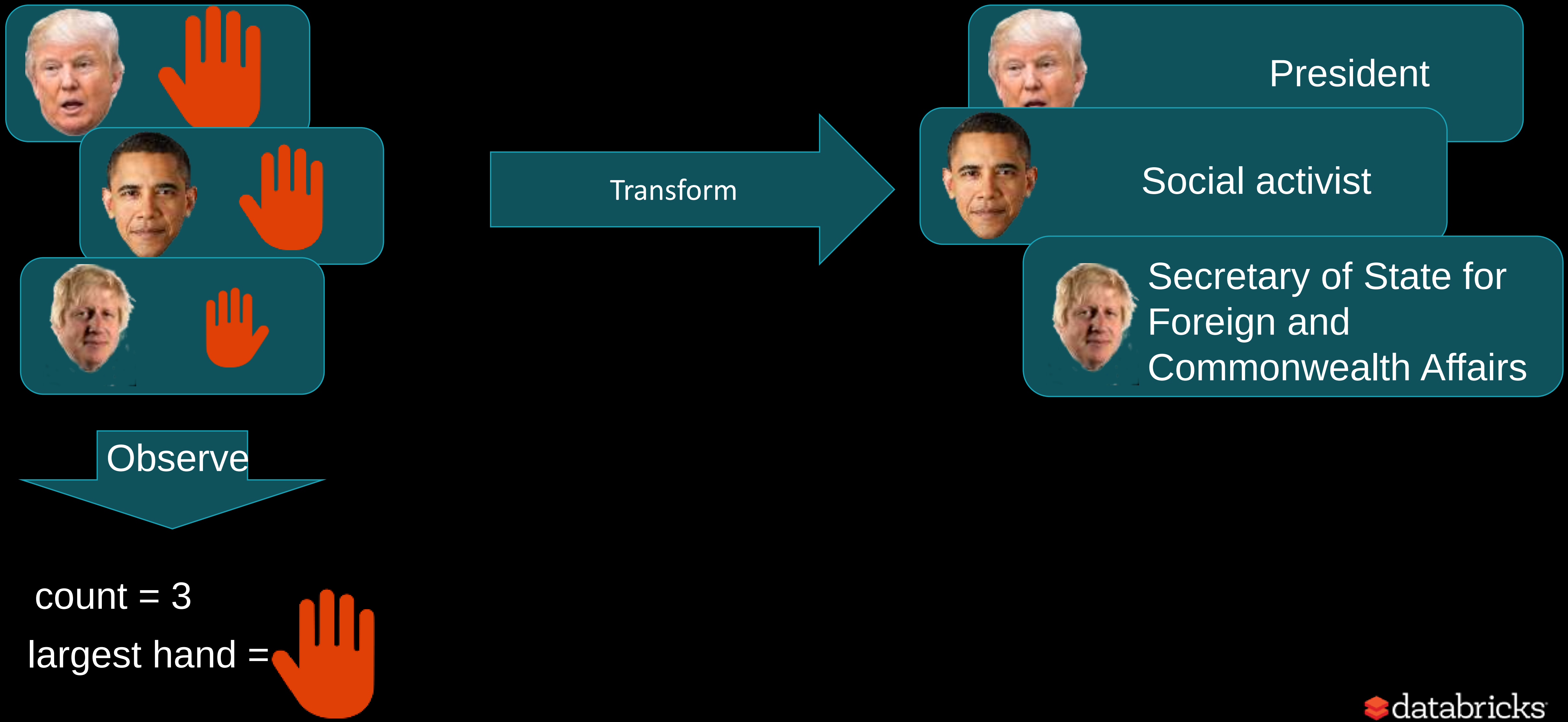
- Typing (schema) mismatch
- Missing source or sink
- Resource leak
- Eager evaluation

Classics to the rescue

Theoretical foundations for a data system

- A dataset is a collection of elements, all of the same type
  - Scala: Dataset[T]
- Principle: the content of a dataset cannot be accessed directly
  - A dataset can be *queried*
- An observable is a single element, with a type
  - intuition: dataset with a single row
  - Scala: Observable[T]

Theoretical foundations for a data system



Theoretical foundations for a data system

- *Principle*: the observation only depends on the content of the dataset
  - You cannot observe partitions, ordering of elements, location on disk, etc.

- Mathematical consequence: all reduction operations on datasets are monoids:
  - $f(A \cup B) = f(A) + f(B) = f(B) + f(A)$
  - $f(\text{empty}) = 0$

Theoretical foundations for a data system

- *Principle*: closed world assumption
  - All the effects are modeled within the framework
  - The inputs and the transforms are sufficient to generate the outputs

- Practical consequence: strong checks and sci-fi optimizations

Examples of operations

- They are what you expect:
  - Dataset[Int] : a dataset of integers
  - Observable[Int] : an observation on a dataset
- max: Dataset[Int] => Observable[Int]
- collect: Dataset[Int] => Observable[List[Int]]

Karps

- An implementation of these principles on top of Spark
- It outputs a graph of logical plans for Spark (or other systems)
- Makes a number of correctness checks for your program
- *Automatically converts (a subset of) Pandas programs to Spark.*

Demo 1

Enabling complex data programs

- Lazy construction of very complex programs
- Most operations in Spark can be translated to a small set of primitive actions with well-defined composition rules.
- The optimizer can then rewrite the program without changing the outcome
- Optimizations can leverage further SQL optimizations

Demo 2

Future directions

- More complete python (pandas) interface
- I/O in Python
- Finish GroupBy (cool stuff ahead)
- Tracing & Profiling
- SQL (simple and cool stuff to do in this area)

Conclusion: trends in data processing

- How to manage the complexity of data flows?
- Taking inspiration from the functional world
- Spark provides solid foundation
- Laziness, declarative APIs alleviate complexity

Trying this demo

- https://github.com/tjhunter/karps

- Notebooks:
  - https://github.com/tjhunter/karps/tree/master/notebooks

Thank You

Dealing with In and Out

- The only type of I/O: read and write datasets
- This is an observable
- Operations are deterministic + results are cached
  - -> only recompute when the data changes

- Demo

Example: Caching