Distributed Computing

and Spark

Veljko Krunic

Disclaimer

- These slides are intended for training purposes only, with additional explanation to be provided by the trainer
- Applying this material to real world projects (or based only on the context of this slides) is to be done only at your own risk
  - The author will not share any responsibility for the result of an implementation unless he was part of the implementation team

Distributed Computing

Why?

- Because data is getting too big for the single machine
- 1h of video (depending of compression format) can be in the range 1.8 GB/h to 40+GB/h
- It is estimated that
  - In 2012 every day we generated 2.5 exabytes of data
  - By 2020 we would have 44 zettabytes of data

1 ZB = 1000^7 bytes = 10^{21} bytes = 1 000 000 000 000 000 000 000 bytes = 1000 exabytes = 1 million petabytes = 1 billion terabytes = 1 trillion gigabytes.

Distributed Computing

- Divide work across a set of different machines
- Some of the basic problems are balancing
  - Time of communication
  - Time to do the work

Coordinating Work

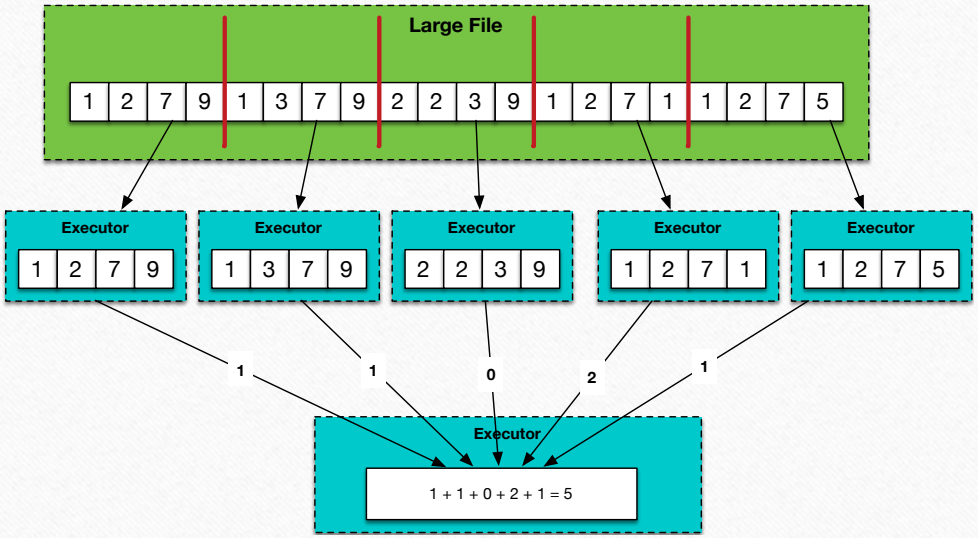
- Multiple threads of work can collide
  - https://en.wikipedia.org/wiki/Isolation_(database_systems)#Read_phenomena
- Very old problem
  - Well studied in database context

Problems That Are Easy to Distribute

- So called "embarrassingly parallel problems"
- Examples
  - Distributed search for number
  - Join of small table with big one

Distributed search

- Find all occurrences of “1” in a given large file
- Easy
  - Divide the file among machines
  - Search within each machine
  - Report your result



Lets Expand…

- Suppose you are searching for multiple values
  - {1, 2, 3, 4}
- Can we maintain the same approach?

Can We Do Joins That Way?

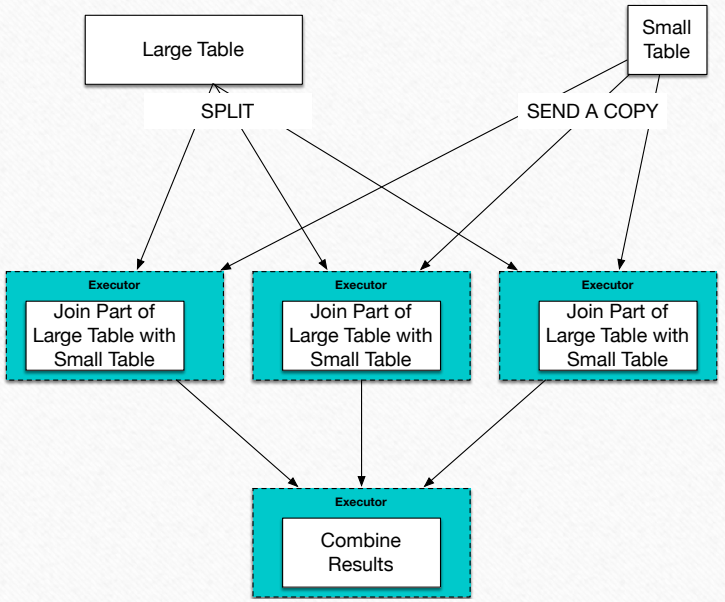
- Suppose that table *small_table* is small

```
SELECT big_table.name
FROM small_table, big_table
WHERE small_table.id = big_table.id
```

- Can we use the same approach?
- Yes, if the *small_table* is small enough

Broadcast

- Send the *small_table* to all of your machines
- Divide the *big_table* across the machines
- Join on each machine
- Aggregate results



How Big…

- How big can a *small_table* be?
  - Must fit in memory
- What if it doesn't?
  - Then you must send parts of both tables to both machines
  - The details of that process don't matter at this time
  - But this communication is an example of what we will call *shuffle* later
- Some operations require more communication then others

Spark

What Is Spark?

- Unified processing engine that can analyze big data
- API for all of the following is similar
  - SQL/Dataframe
  - Streaming
  - Machine Learning
  - Graph

Important Ideas

Important Ideas

- In memory computation
- Make programming a distributed system easier
  - Make coordination easier
  - Simple APIs
- Behind the scene optimizations
- Batteries included
  - In the form of a machine learning library

In Memory Computation

- Computation is done in memory
- The results of one computation can be cached and used by other computations
- Memory usage is optimized
  - Tungstent etc.

Make Coordination Easier

- Immutability
  - If the underlying data can't change, read anomalies disappear
- But what happens when we need to change values?
  - We create new objects with the changed values
  - And use those newly created objects

Optimizations

- A lot of work went into optimizing both memory usage and speed
  - Catalyst
  - Tungsten
- One of things that helps optimization is *lazy execution*

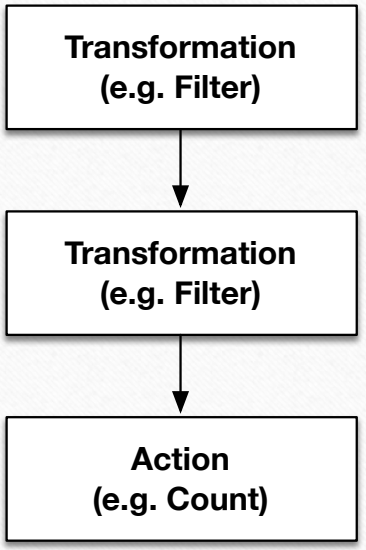
Lazy Execution

- Commands that are not executed *until* you need its results
  - Transformations
  - Actions
- Helps in optimization
  - Informally

df1 = df[$"x" < 3]

df2 = df1[$"x" > 1]

  - Becomes df[$"x" < 3 && $"x" > 1]



Execution Happens Here

Failure Recovery

- Transformation only adds info to the *lineage graph*
- Action triggers graph execution
- The recovery mechanism is simply re-executing your lineage graph!

Summary

- There are tradeoffs between distribution and computation
- Spark uses immutability and lazy execution