Recommendation System Optimization based on Spark

Insight Data Engineering Fellow Project

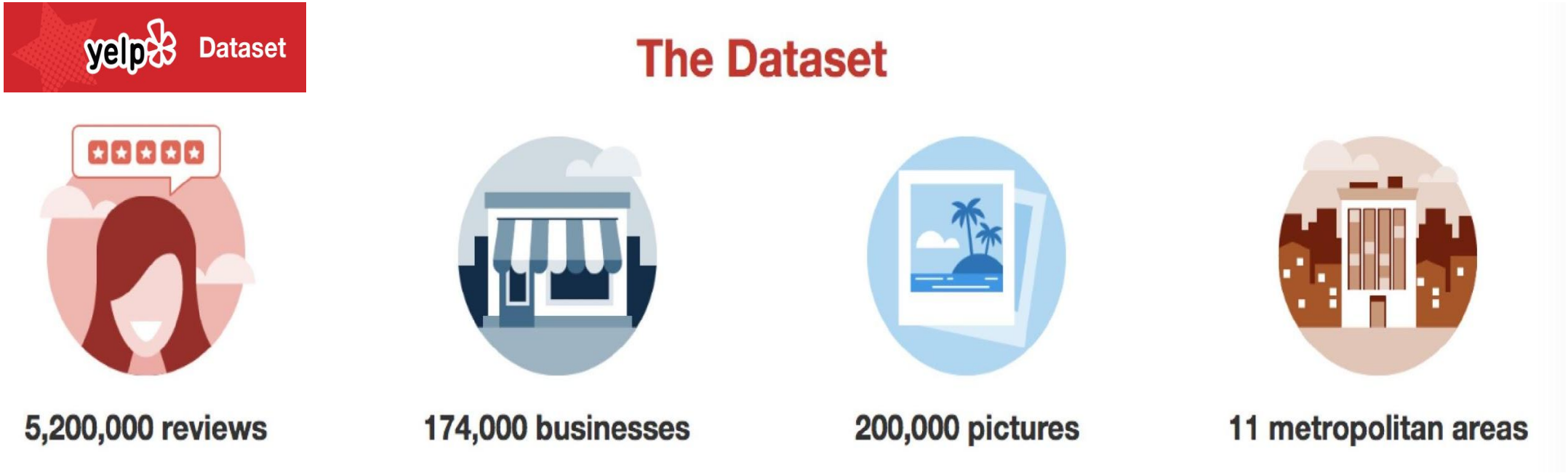
Xiaojin (Ruby) Liu

May. 18, 2018

Motivation

Dataset

- Yelp Dataset
  - Started with a small subset of Yelp Dataset.
  - Scale up to Whole dataset
  - Simulated more data



Algorithm

- Item-based Collaborative Filtering

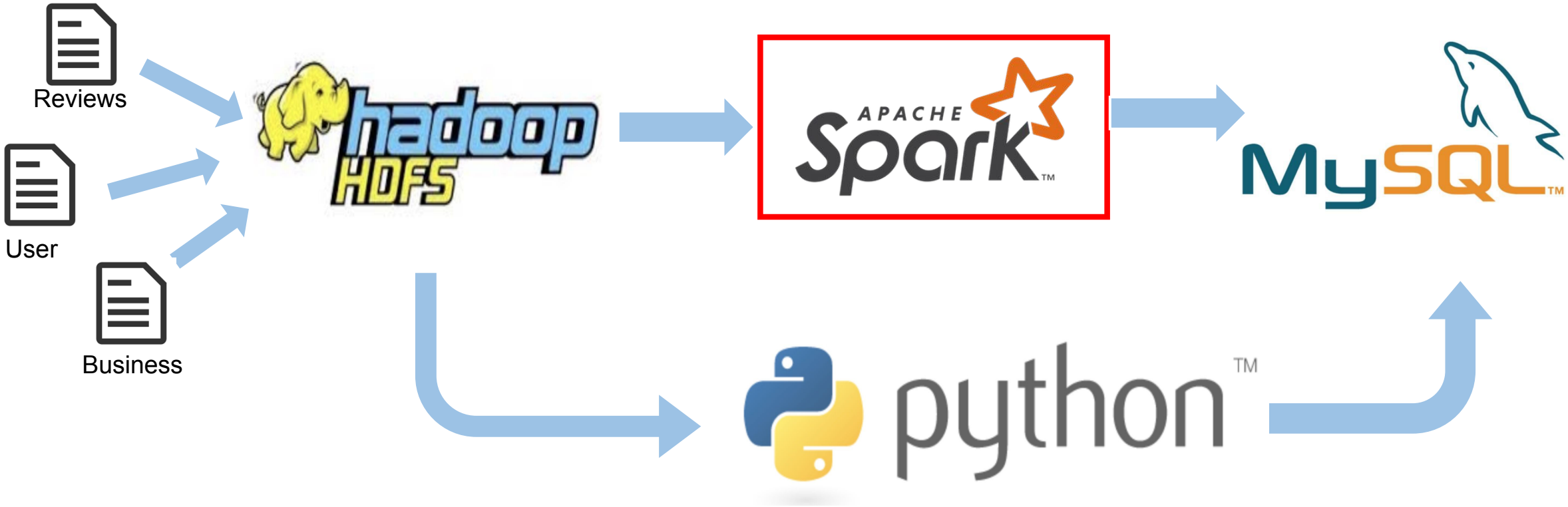
Mr. Doggy

Like

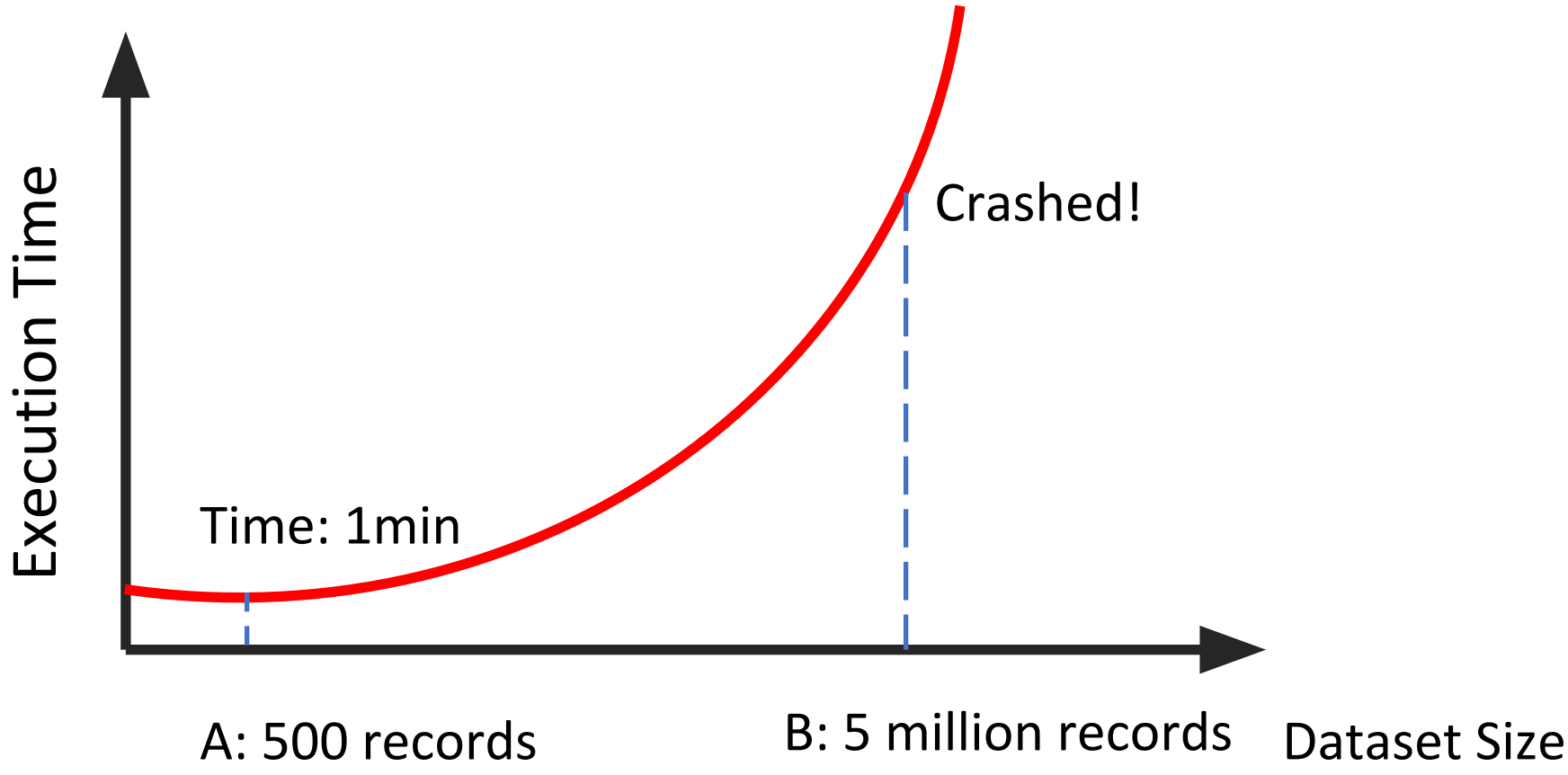
Recommend

Similar

Data Flow


Reviews
User
Business
hadoop
HDFS
APACHE
Spark
MySQL
python

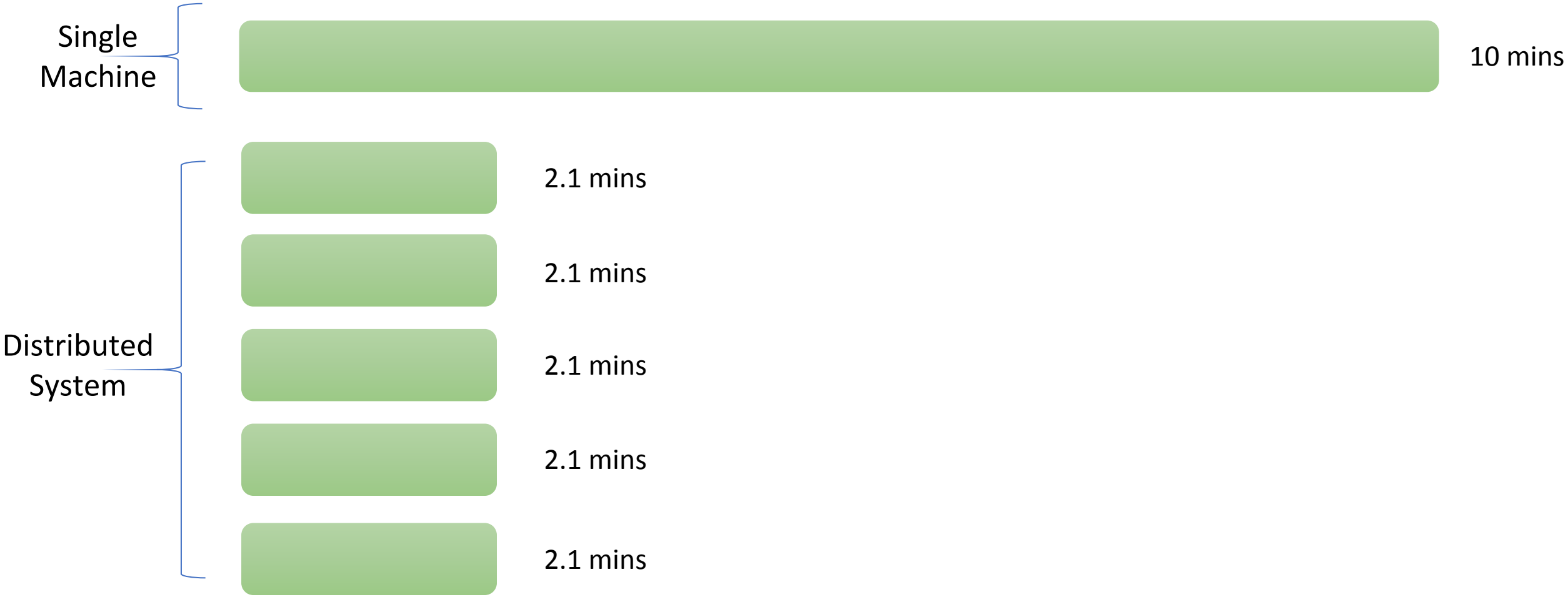
Data Engineering Challenge



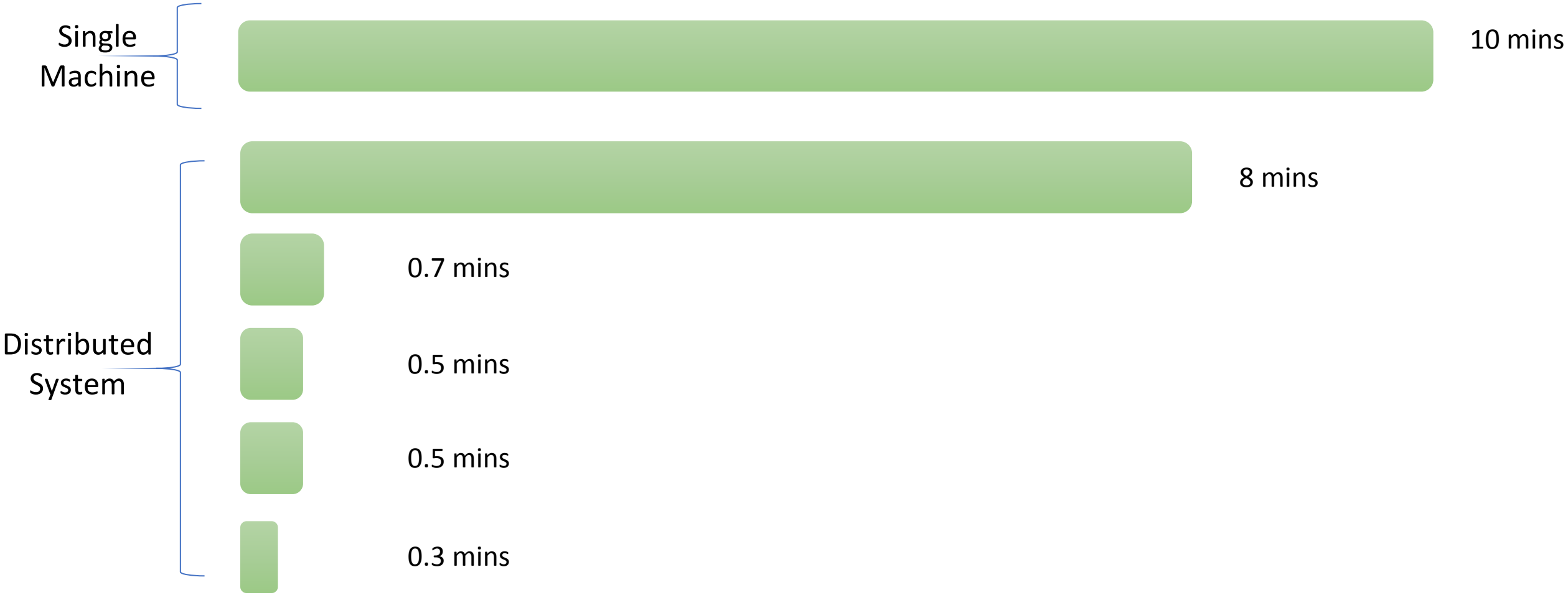
Optimization – Step One

- Data Preprocessing
  - Change business_id and user_id from string to Integer.
- Tuning Spark
  - Increased the value of “Spark.memory.fraction” from 0.6(default) to 0.75 to reduce disk spill.
  - Decreased the value of “spark.memory.storageFraction” from 0.5 to 0.35, save memory for spark execution.

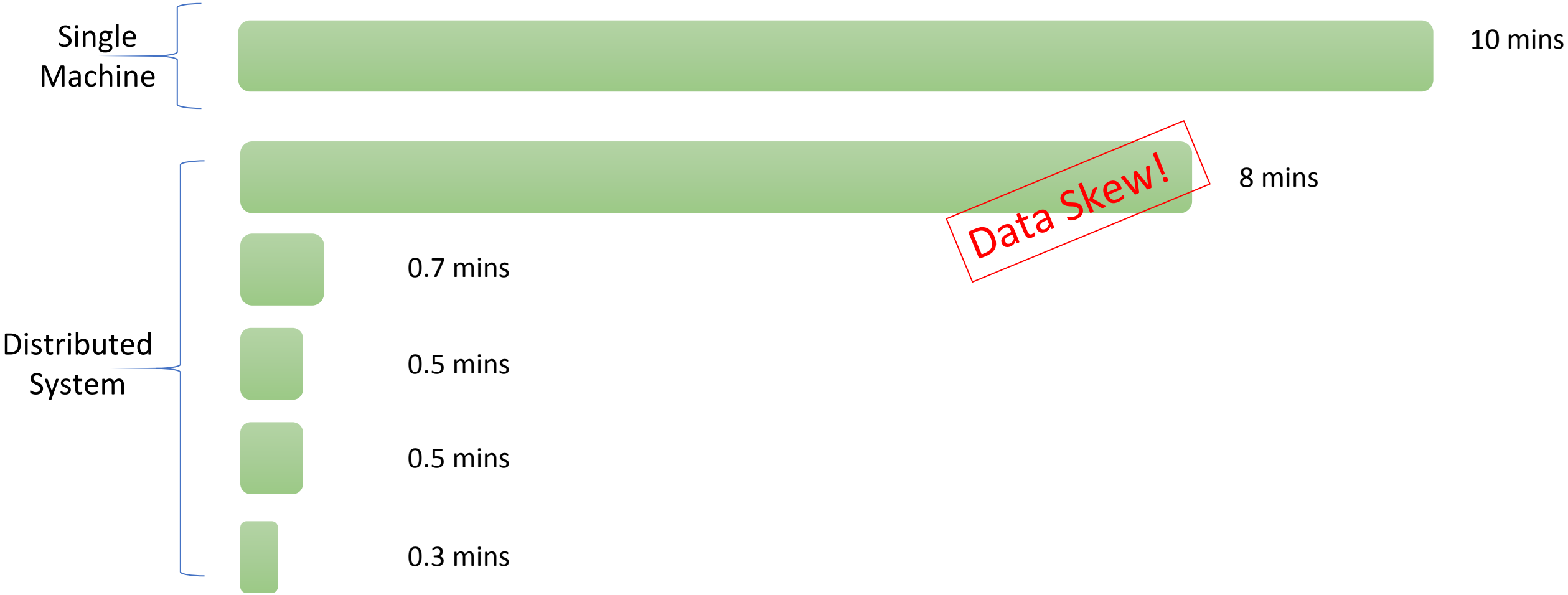
Ideal execution time of distributed system



Real execution time of distributed system



Real execution time of distributed system



Why Data Skew?

-- 20% business provide 80% reviews

Key	Value	
Business	User	Star
Starbucks	Tom	5
Five Guys	Tom	4
Starbucks	Jim	3
Qdoba	John	2
Five Guys	Alice	5
Starbucks	Romia	2
Starbucks	Juliet	5
Qdoba	Sam	3
Starbucks	Carl	5
Starbucks	Wendy	3
Starbucks	Kate	4
Starbucks	Vivian	5

Table Join

Key	Value
Business	Avg_Star
Starbucks	4.5
Five Guys	3.8
Qdoba	3.5

Why Data Skew?

-- 20% business provide 80% reviews

Key	Value	
Business	User	Star
Starbucks	Tom	5
Five Guys	Tom	4
Starbucks	Jim	3
Qdoba	John	2
Five Guys	Alice	5
Starbucks	Romia	2
Starbucks	Juliet	5
Qdoba	Sam	3
Starbucks	Carl	5
Starbucks	Wendy	3
Starbucks	Kate	4
Starbucks	Vivian	5

Table Join

Key	Value
Business	Avg_Star
Starbucks	4.5
Five Guys	3.8
Qdoba	3.5

Send the data to different partitions by hash code of the key

Executor I

Key	Value A
Starbucks	4.5

Key	Value B	
Starbucks	Tom	5
Starbucks	Jim	3
Starbucks	Romia	2
Starbucks	Juliet	5
Starbucks	Carl	5
Starbucks	Wendy	3
Starbucks	Kate	4
Starbucks	Vivian	5

Executor II

Key	Value A
Five Guys	3.8

Key	Value B	
Five Guys	Tom	4
Five Guys	Alice	5

Executor III

Key	Value A
Qdoba	3.5

Key	Value B	
Qdoba	John	2
Qdoba	Sam	3

Executor I

Key	Value A
Starbucks	4.5

Key	Value B	
Starbucks	Tom	5
Starbucks	Jim	3
Starbucks	Romia	2
Starbucks	Juliet	5
Starbucks	Carl	5
Starbucks	Wendy	3
Starbucks	Kate	4
Starbucks	Vivian	5

Executor II

Key	Value A
Five Guys	3.8

Key	Value B	
Five Guys	Tom	4
Five Guys	Alice	5

Executor III

Key	Value A
Qdoba	3.5

Key	Value B	
Qdoba	John	2
Qdoba	Sam	3

Execute “join” on different partitions parallelly

Executor I

Key	Value A
Starbucks	4.5

Key	Value B	
Starbucks	Tom	5
Starbuck		3
Starbuck		2
Starbuck		5
Starbuck		5
Starbuck		3
Starbucks	Kate	4
Starbucks	Vivian	5

Running!

!

Executor II

Key	Value B		Value A
Five Guys	Tom	4	3.8
Five Guys	Alice	5	3.8

Finished!

Executor III

Key	Value B		Value A
Qdoba	John	2	3.5
Qdoba	Sam	3	3.5

Finished!

Executor I

Key	Value B		Value A
Starbucks	Tom	5	4.5
Starbucks	Jim	3	4.5
Starbucks	Romia	2	4.5
Starbucks	Juliet	5	4.5
Starbucks	Carl	5	4.5
Starbucks	Wendy	3	4.5
Starbucks	Kate	4	4.5
Starbucks	Vivian	5	4.5

Finished!

Executor II

Key	Value B		Value A
Five Guys	Tom	4	3.8
Five Guys	Alice	5	3.8

Finished!

Executor III

Key	Value B		Value A
Qdoba	John	2	3.5
Qdoba	Sam	3	3.5

Finished!

Data Skew Solution – Broadcast Small Table

Key	Value	
Business	User	Star
Starbucks	Tom	5
Five Guys	Tom	4
Starbucks	Jim	3
Qdoba	John	2
Five Guys	Alice	5
Starbucks	Romia	2
Starbucks	Juliet	5
Qdoba	Sam	3
Starbucks	Carl	5
Starbucks	Wendy	3
Starbucks	Kate	4
Starbucks	Vivian	5

Table Join

Key	Value
Business	Avg_Star
Starbucks	4.5
Five Guys	3.8
Qdoba	3.5

Broadcast to each executor

Data Skew Solution – Broadcast Small Table

Key	Value	
Business	User	Star
Starbucks	Tom	5
Five Guys	Tom	4
Starbucks	Jim	3
Qdoba	John	2
Five Guys	Alice	5
Starbucks	Romia	2
Starbucks	Juliet	5
Qdoba	Sam	3
Starbucks	Carl	5
Starbucks	Wendy	3
Starbucks	Kate	4
Starbucks	Vivian	5

Table Join

Key	Value
Business	Avg_Star
Starbucks	4.5
Five Guys	3.8
Qdoba	3.5

Divide in 3 parts evenly

Broadcast to each partition

Data Skew Solution – Broadcast Small Table

Executor I

Key	Value
Business	Avg_Star
Starbucks	4.5
Five Guys	3.8
Qdoba	3.5

Key	Value	
Business	User	Star
Starbucks	Tom	5
Five Guys	Tom	4
Starbucks	Jim	3
Qdoba	John	2

Executor II

Key	Value
Business	Avg_Star
Starbucks	4.5
Five Guys	3.8
Qdoba	3.5

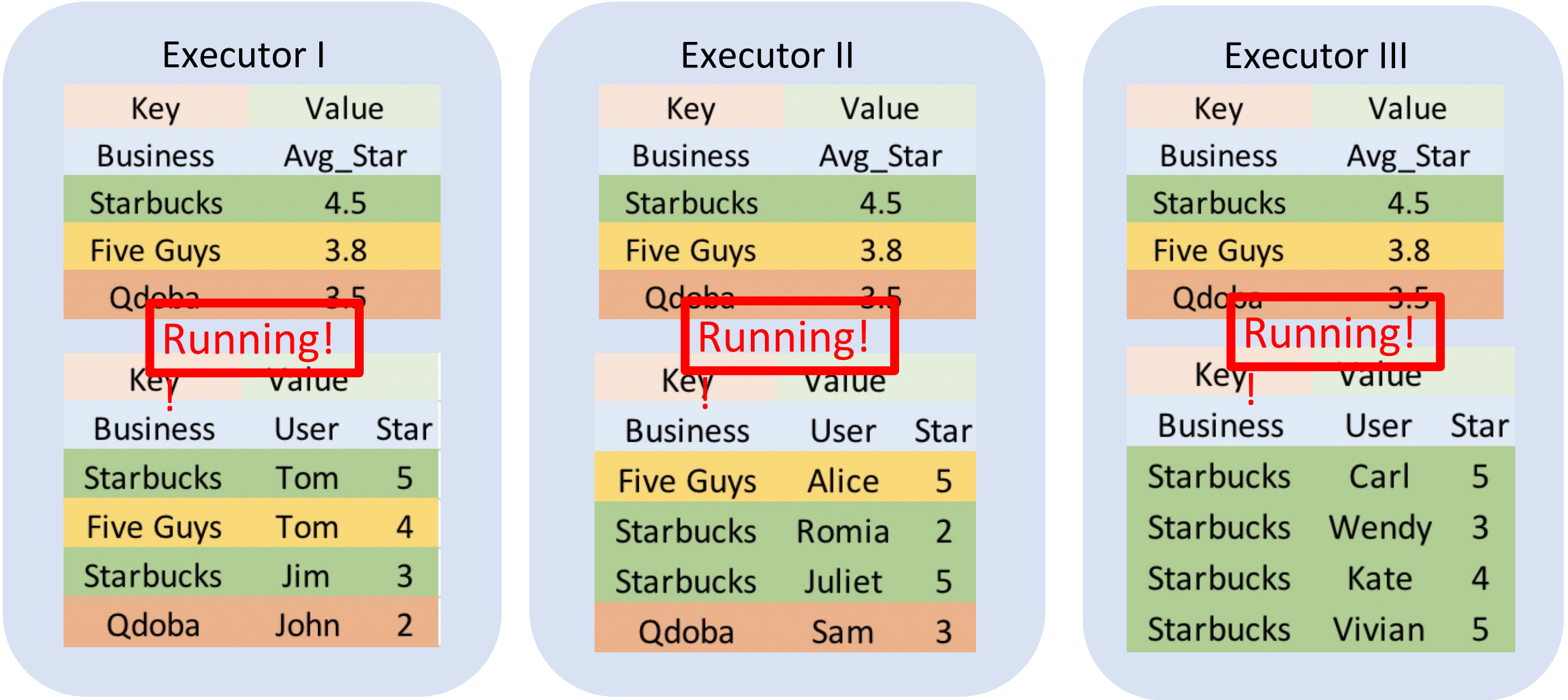
Key	Value	
Business	User	Star
Five Guys	Alice	5
Starbucks	Romia	2
Starbucks	Juliet	5
Qdoba	Sam	3

Executor III

Key	Value
Business	Avg_Star
Starbucks	4.5
Five Guys	3.8
Qdoba	3.5

Key	Value	
Business	User	Star
Starbucks	Carl	5
Starbucks	Wendy	3
Starbucks	Kate	4
Starbucks	Vivian	5

Data Skew Solution – Broadcast Small Table



Data Skew Solution – Broadcast Small Table

Executor I

Key	Value B		Value A
Business	User	Star	Avg_Star
Starbucks	Tom	5	4.5
Five Guys	Tom	4	3.8
Starbucks	Jim	3	4.5
Qdoba	John	2	3.5

Finished!

Executor II

Key	Value B		Value A
Business	User	Star	Avg_Star
Five Guys	Alice	5	3.8
Starbucks	Romia	2	4.5
Starbucks	Juliet	5	4.5
Qdoba	Sam	3	3.5

Finished!

Executor III

Key	Value		Value A
Business	User	Star	Avg_Star
Starbucks	Carl	5	4.5
Starbucks	Wendy	3	4.5
Starbucks	Kate	4	4.5
Starbucks	Vivian	5	4.5

Finished!

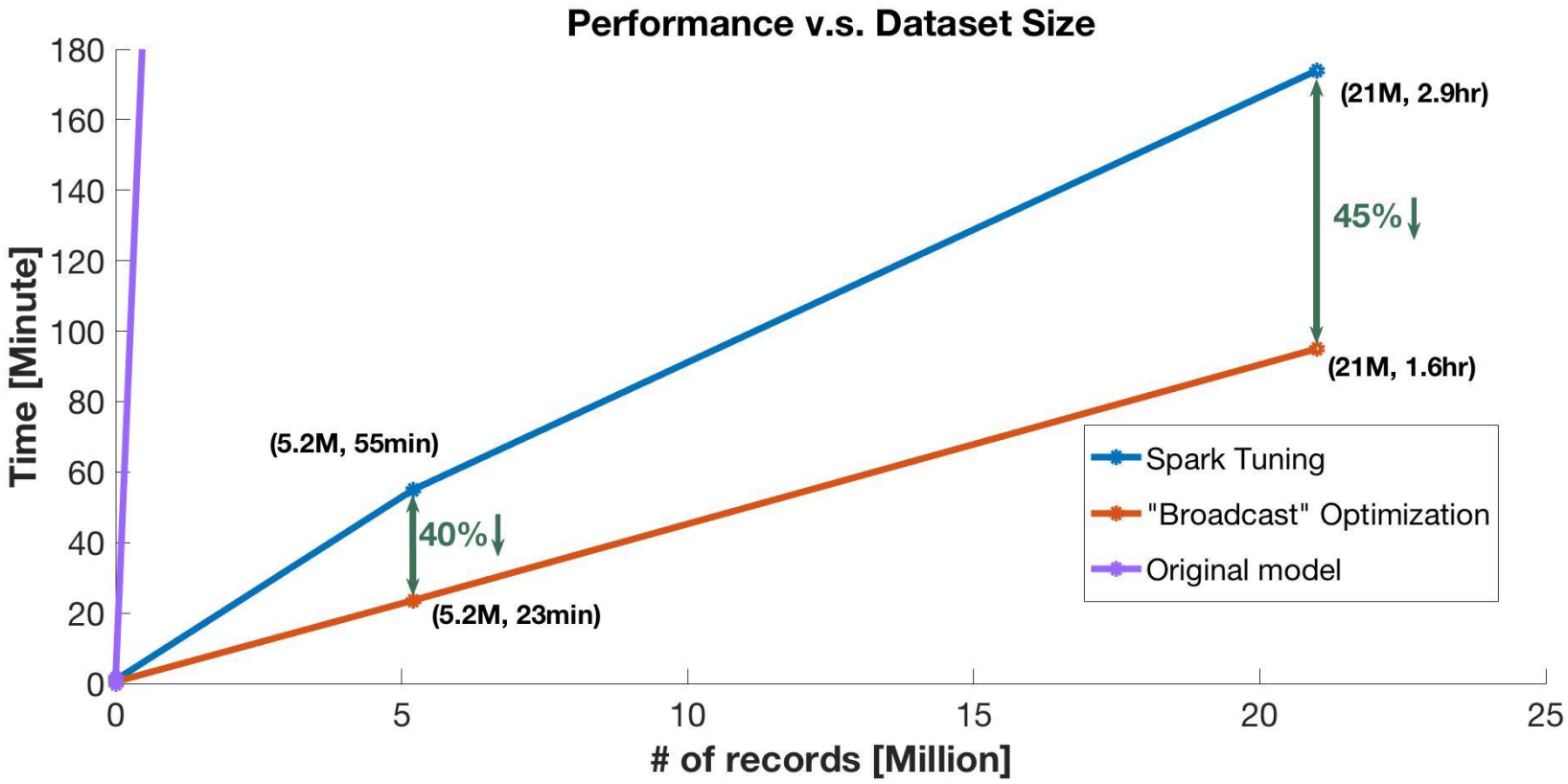
Data Skew Solution

```
reviews = sc.textFile(args.reviews)\
            .map(lambda x: tuple(x.split(',')))\
            .map(lambda (u, i, riu):(int(u), (int(i), float(riu)))).partitionBy(args.N).cache()

rating_pairs = reviews.join(reviews).filter(lambda (x,y) : y[0][0] < y[1][0])\
                      .map(lambda (x,y) : (y[0][0], (y[1][0], y[0][1], y[1][1])))\
                      .join(avg_star).map(lambda (i,((j, ri, rj), avg_i)) : (j, (i, avg_i, ri, rj)))\
                      .join(avg_star).map(lambda (j, ((i, avg_i, ri, rj), avg_j)) : ((i,j),(ri, rj, avg_i, avg_j)))\
                      .mapvalues(lambda (ri, rj, avg_i, avg_j) : (ri - avg_i, rj - avg_j))\
                      .reduceByKey(lambda x,y : x+y)\
                      .mapValues(list).mapValues(similarity_cal).filter(lambda (x,y) : y < 0.9999 and y > -0.9999)
```

```
broadcast_avg_star = sc.broadcast(avg_star.collect())
avg_star.unpersist()
reviews = sc.textFile(args.reviews)\
            .map(lambda x: tuple(x.split(',')))\
            .map(lambda (u, i, riu):(int(u), (int(i), float(riu)))).partitionBy(args.N).cache()

rating_pairs = reviews.join(reviews).filter(lambda (x,y) : y[0][0] < y[1][0])\
                      .map(lambda (x,y) : ((y[0][0],y[1][0]), (y[0][1], y[1][1])))\
                      .map(lambda (x,y) : (x,(y[0],y[1],broadcast_avg_star.value[x[0]][1],broadcast_avg_star.value[x[1]][1])))\
                      .mapValues(lambda (ri, rj, avg_i, avg_j) : (ri - avg_i, rj-avg_j))\
                      .partitionBy(args.N)\
                      .reduceByKey(lambda x,y : x+y)\
                      .mapValues(list).mapValues(similarity_cal)\
                      .filter(lambda (x,y) : y<0.9999 and y > -0.9999)\
                      .saveAsTextFile(args.output)
```

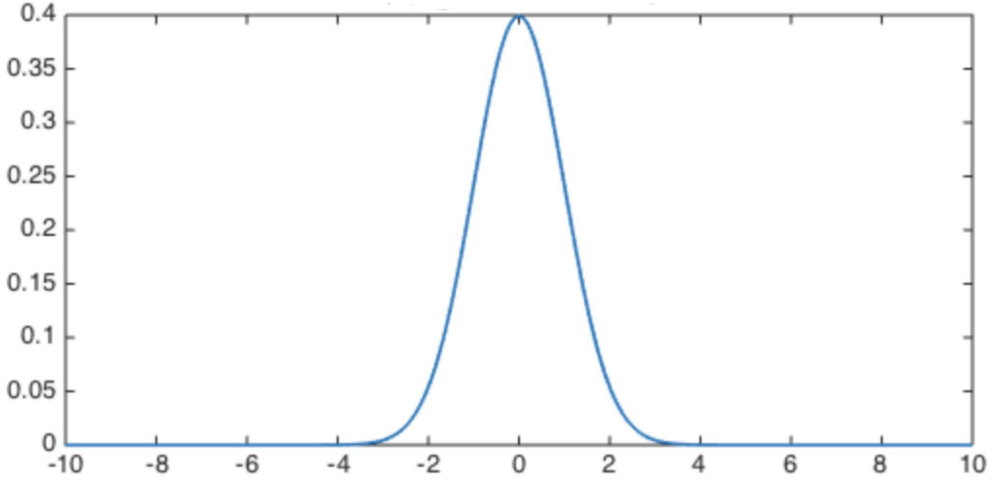
Performance

- System: 1 master node, 3 worker nodes.
- Memory: 6GB/node.



Application

- Transaction data processing
  - 20% companies provide 80% transaction data
- Traffic data processing
  - 20% geographic locations provide 80% traffic data
- Any data follows normal distribution with small sigma value.



Ruby Liu

Master in Computer Engineering,

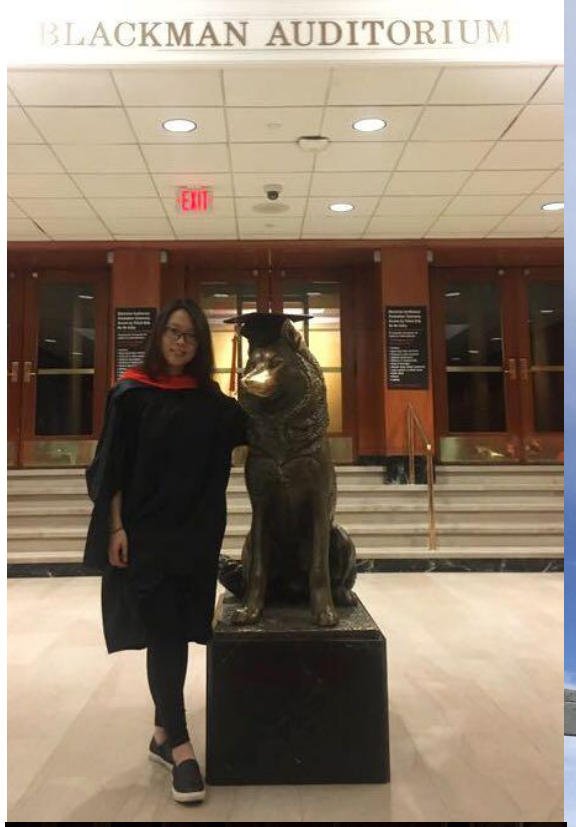
@ Northeastern University

Bachelor in Software Engineering,

Double degree in Mathematics,

@ Nankai University

- Ballet
- Swimming
- Travelling
- Debug 👻



Q&A

Thank you!

Further work

- Joined two big tables in Spark
  - Add a column “index” to the balanced table
  - Iteratively broadcast part of the balanced table according to the index value.