cloudera

# BigPetStore on Spark

## Implementing use cases on a unified data engine

Marton Balassi | Solutions Architect
@MartonBalassi | mbalassi@cloudera.com

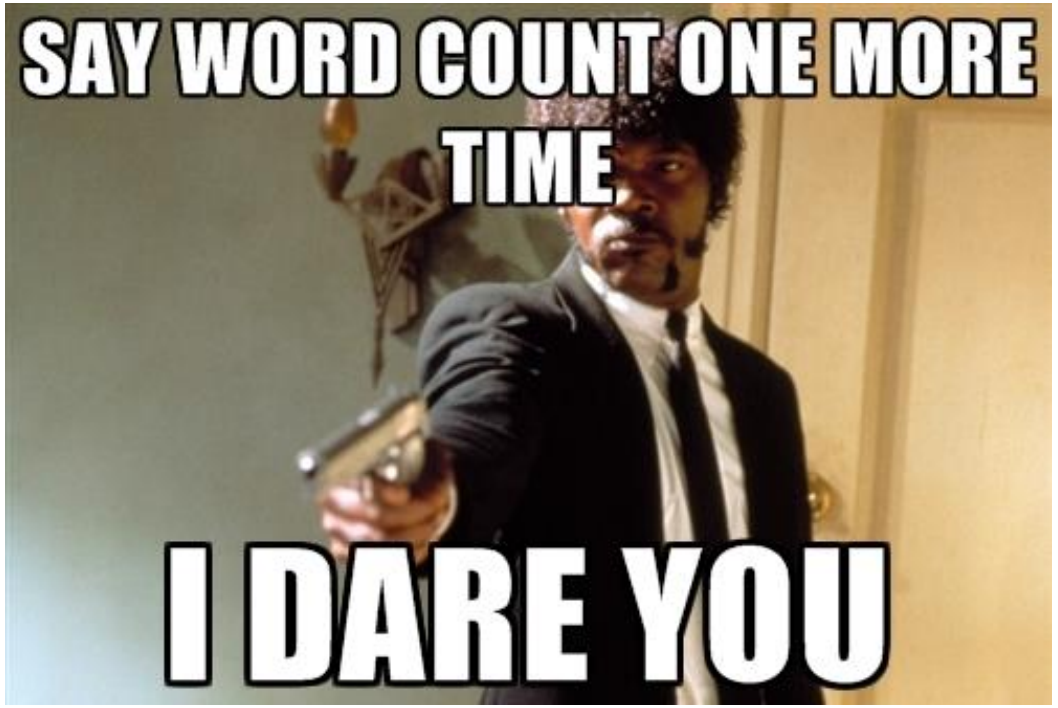
# Outline

- BigPetStore model
- Data generator with the RDD API
- ETL with RDDs & Spark SQL
- Matrix factorization with MLlib
- Recommendation with the Spark Streaming API
- Summary

cloudera

# BigPetStore

- Blueprints for Big Data applications
- Consists of:
- Data Generators
- Examples using tools in Big Data ecosystem to process data
- Build system and tests for integrating tools and multiple JVM languages
  - Part of the Apache BigTop project




cloudera

# BigPetStore model

- Customers visiting pet stores generating transactions, location based

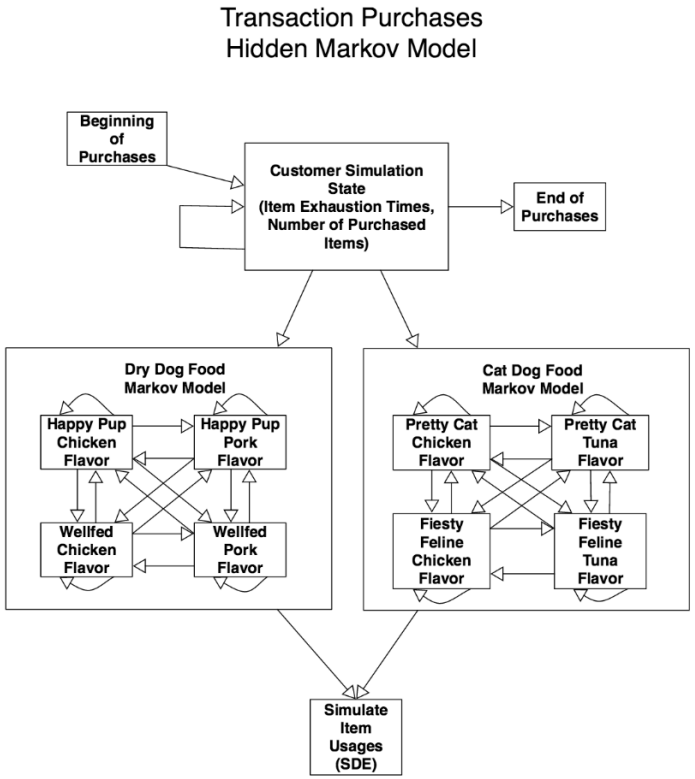
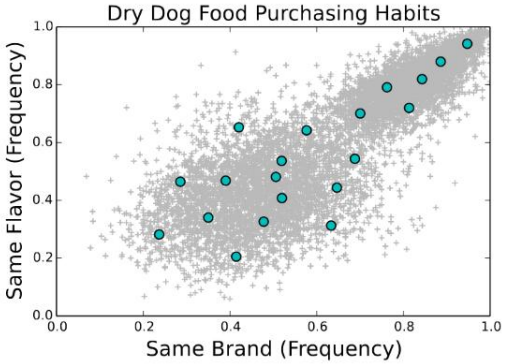




Fig. 4. Distributions of Distances of Customers to Closest Stores




Nowling, R.J.; Vyas, J., "A Domain-Driven, Generative Data Model for Big Pet Store," in *Big Data and Cloud Computing (BDCloud), 2014 IEEE Fourth International Conference on* , vol., no., pp.49-55, 3-5 Dec. 2014

cloudera

# Data generation

- Use RJ Nowling's Java generator classes
- Write transactions to JSON

```
val sc = new SparkContext(conf)
val (stores, productCategories, customers) = getData()
val startTime = getCurrentMillis()
val (storesBC, productsBC) = (sc.broadcast(stores), sc.broadcast(productCategories))


val transactions = sc.parallelize(customers)
        .mapPartitionsWithIndex(generateTransactions)
        .map{t => t.setDateTime(t.getDateTime + startTime); t}

transactions.saveAsTextFile(output)
```

cloudera

# ETL with the RDD API

- Read the dirty JSON
- Output (customer, product) pairs for the recommender

```
val sc = new SparkContext(conf)
val transactions = sc.textFile(json).map(new Transaction(_))

val productsWithIndex = transactions.flatMap(_.getProducts)
        .distinct
        .zipWithUniqueId

val customerAndProductPairs = transactions
        .flatMap(t => t.getProducts.map(p => (t.getCustomer.getId, p)))
        .map(pair => (pair._2, pair._1.toLong))
        .join(productsWithIndex).map(res => (res._2._1, res._2._2))
        .distinct

customerAndProductPairs.map(mkstring).saveAsTextFile(output)
```

cloudera

# ETL with SparkSQL

- Read the dirty JSON
- SQL queries: select the stores with the most transacions

```
val (sc, sqlContext) = (new SparkContext(conf), new SQLContext(sc))
val transactions = sqlContext.read.json(input)
transactions.registerTempTable(”transactions”)

val countByStore = sqlContext.sql(”SELECT store.id id, store.name name, COUNT(store.id) count ” +
         ”FROM transactions GROUP BY store.id, store.name”)
countByStore.registerTempTable(”countByStore”)

val bestStores = sqlContext.sql(”SELECT ct.id, ct.name, max.count ” +
        ”FROM countByStore, (SELECT MAX(count) as count FROM countByStore) max ” +
         ”WHERE ct.count = max.count”
```

cloudera

# ETL with SparkSQL (2)

- Read the dirty JSON
- SQL queries: list transaction count by month

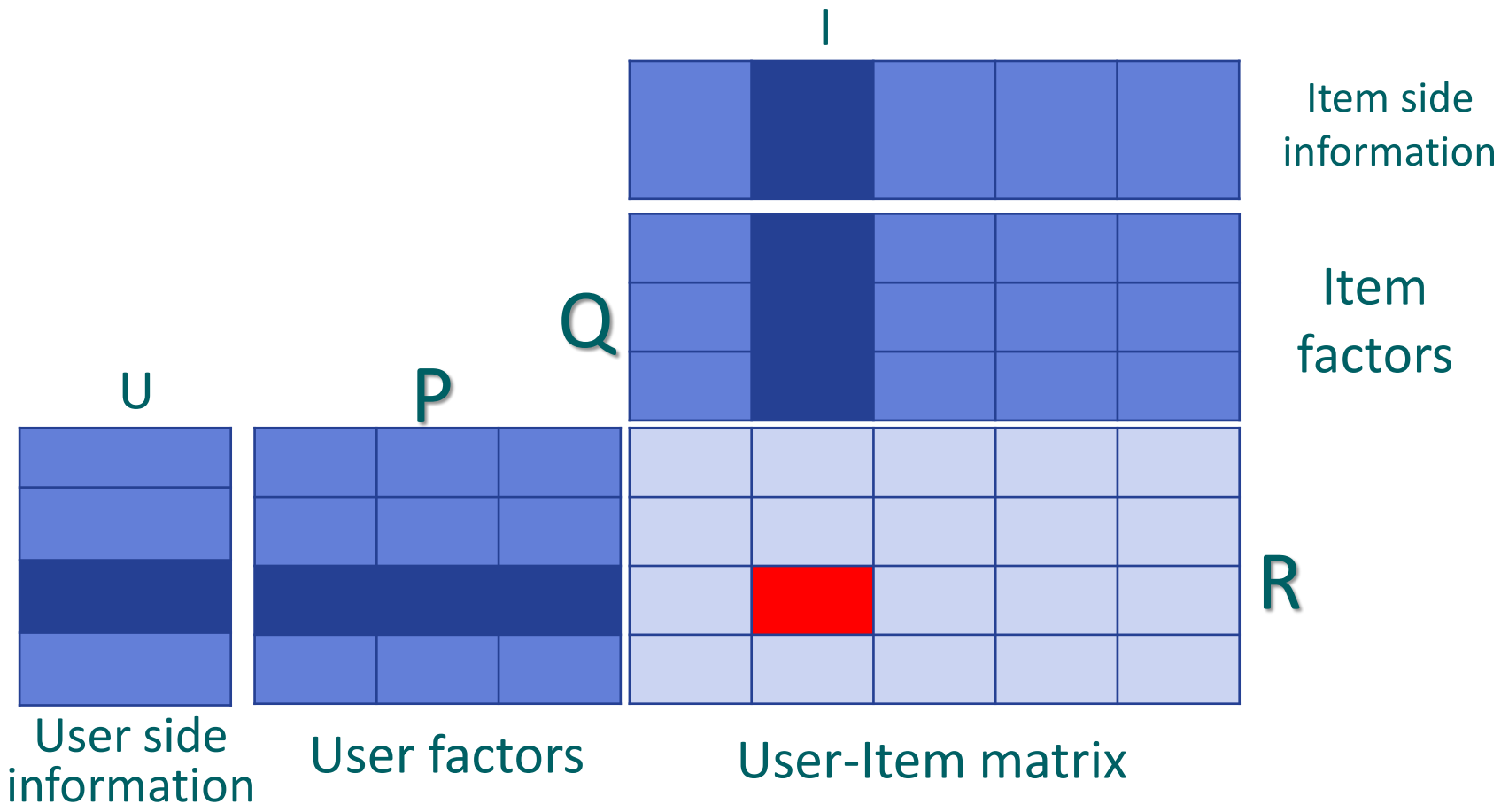
```
def month = (dateTime : Double) => {
        val millis = (dateTime * 24 * 3600 * 1000).toLong
        new Date(millis).getMonth
}

sqlContext.udf.register(”month”, month)

val byMonthCont = sqlContext.sql(”SELECT month(dateTime), COUNT(month) ” +
        ”FROM transactions ”
         ”GROUP BY month(dateTime)”)
```

cloudera

# A little recommender theory




- R is potentially huge, approximate it with P∗Q
- Prediction is TopK(user's row ∗ Q)

# Matrix factorization with MLlib

- Read the (customer, product) pairs
- Write P and Q to file

```
val sc = new SparkContext(conf)
val transactions = sc.textFile(input)
        .map(s => Rating(s))

val model = ALS.trainImplicit(input, numFactors, iterations, lambda, alpha)

model.save(sc, out)
```

cloudera

# Recommendation with DStreams

- Get the ratings for the row of the user
- (Could be optimized)

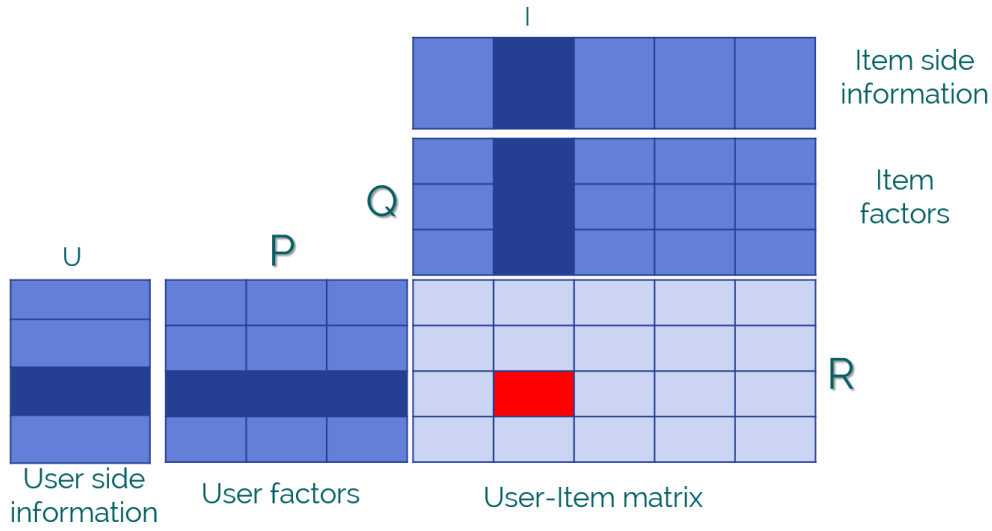
```
val sc = new SparkContext(conf)
val ssc = new StreamingContext(sc, Seconds(1))

val model = MatrixFactorizationModel.load(sc, input)

val query = ssc.socketTextStream(”localhost”, 9999)
        .map(user => getItems(user))

val result = query.foreachRDD{ userVector =>
        val ratings = model.predict(userVector)
        print(topk(ratings))
}

ssc.start(); ssc.awaitTermination()
```




cloudera

# Summary

- Go beyond WordCount with BigPetStore
- Feel free to mix the RDD, DataFrames, SQL, MLlib and streaming APIs (and all the others) in your Spark workflows
- Data generation, cleaning, ETL, Machine learning, streaming prediction on top of one engine with under 500 lines of code
- A Spark pet project is always fun. No pun intended.

# Big thanks to

- The BigPetStore folks:

  Suneel Marthi

  Ronald J. Nowling

  Jay Vyas

- Clouderans supporting the pet project:

  Sean Owen

  Alexander Bartfeld

  Alexander Bibighaus

cloudera

cloudera

# Thank you

@MartonBalassi
mbalassi@cloudera.com