클라우드 기반 빅데이터 환경구축 (Part 2)

실습조교: 이계원, 엄태건

서울대학교 컴퓨터공학부

Overview

0. 실습에서 사용할 소프트웨어
1. 빅데이터 분석을 위한 클라우드 환경 셋업
2. 클라우드상 배치 처리 분석
3. 클라우드상 대화형 질의 분석
4. **클라우드상 스트림 처리 분석**
5. **클라우드상 기계학습 분석**
6. 클라우드상 딥러닝 분석

4. 클라우드상
스트림 처리 분석

Spark Streaming Basics (Review)

Stream Processing

- Processes unbounded incoming stream data in real-time
- Example: Real-time network log aggregation, Twitter trend analysis, Anomaly detection, …
- vs Batch processing
  - Batch processing processes **bounded** set of data and produces results
  - Stream processing processes **unbounded** incoming stream data events **continuously**
- Two types of execution models
  - Continuous operator model
  - Micro-batch model

Spark Streaming Basic

- Spark Streaming Implement micro-batch model stream processing on top of Apache Spark
- Run a streaming computation as a series of small, deterministic batch jobs (**micro-batches**)

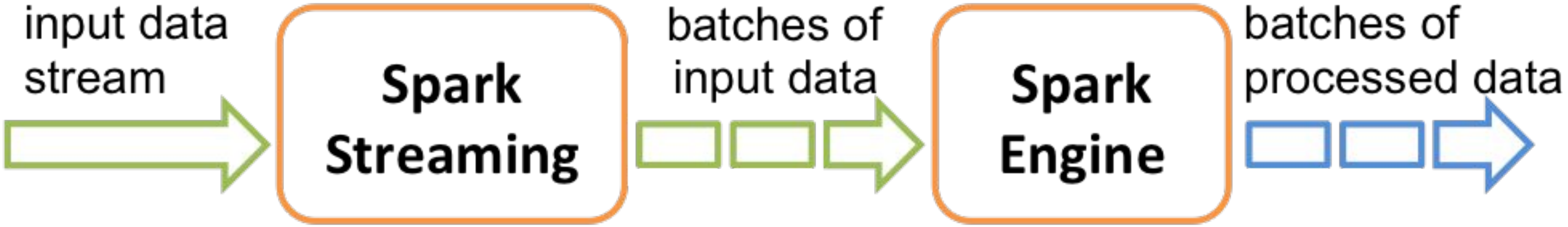
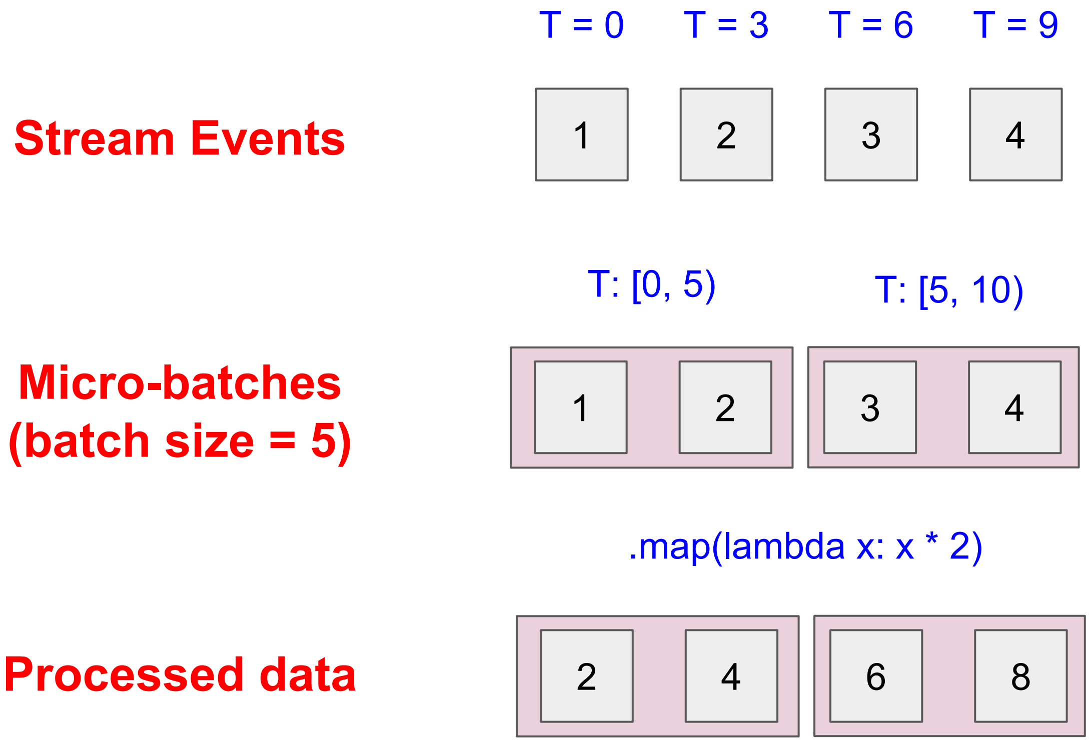


Image: https://spark.apache.org/docs/latest/streaming-programming-guide.html

Spark Streaming Query Processing Example



T = 0 T = 3 T = 6 T = 9

Stream Events

1 2 3 4

T: [0, 5) T: [5, 10)

Micro-batches (batch size = 5)

1 2 3 4

.map(lambda x: x * 2)

Processed data

2 4 6 8

Spark Streaming Basic (Contd.)

- vs Continuous operator model (Storm, Heron)
  - Advantage
    - Fast fault recovery via parallel recovery
    - Straggler handling
  - Disadvantage
    - High latency (second-scale) caused by micro-batch processing & scheduling overhead

Stream Processing via Spark Streaming

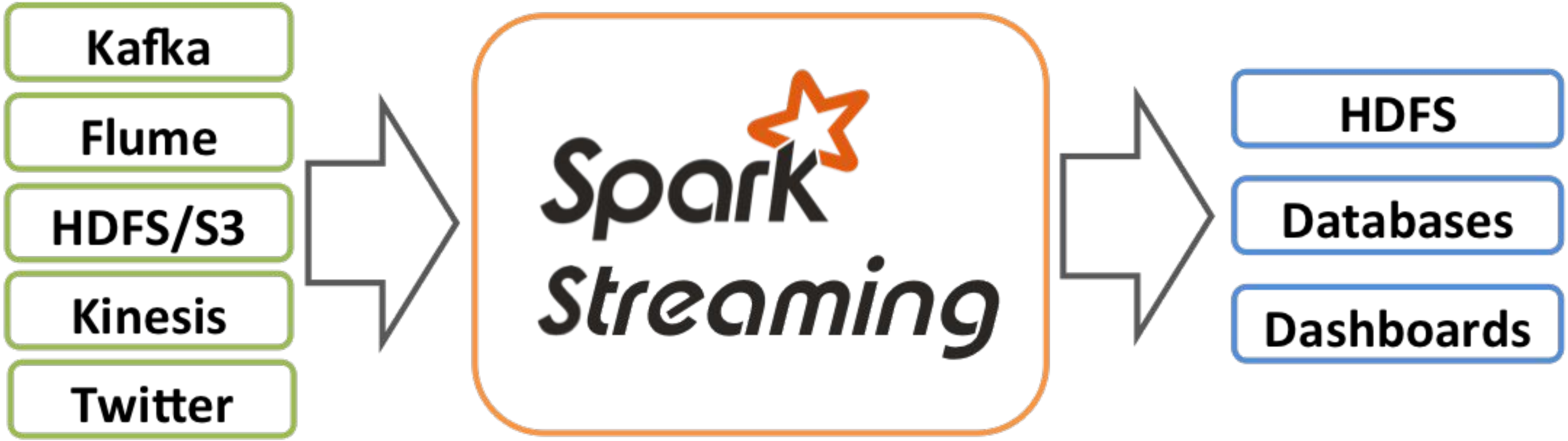


Image: https://spark.apache.org/docs/latest/streaming-programming-guide.html

Spark Streaming Practice

Prerequisites

- Java 8 or higher version
- AWS YARN cluster with HDFS setup
- Spark cluster working on YARN

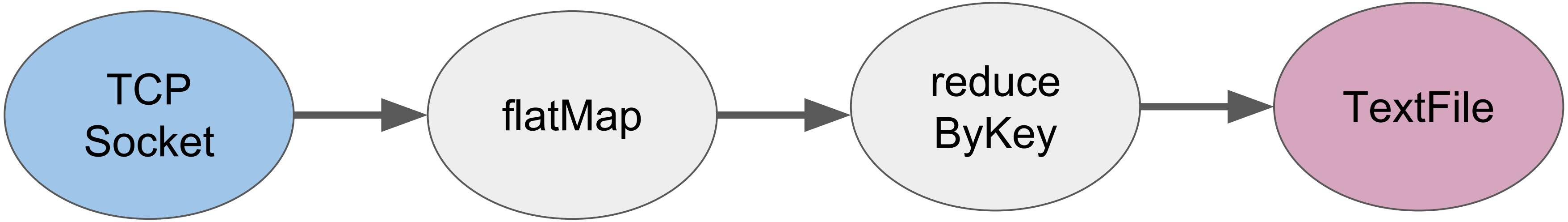
What we will do

- Part 1: Implementing & running simple wordcount example query
- Part 2: Setting up Kafka source for large-scale message transportation
- Part 3: Implementing & running windowed movie aggregation

Detailed information can be found on Jupyter notebook

Part 1: Running a simple wordcount query

- Continuously get the text stream data via TCP socket
- Split the sentence into multiple words
- Update the word count data for each word
- Emit the output to a text file
- Test the running query with netcat



TCP Socket

flatMap

reduce ByKey

TextFile

Part 1 Detail (1) - Set up streaming context

```
spark_context = SparkContext(...)

streaming_context = StreamingContext(sc, 1)
```

Batch interval (1 second)

Part 1 Detail (2) - Write stream query

```
# Fetch the line from a tcp socket
lines = streaming_context.socketTextStream(...)
# Processing something
…
# Output the result to the save
results.saveAsTextFiles(...)
```

Part 2: Setting Up Kafka source

- Download the movie data json set (https://raw.githubusercontent.com/prust/wikipedia-movie-data/master/movies.json)
- Install & configure Kafka messaging server
- Make a script which publishes real-time movie watching data stream to the Kafka messaging server

Apache Kafka

- Kafka is a distributed messaging & streaming platform
  - We will use messaging features of Kafka in this tutorial
- Kafka supports large-scale & reliable publishing / subscribing data stream events per topic
  - Similar to message brokers (RabbitMQ, …)
- Input data is partitioned, distributed, and replicated to multiple machines to guarantee fault tolerance
- For more information, consult to
  - https://kafka.apache.org/documentation/#introduction

Part 3: Windowed Movie Popularity Aggregation

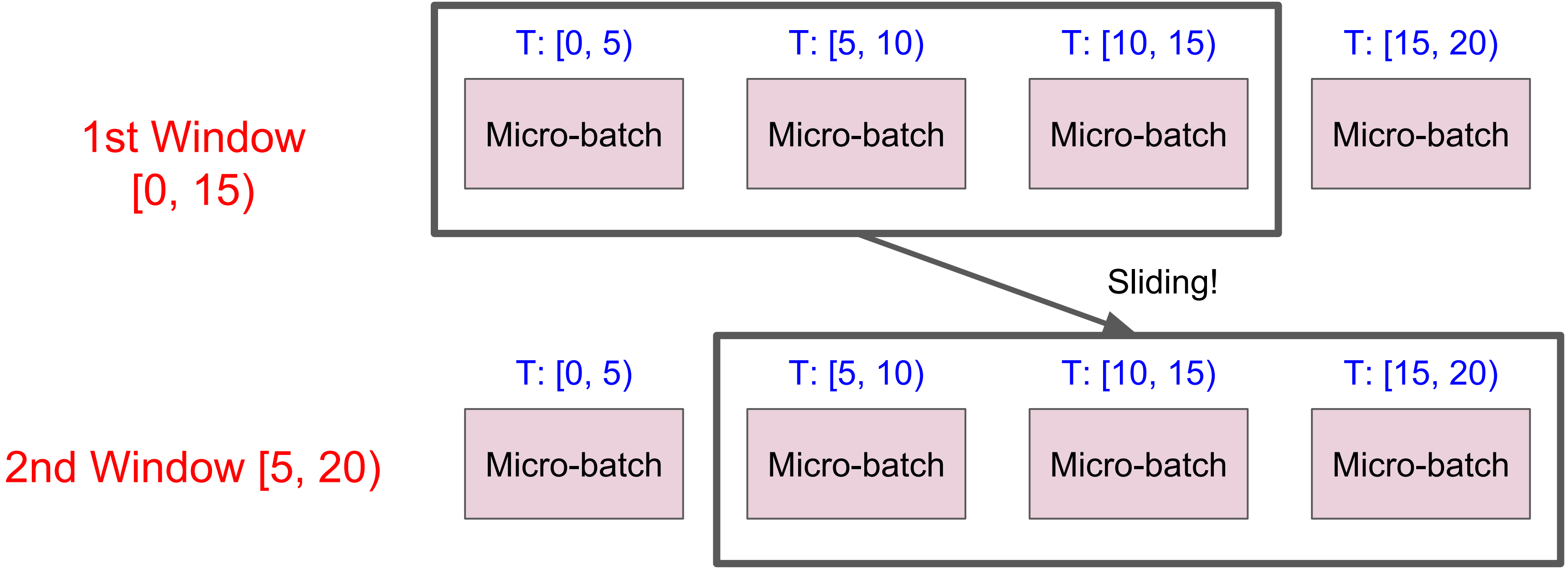
- Get the json-typed movie data stream from Kafka
- Parse the json data into key-value data stream
- Implement windowed aggregation
- Write the result to HDFS file

Recap: Window Operation

- Spark Streaming allows processing data over a recent set of data, called **Window**
- The types of windows are
  - Time-based window
  - Count-based window
  - Session window
- We will implement time-based window
  - Window Length & Slide Interval

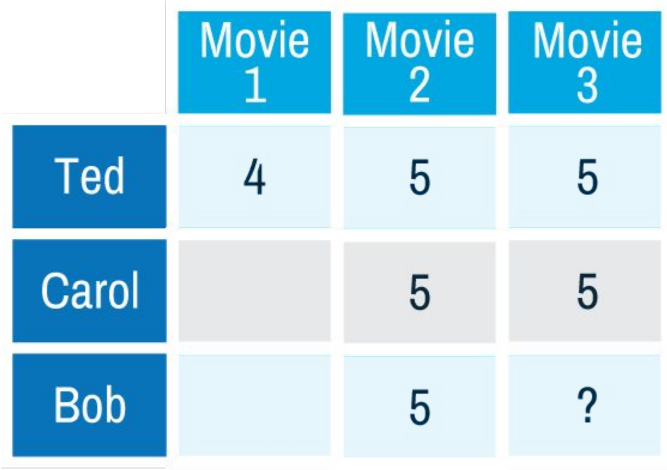
Sliding Window Operation in Spark Streaming

Window Size = 20, Slide Interval = 5



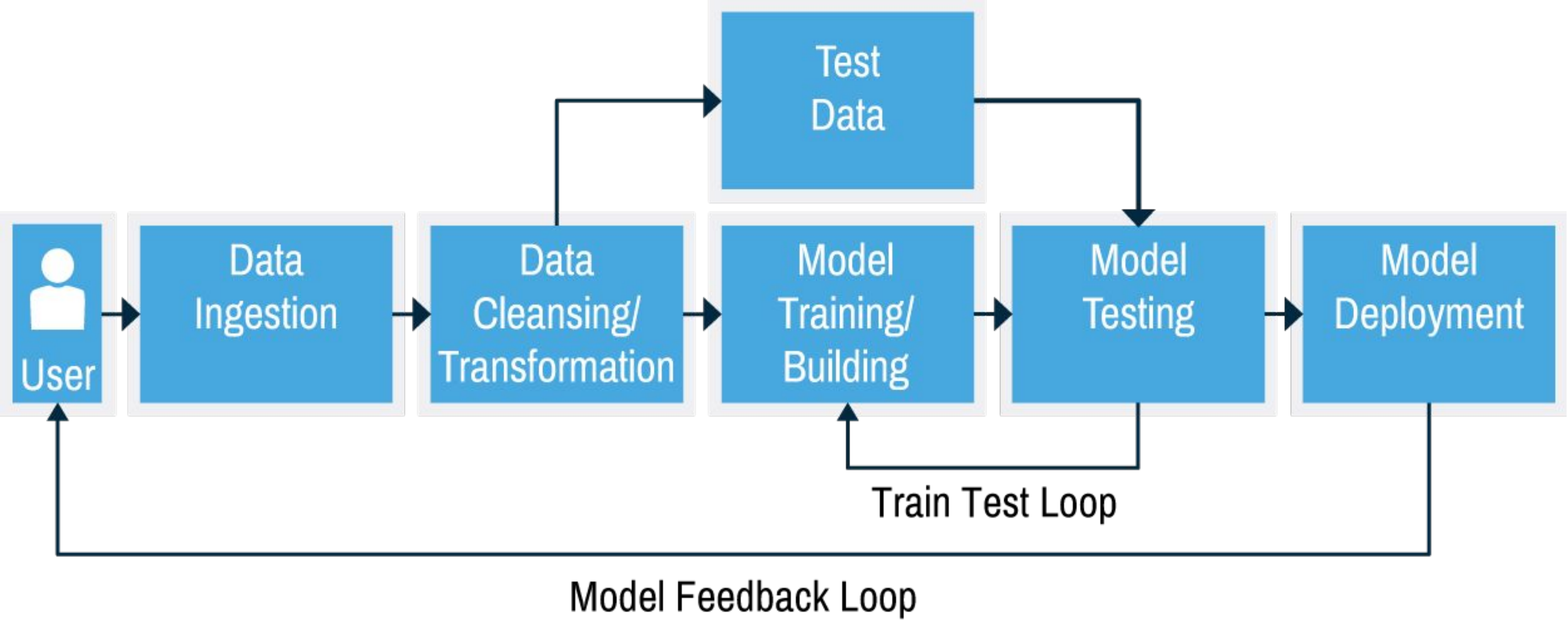
5. 클라우드상 기계학습 분석

Spark MLlib을 활용한 영화 추천 모델 만들기



	Movie 1	Movie 2	Movie 3
Ted	4	5	5
Carol		5	5
Bob		5	?

Image: https://mapr.com/ebooks/spark/08-recommendation-engine-spark.html



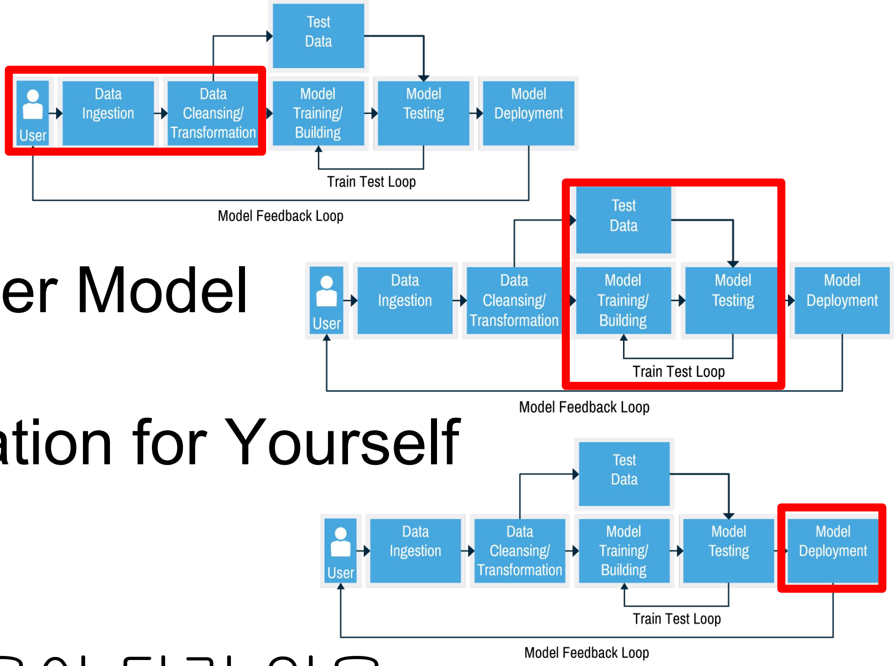
Typical Machine Learning Workflow

Image: https://mapr.com/ebooks/spark/08-recommendation-engine-spark.html

Movie Recommendation with Spark MLlib (ALS)

Contents Overview

- Part 1: Preparing your data
- Part 2: Build a Recommender Model
- Part 3: Movie Recommendation for Yourself



Jupyter notebook에 자세한 내용이 담겨 있음

Part 1: Preparing your data

We will use the following dataset:

1. ../data/ratings.data.gz
   a. <MovieName>::<Rating>::UserId

```
One Flew Over the Cuckoo's Nest (1975)::5::1
James and the Giant Peach (1996)::3::1
My Fair Lady (1964)::3::1
```

2. ../data/movies.dat
   a. <MovieId>::<MovieName>::Genre

```
1::Toy Story (1995)::Animation|Children's|Comedy
2::Jumanji (1995)::Adventure|Children's|Fantasy
3::Grumpier Old Men (1995)::Comedy|Romanc
```

Part 1: Preparing your data

```
One Flew Over the Cuckoo's Nest
(1975)::5::1
James and the Giant Peach (1996)::3::1
My Fair Lady (1964)::3::1
```

```
1::Toy Story
(1995)::Animation|Children's|Comedy
2::Jumanji
(1995)::Adventure|Children's|Fantasy
3::Grumpier Old Men
(1995)::Comedy|Romanc
```

DataFrame

UserId	MovieId	Rating
1	1	5
2	5	3
1	3	3

Part 2: Build a Movie Recommender Model

1. Create an ALS object and build a recommender model

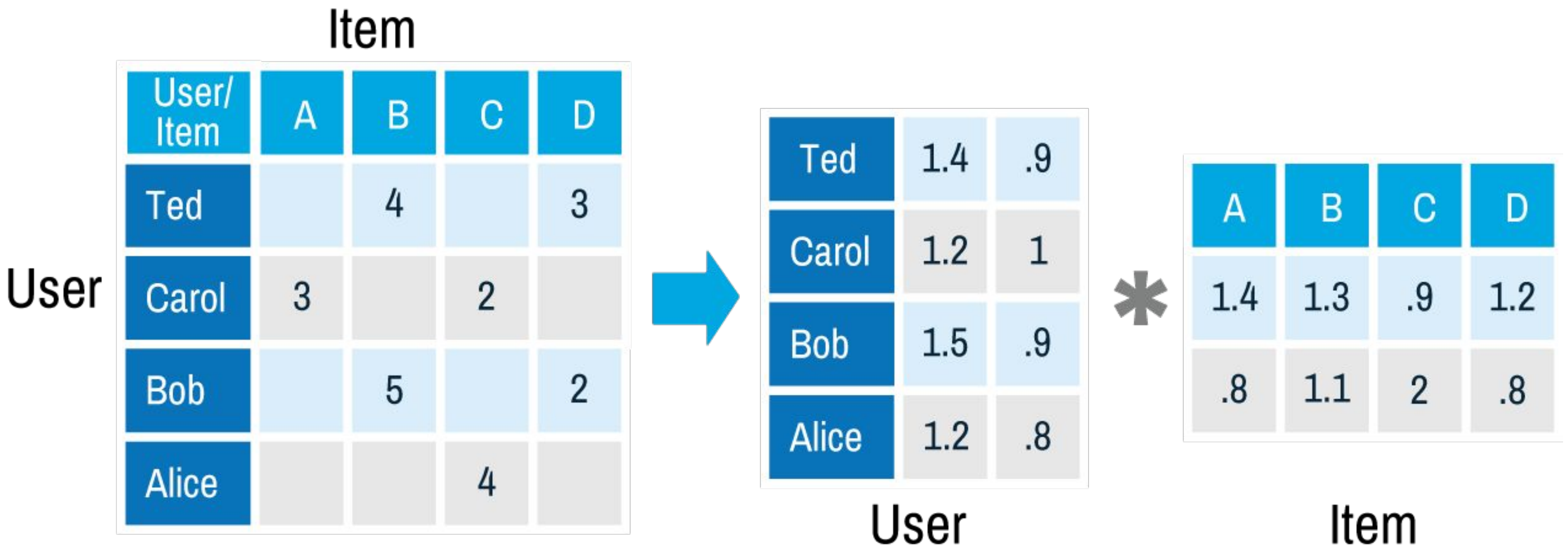
```
from pyspark.ml.recommendation import ALS

# Create an ALS object
als = ALS(rank=8, maxIter=5, numUserBlocks=10, numItemBlocks=10,
          implicitPrefs=False, userCol='userId',
          itemCol='movieId', ratingCol='rating',
          coldStartStrategy="drop")

# Build our first recommender model!
# Spark automatically trains the model with your dataset
firstModel = als.fit(ratings)
```

userId	movieId	rating
1	1	5
2	5	3
1	3	3

Part 2: Build a Movie Recommender Model



User/ Item	A	B	C	D
Ted		4		3
Carol	3		2	
Bob		5		2
Alice			4	

Ted	1.4	.9
Carol	1.2	1
Bob	1.5	.9
Alice	1.2	.8

A	B	C	D
1.4	1.3	.9	1.2
.8	1.1	2	.8

Image: https://mapr.com/ebooks/spark/08-recommendation-engine-spark.html

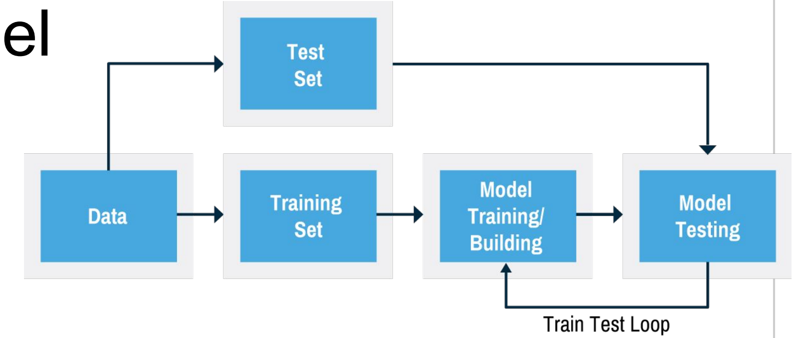
Part 2: Build a Movie Recommender Model

2. Improve your recommender model

a. Split your dataset!

  i. Training set

  ii. Test set



```
# Split the dataset into 80% and 20% testSet
trainingSet, testSet = (ratings.randomSplit([0.8, 0.2]))
```

Image: https://mapr.com/ebooks/spark/08-recommendation-engine-spark.html

Part 2: Build a Movie Recommender Model

2b. Estimating error using Root Mean Square Error (RMSE)

```
from pyspark.ml.evaluation import RegressionEvaluator

evaluator = RegressionEvaluator(metricName="rmse", labelCol="rating",
predictionCol="prediction")
rmse = evaluator.evaluate(testPrediction)
```

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N}(Predicted_i - Actual_i)^2}{N}}$$

2c. Let's try different parameters and choose the set of parameters which results in lowest RMSE value.

Part 3: Movie Recommendation for Yourself

3a. Make your dataset of movie ratings

3b. Train the recommender model with your dataset

3c. Recommend movies for you by using \`recommendForUserSubset\` method

```
# Pick 10 movie recommendations for you
myRatingPrediction = model.recommendForUserSubset(myRatingsSet, 10)

myRatingPrediction.show()
print(myRatingPrediction.select("recommendations").take(10))
```

Homework #1:
AWS, HDFS, Spark SQL,
Spark Streaming

Homework #1-1: AWS, HDFS, Spark SQL

- You'll do by yourself…
  - Launch AWS instances
  - Load data into HDFS
  - Run a Spark SQL query, and visualize the results

- You'll submit…
  - Screenshots of Spark Web UI
  - Screenshot of visualized output dataset

Homework #1-2: Stream Join

- You’ll do yourself…
  - Generate KAFKA datastream
  - Transform the input streams into keyed window
  - Perform join operation on data streams
  - Write the result to HDFS

- You’ll submit…
  - The stream query source code
  - The 1-min running result file written to HDFS