

25 YEARS ANNIVERSARY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY



HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

Chapter 7

Stream processing

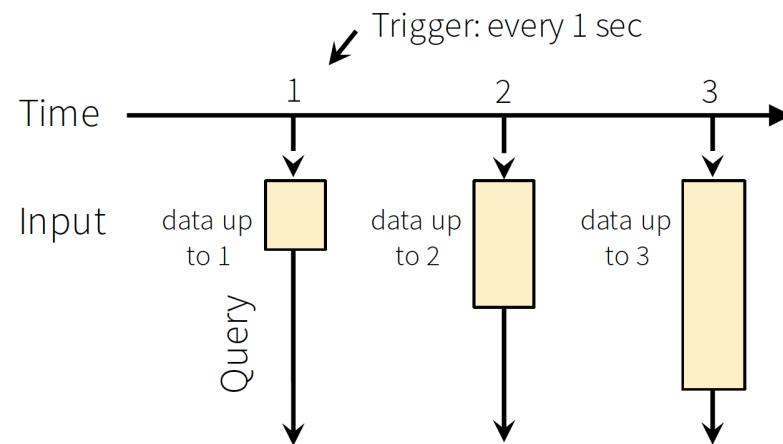
Structured streaming in Spark

Pain points with DStreams

- Processing with event-time, dealing with late data
 - DStream API exposes batch time, hard to incorporate event-time
- Interoperate streaming with batch AND interactive
 - RDD/DStream has similar API, but still requires translation
- Reasoning about end-to-end guarantees
 - Requires carefully constructing sinks that handle failures correctly
 - Data consistency in the storage while being updated

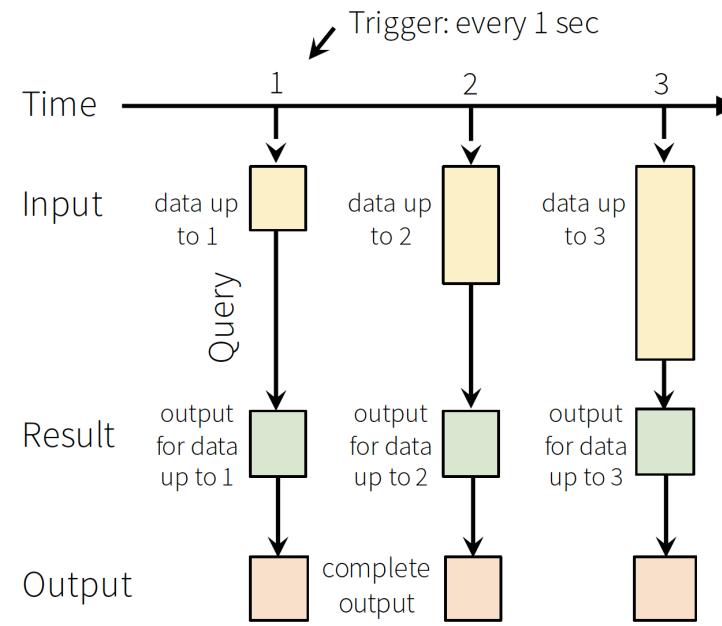
New model

- Input: data from source as an append-only table
- Trigger: how frequently to check input for new data
- Query: operations on input usual map/filter/reduce new window, session ops



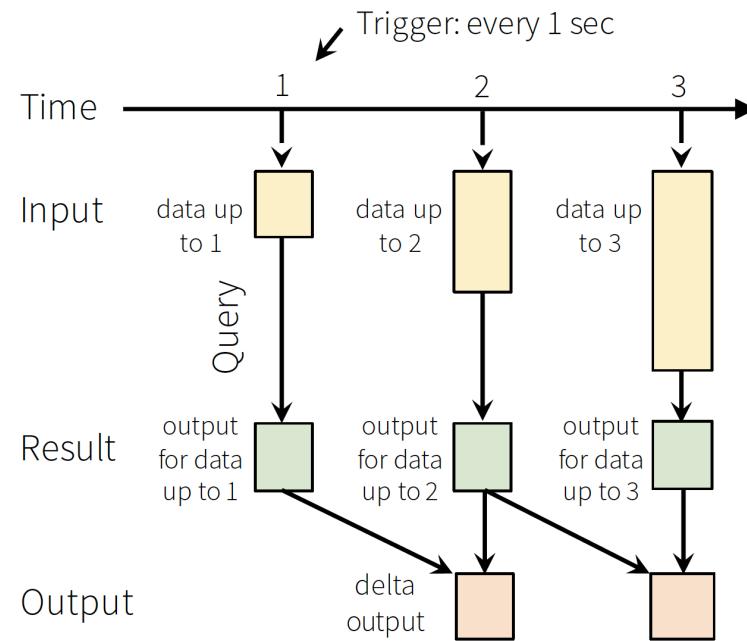
New model (2)

- Result: final operated table updated every trigger interval
- Output: what part of result to write to data sink after every trigger
- Complete output: Write full result table every time



New model (3)

- Delta output: Write only the rows that changed in result from previous batch
- Append output: Write only new rows
- *Not all output modes are feasible with all queries



Batch ETL with DataFrames

```
input = spark.read
```

```
    .format("json")
```

```
    .load("source-path")
```

- Read from Json file

```
result = input
```

```
    .select("device",  
           "signal")
```

```
    .where("signal > 15")
```

- Select some devices

- Write to parquet file

```
result.write
```

```
    .format("parquet")
```

```
    .save("dest-path")
```

Streaming ETL with DataFrames

```
input = spark.read  
    .format("json")  
    .stream("source-  
path")  
  
result = input  
    .select("device",  
    "signal")  
    .where("signal >  
15")  
  
result.write  
    .format("parquet")  
    .startStream("dest-  
path")
```

- Read from Json file stream
 - Replace load() with stream()
- Select some devices
 - Code does not change
- Write to Parquet file stream
 - Replace save() with startStream()

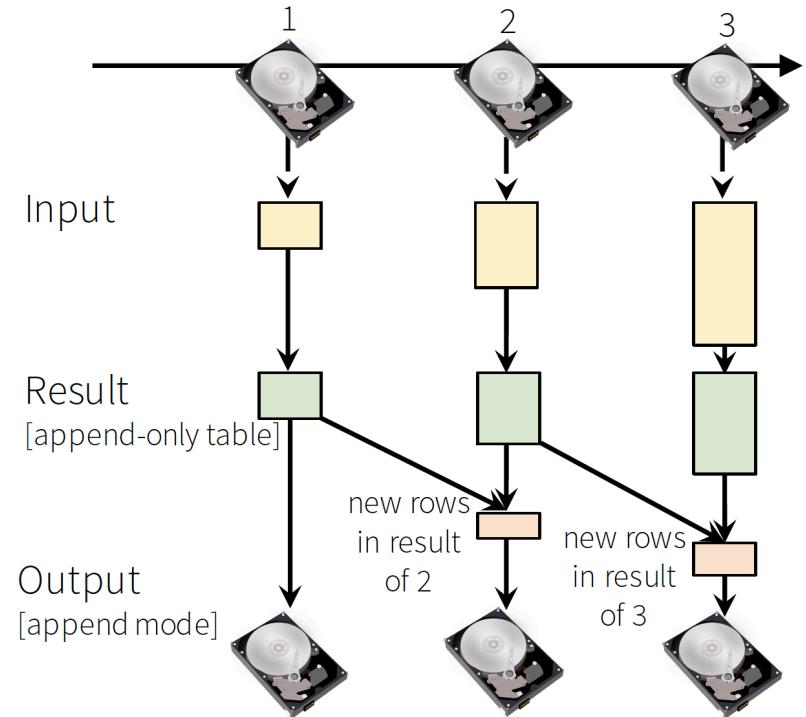
Streaming ETL with DataFrames

```
input = spark.read  
    .format("json")  
    .stream("source-  
path")  
  
result = input  
    .select("device",  
    "signal")  
    .where("signal >  
15")  
  
result.write  
    .format("parquet")  
    .startStream("dest-  
path")
```

- `read...stream()` creates a streaming DataFrame, does not start any of the computation
- `write...startStream()` defines where & how to output the data and starts the processing

Streaming ETL with DataFrames

```
input = spark.read  
    .format("json")  
    .stream("source-  
path")  
  
result = input  
    .select("device",  
    "signal")  
    .where("signal >  
15")  
  
result.write  
    .format("parquet")  
    .startStream("dest-  
path")
```



Continuous Aggregations

```
input.avg("signal")
```

- Continuously compute average signal across all devices

```
input.groupBy("device-type")
    .avg("signal")
```

- Continuously compute average signal of each type of device

Continuous Windowed Aggregations

```
input.groupBy(  
    $"device-type",  
    window($"event-  
time-  
    col", "10 min"))  
.avg("signal")
```

- Continuously compute average signal of each type of device in last 10 minutes using event-time
- Simplifies event-time stream processing (not possible in DStreams)
Works on both, streaming and batch jobs

Joining streams with static data

```
kafkaDataset = spark.read  
    .kafka("iot-updates")  
    .stream()
```

- Join streaming data from Kafka with static data via JDBC to enrich the streaming data ...

```
staticDataset = ctxt.read  
    .jdbc("jdbc://", "iot-  
device-info")
```

```
joinedDataset =  
    kafkaDataset.join(  
        staticDataset,  
        "device-type")
```

- ... without having to think that you are joining streaming data

Output modes

Defines what is outputted every time there is a trigger

Different output modes make sense for different queries

- Append mode with non-aggregation queries

```
input.select("device", "signal")
      .write
      .outputMode("append")
      .format("parquet")
      .startStream("dest-path")
```

- Complete mode with aggregation queries

```
input.agg(count("*"))
      .write
      .outputMode("complete"
)
      .format("parquet")
      .startStream("dest-path")
```

Query Management

```
query = result.write  
    .format("parquet")  
    .outputMode("append")  
)  
    .startStream("dest-  
path")
```

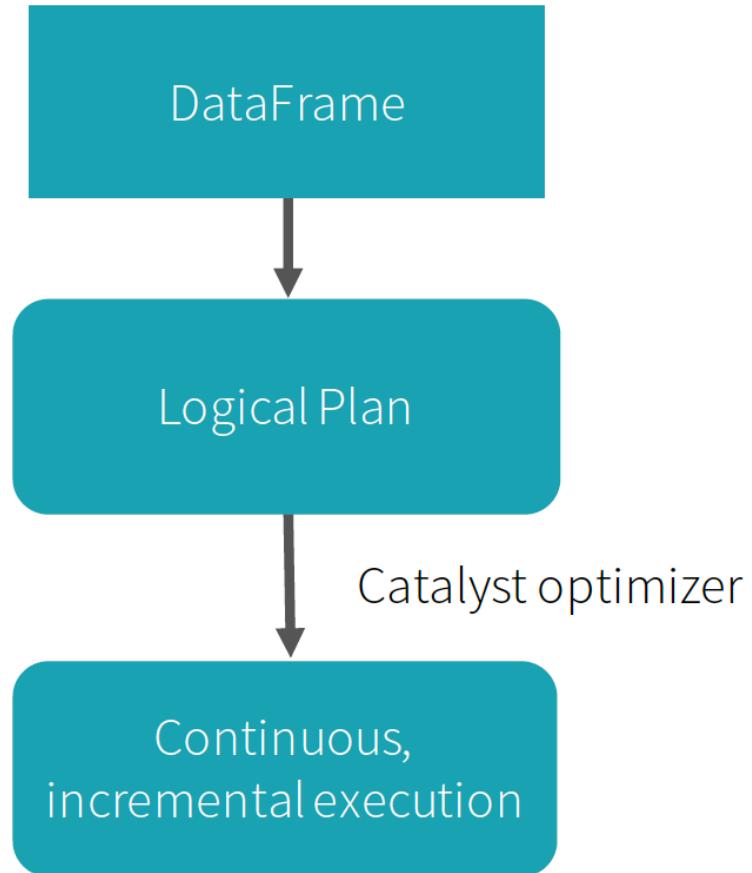
```
query.stop()  
query.awaitTermination()  
query.exception()
```

```
query.sourceStatuses()  
query.sinkStatus()
```

- query: a handle to the running streaming computation for managing it
 - Stop it, wait for it to terminate
 - Get status
 - Get error, if terminated
- Multiple queries can be active at the same time
- Each query has unique name for keeping track

Query execution

- Logically
 - Dataset operations on table (i.e. as easy to understand as batch)
- Physically
 - Spark automatically runs the query in streaming fashion (i.e. incrementally and continuously)

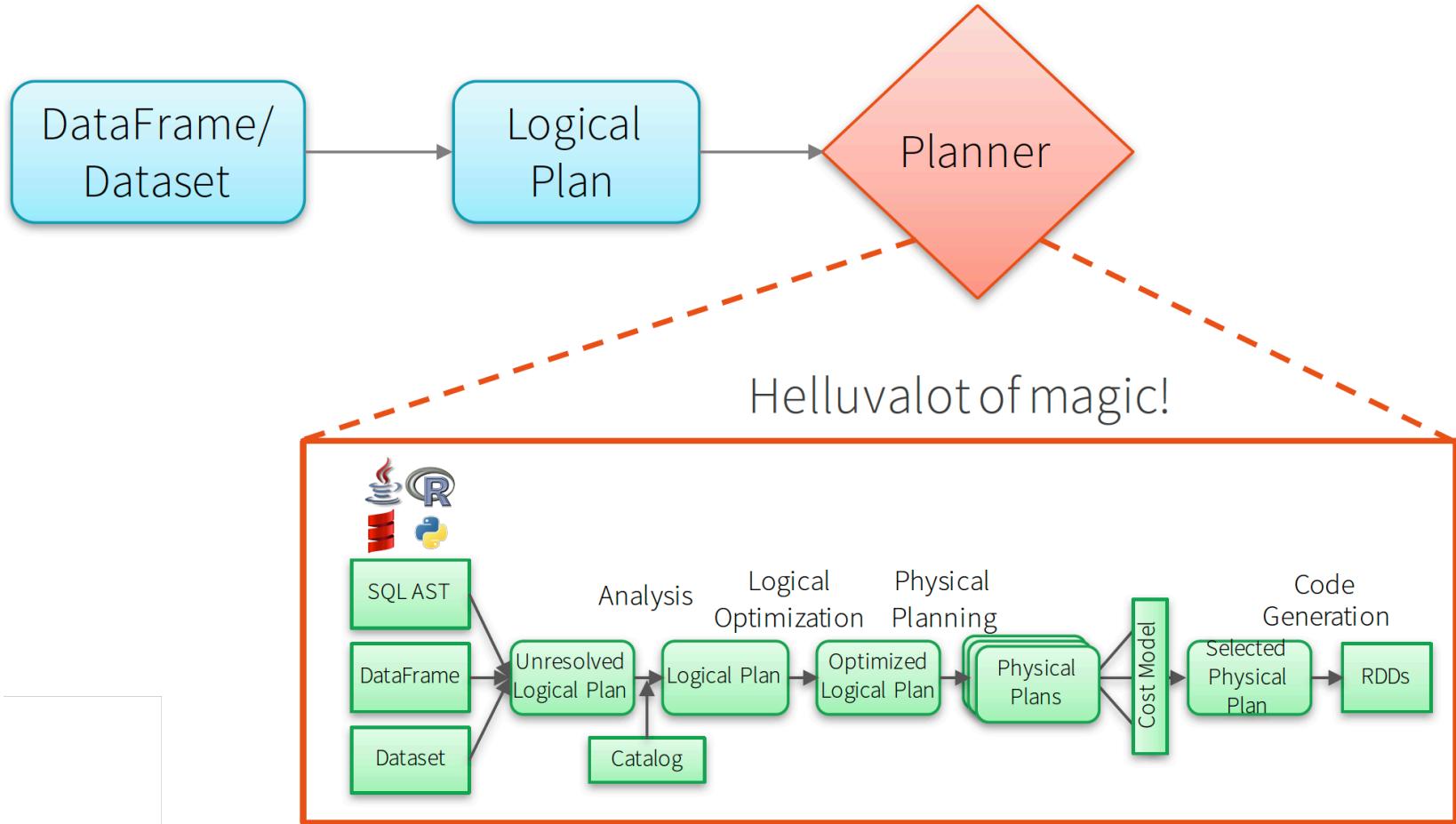


Structured Streaming

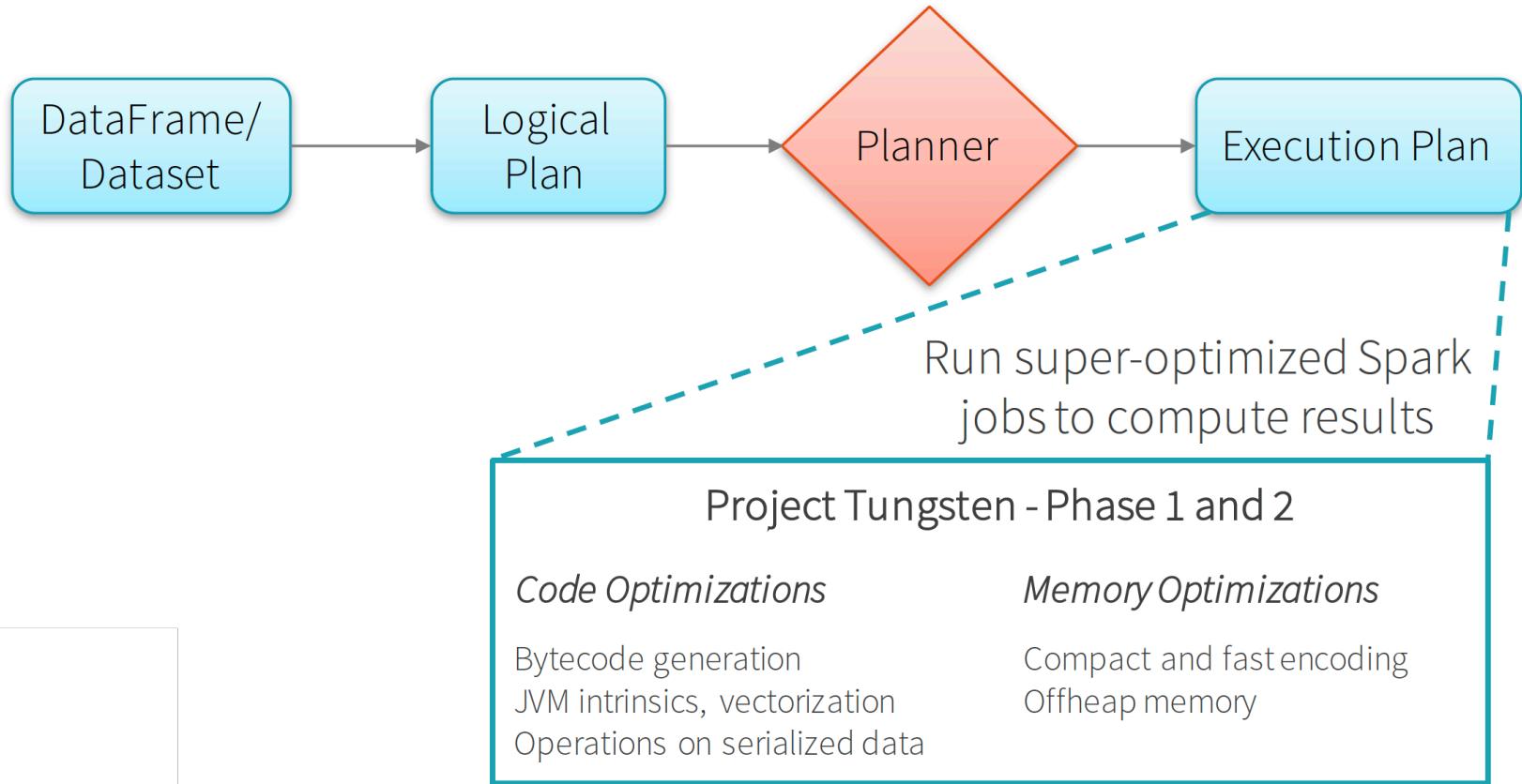
- High-level streaming API built on Datasets/DataFrames
 - Event time, windowing, sessions, sources & sinks
 - **End-to-end exactly once semantics**
- Unifies streaming, interactive and batch queries
 - Aggregate data in a stream, then serve using JDBC
 - Add, remove, change queries at runtime
 - Build and apply ML models

Internal execution

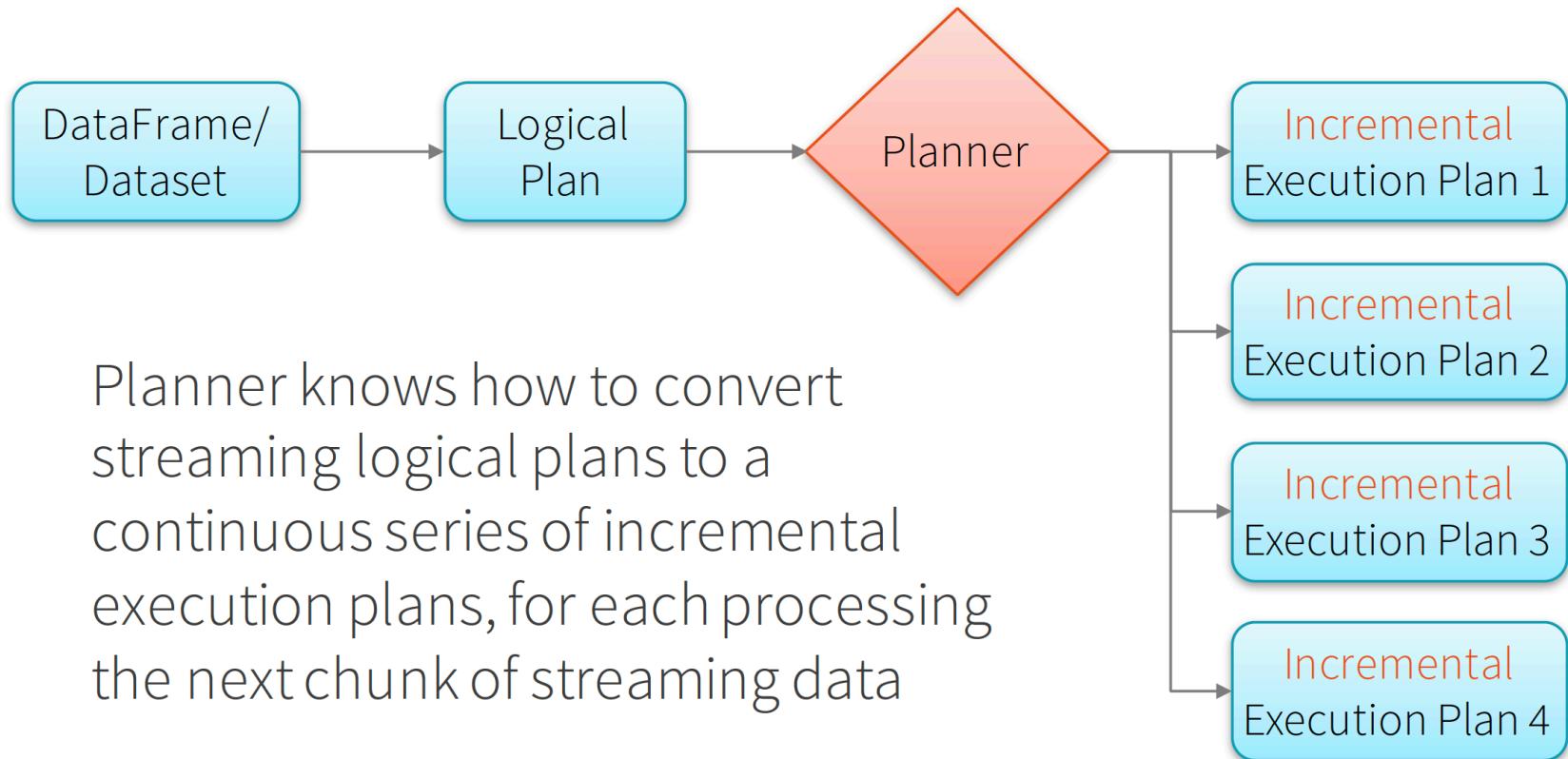
Batch Execution on Spark SQL



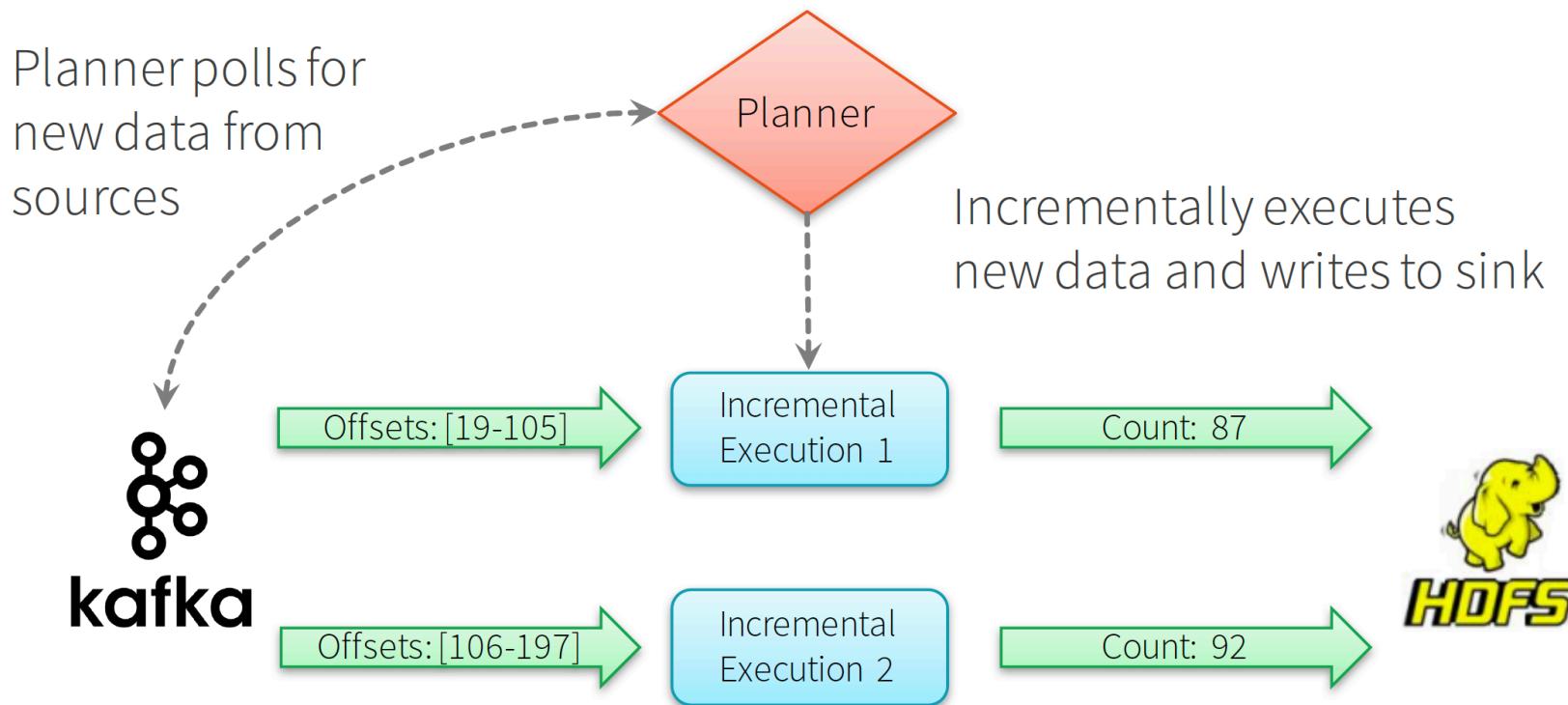
Batch Execution on Spark SQL



Continuous Incremental Execution

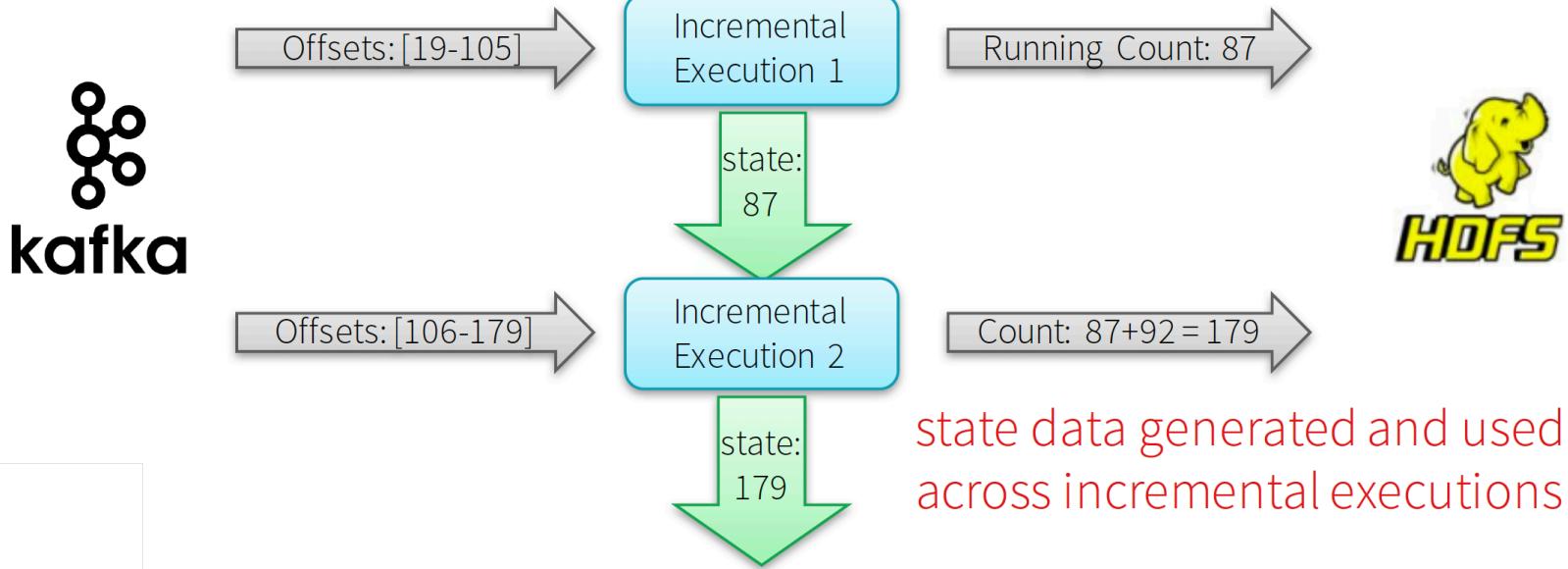


Continuous Incremental Execution



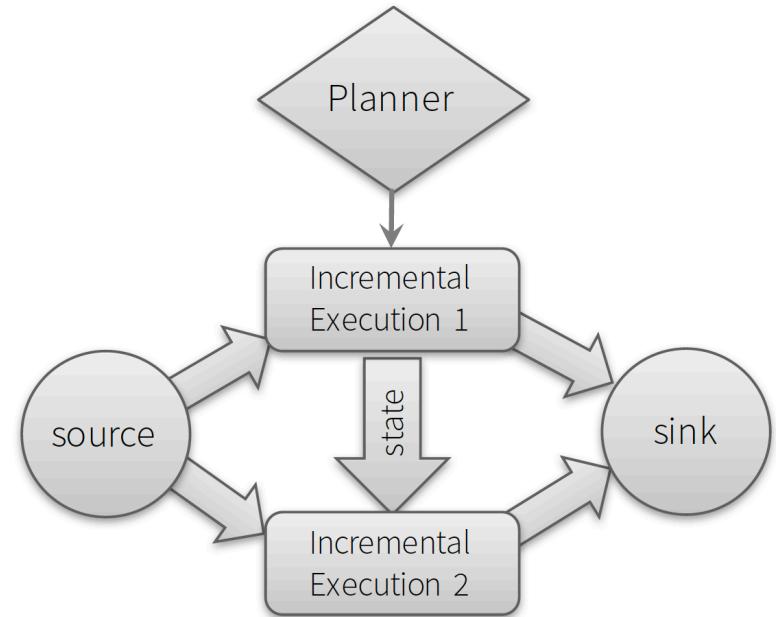
Continuous Aggregations

Maintain running aggregate as **in-memory state**
backed by **WAL in file system** for fault-tolerance



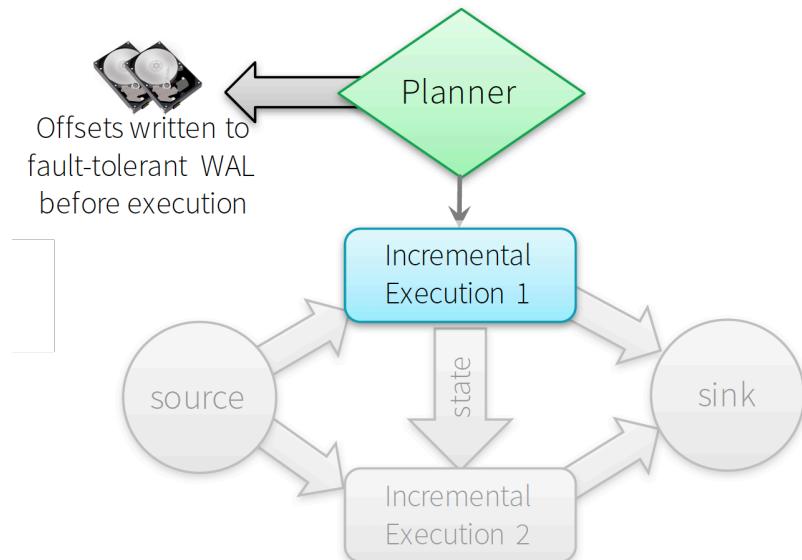
Fault-tolerance

- All data and metadata in the system needs to be recoverable / replayable



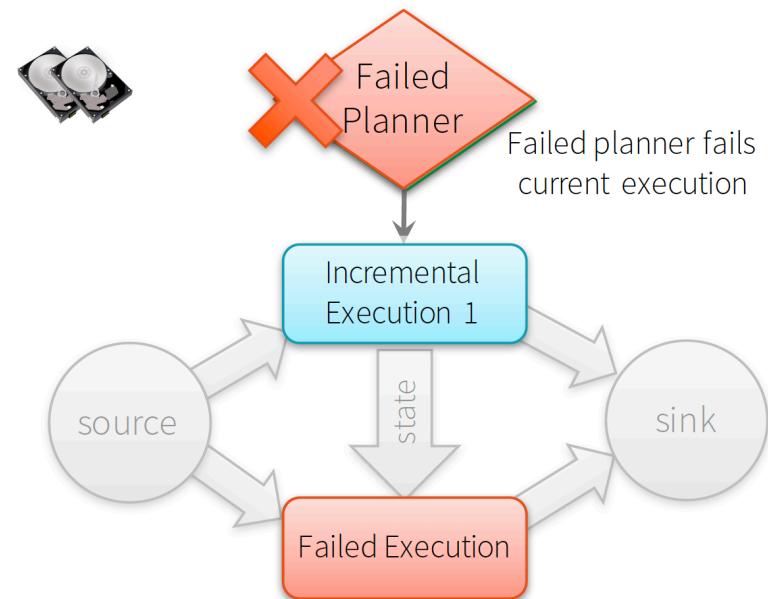
Fault-tolerant Planner

- Tracks offsets by writing the offset range of each execution to a write ahead log (WAL) in HDFS



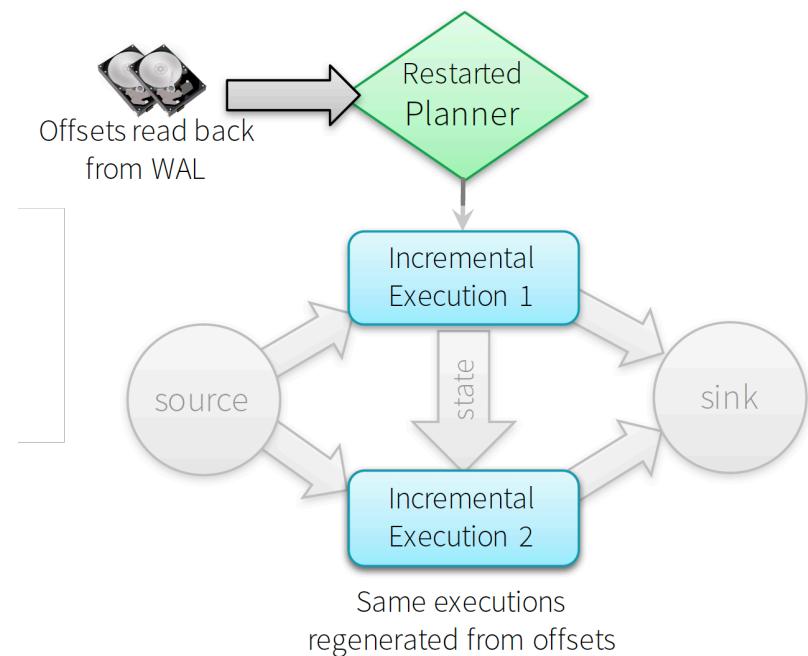
Fault-tolerant Planner

- Tracks offsets by writing the offset range of each execution to a write ahead log (WAL) in HDFS



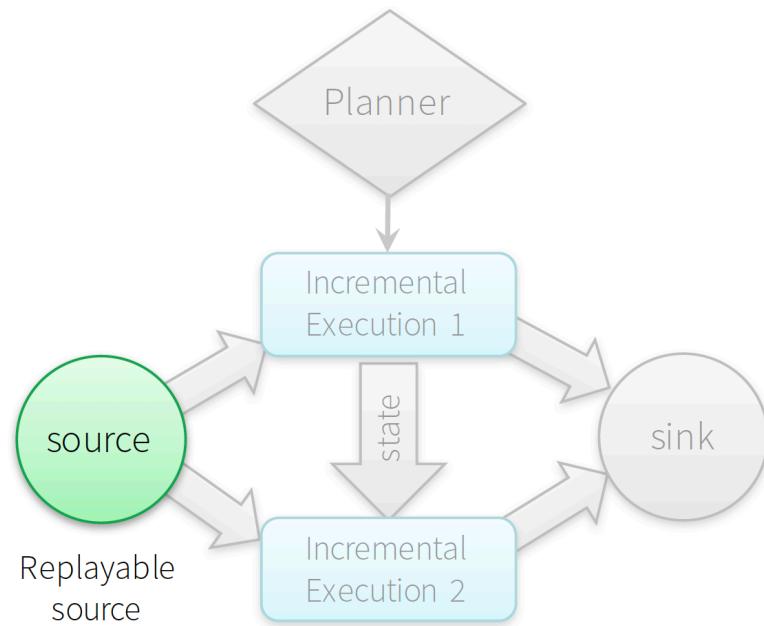
Fault-tolerant Planner

- Tracks offsets by writing the offset range of each execution to a write ahead log (WAL) in HDFS
- Reads log to recover from failures, and re-execute exact range of offsets



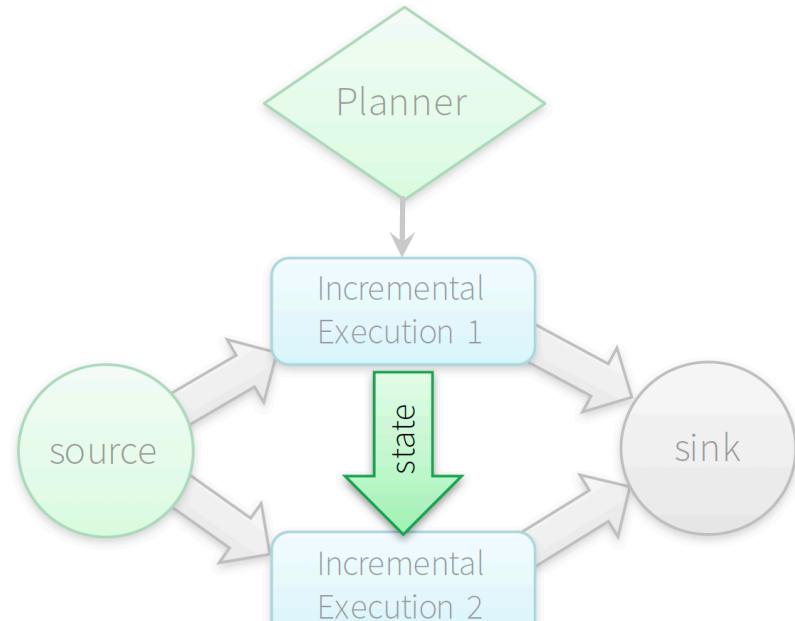
Fault-tolerant Sources

- Structured streaming sources are by design replayable (e.g. Kafka, Kinesis, files) and generate the exactly same data given offsets recovered by planner



Fault-tolerant State

- Intermediate "state data" is maintained in versioned, keyvalue maps in Spark workers, backed by HDFS
- Planner makes sure "correct version" of state used to reexecute after failure

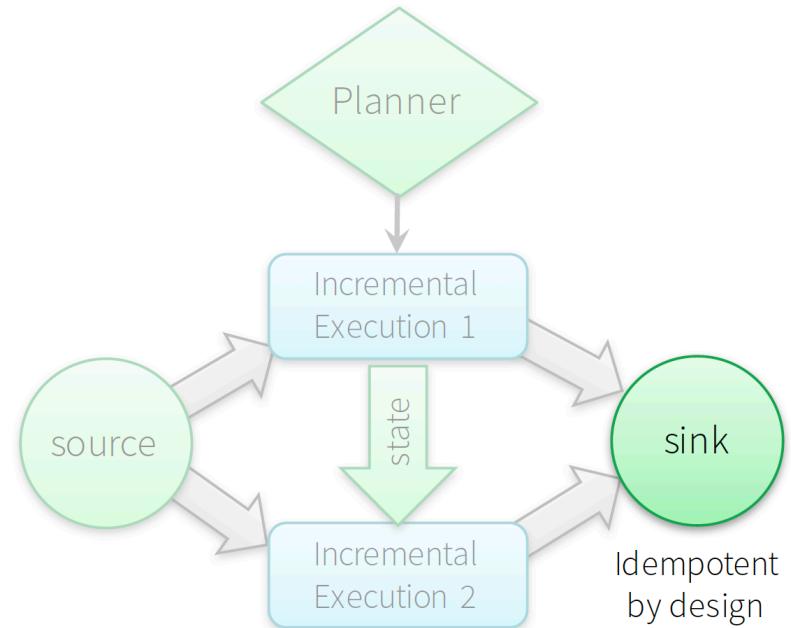


state is fault-tolerant with WAL



Fault-tolerant Sink

- Sink are by design idempotent (deterministic), and handles re-executions to avoid double committing the output



Fault-tolerance

offset tracking in WAL

+

state management

+

fault-tolerant sources and sinks

=

end-to-end
exactly-once
guarantees

Structured streaming

Fast, fault-tolerant, exactly-once
stateful stream processing
without having to reason about streaming

Usecase

- <https://mapr.com/blog/real-time-analysis-popular-uber-locations-spark-structured-streaming-machine-learning-kafka-and-mapr-db/>

Usecase – Twitter sentiment analysis



Trending Topics can be used to create campaigns and attract larger audience.

Sentiment Analytics helps in crisis management, service adjusting and target marketing.

- ❑ Sentiment refers to the emotion behind a social media mention online.
- ❑ Sentiment Analysis is categorising the tweets related to particular topic and performing data mining using Sentiment Automation Analytics Tools.
- ❑ We will be performing Twitter Sentiment Analysis as our Use Case for Spark Streaming.



The figure shows a comparison of trending topics on Facebook and Twitter. The Facebook section on the left displays a list of trending topics with small profile icons next to each. The Twitter section on the right shows a similar list, with the first item being a promoted post. Both sections include a 'TRENDING' heading and a list of hashtags.

Facebook Trends	Twitter Trends
Kourtney Kardashian: Kourtney Kardashian Confirms: 'I'm Pregnant'	#XboxE3 Promoted
Miss USA: Miss Nevada Nia Sanchez Crowned as 63rd Miss USA	Rik Mayall
Sir Mix-a-Lot: Sir Mix-a-Lot Takes 'Baby Got Back' Classical With Beatt...	The Young Ones
	#E32014
	#FireSideTieOnSaleNow
	Phantom Dust
	The Division
	Rise of the Tomb Raider
	#TonyAwards
	#SoundCloud

Figure: Facebook And Twitter Trending Topics



BAI HỌC
BẮC KHÔA
25 YEARS ANNIVERSARY
SOICT

SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

34

Problem statement



Problem Statement

To design a Twitter Sentiment Analysis System where we populate real time sentiments for crisis management, service adjusting and target marketing

Sentiment Analysis is used to:

- Predict the success of a movie
- Predict political campaign success
- Decide whether to invest in a certain company
- Targeted advertising
- Review products and services

Sentiment analysis for Nike



Figure: Twitter Sentiment Analysis For Nike

Sentiment analysis for Adidas



Figure: Twitter Sentiment Analysis For Adidas

Importing packages

```
//Import the necessary packages into the Spark Program
import org.apache.spark.streaming.{Seconds, StreamingContext}
import org.apache.spark.SparkContext.
import org.apache.spark.streaming.twitter.
import org.apache.spark.SparkConf
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext.
import org.apache.spark.
import org.apache.spark.rdd.
import org.apache.spark.rdd.RDD
import org.apache.spark.SparkContext.
import org.apache.spark.sql
import org.apache.spark.storage.StorageLevel
import scala.io.Source
import scala.collection.mutable.HashMap
import java.io.File
```

Twitter token authorization

```
object mapr {

  def main(args: Array[String]) {
    if (args.length < 4) {
      System.err.println("Usage: TwitterPopularTags <consumer key>
<consumer secret> " +
      "<access token> <access token secret> [<filters>]")
      System.exit(1)
    }

    StreamingExamples.setStreamingLogLevels()
    //Passing our Twitter keys and tokens as arguments for authorization
    val Array(consumerKey, consumerSecret, accessToken,
    accessTokenSecret) = args.take(4)
    val filters = args.takeRight(args.length - 4)
  }
}
```

Dstream transformation

```
// Set the system properties so that Twitter4j library used by twitter stream
// Use them to generate OAuth credentials
System.setProperty("twitter4j.oauth.consumerKey", consumerKey)
System.setProperty("twitter4j.oauth.consumerSecret", consumerSecret)
System.setProperty("twitter4j.oauth.accessToken", accessToken)
System.setProperty("twitter4j.oauth.accessTokenSecret",
accessTokenSecret)

val sparkConf = new
SparkConf().setAppName("Sentiments").setMaster("local[2]")
val ssc = new StreamingContext(sparkConf, Seconds(5))
val stream = TwitterUtils.createStream(ssc, None, filters)

//Input DStream transformation using flatMap
val tags = stream.flatMap { status =>
status.getHashtagEntities.map(_.getText)}
```

Generating tweet data

```
//RDD transformation using sortBy and then map function
tags.countByValue()
    .foreachRDD { rdd =>
    val now = org.joda.time.DateTime.now()
    rdd
        .sortBy(_.value)
        .map(x => (x, now))
    //Saving our output at ~/twitter/ directory
    .saveAsTextFile(s"~/twitter/$now")
}

//DStream transformation using filter and map functions
val tweets = stream.filter {t =>
    val tags = t.getText.split(" ")
        .filter(_.startsWith("#"))
        .map(_.toLowerCase)
    tags.exists { x => true }
}
```

Extracting sentiments

```
val data = tweets.map { status =>
    val sentiment = SentimentAnalysisUtils.detectSentiment(status.getText)
    val tagss = status.getHashtagEntities.map(_.getText.toLowerCase)
    (status.getText, sentiment.toString, tagss.toString())
}

data.print()
//Saving our output at ~/ with filenames starting like twitterss
data.saveAsTextFiles("~/twitterss", "20000")

ssc.start()
ssc.awaitTermination()
}
```

Results

Markers Properties Servers Data Source Explorer Snippets Console Scala Interpreter (TwitterStreaming)

<terminated> mapr\$ [Scala Application] /usr/lib/jvm/java-8-openjdk-i386/bin/java (09-Feb-2017, 11:56:26 AM)

debug: weighted: 1.0

Time: 1486621640000 ms

(東芝、半導体新棟を着工~メモリー製造、18年夏完成へ https://t.co/DUSgoZAp25 #不動産 #投資 #マネー #株 #市況 #拡散, NEGATIVE, [Ljava.lang.String;@1a25ec3])
(RT @bts_bighit: [투표] Q. 투표하라는 말 지겹다?
아이: 좋아요~ 짜릿해! » 놀 세로워! » #방탄소년단 투표하는 게 최고야!

#가온차트어워드 https://t.co/DHDWR2smt4
#ShortyAwards https://t.co/NEGATIVE, [Ljava.lang.String;@121986a)
(RT @MukePL: Jeżeli na tym zdjęciu widzisz swój świat to daj RT. ☺ #oneDBestfans & #550SBestfans ☺ https://t.co/rn2EmNvJFp, NEGATIVE, [Ljava.lang.String;@1c3681d)
(RT @Horoscasts: #Cancer most enduring quality is an unexpected silly sense of humor., POSITIVE, [Ljava.lang.String;@174ela2)
(I'm listening to "A Song For Mama" by @BoyzIIMen on @PandoraMusic. #pandora https://t.co/7in5Rw3CY0, NEUTRAL, [Ljava.lang.String;@95f6d4)
('Greenwashing' Costing Walmart \$1 Million https://t.co/DBX02RZhM# #Biodegradability #Compostability #biobased, NEGATIVE, [Ljava.lang.String;@151le25)
(RT @camillasxdinah: Serayah representando a las camilizers cuando un hombre se le acerca a Camila #CamilaBestFans https://t.co/8IggLo3RGn, NEGATIVE, [Ljava.lang.String;@78c835)
(RT @CamiLaVoteStats: #CamilaBestFans https://t.co/q5LxPQp0In, NEUTRAL, [Ljava.lang.String;@16e7255)
(@tos 六甲道駅 https://t.co/0rKl8rl5b3 #TFB, NEGATIVE, [Ljava.lang.String;@1a3fe)
(Ilmar pro Marcos: "Vai dormir puta.. Bebe e fica ai com o cu quente." KKKKKKKKKKKKKKKKKKKKKKKKKKK #BBB17, NEGATIVE, [Ljava.lang.String;@1516ece)
...
Adding annotator tokenize

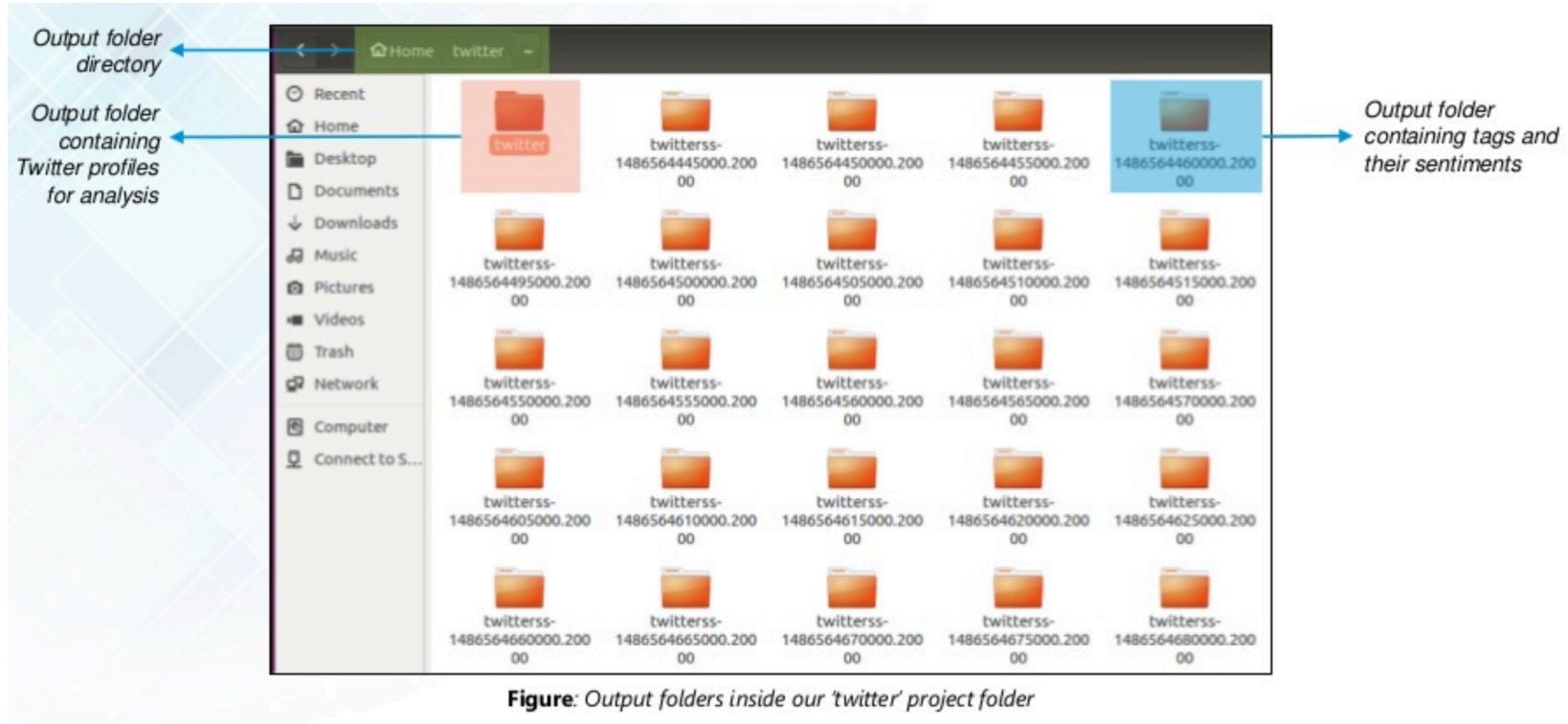
Positive

Neutral

Negative

Figure: Sentiment Analysis Output In Eclipse IDE

Output directory



Output usernames

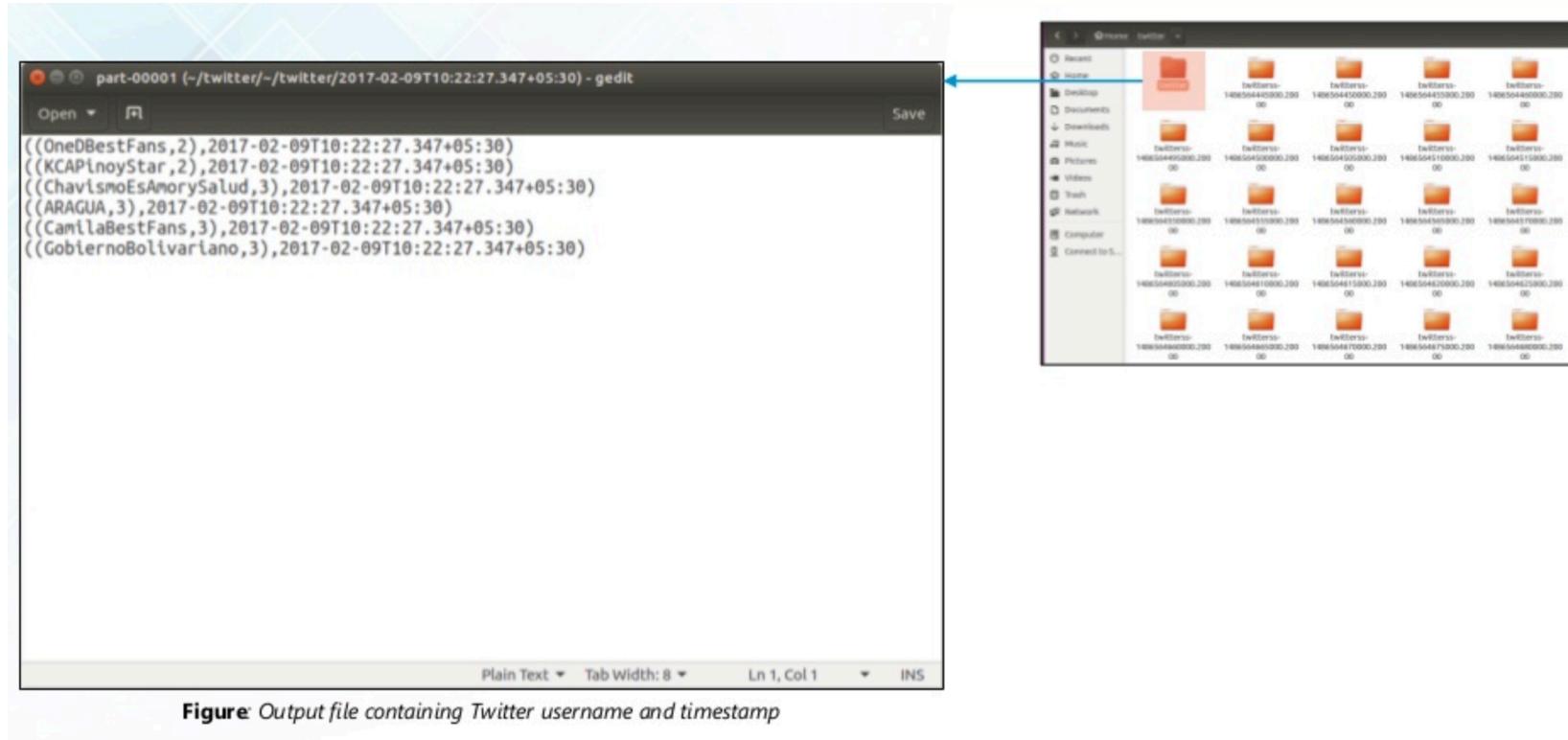
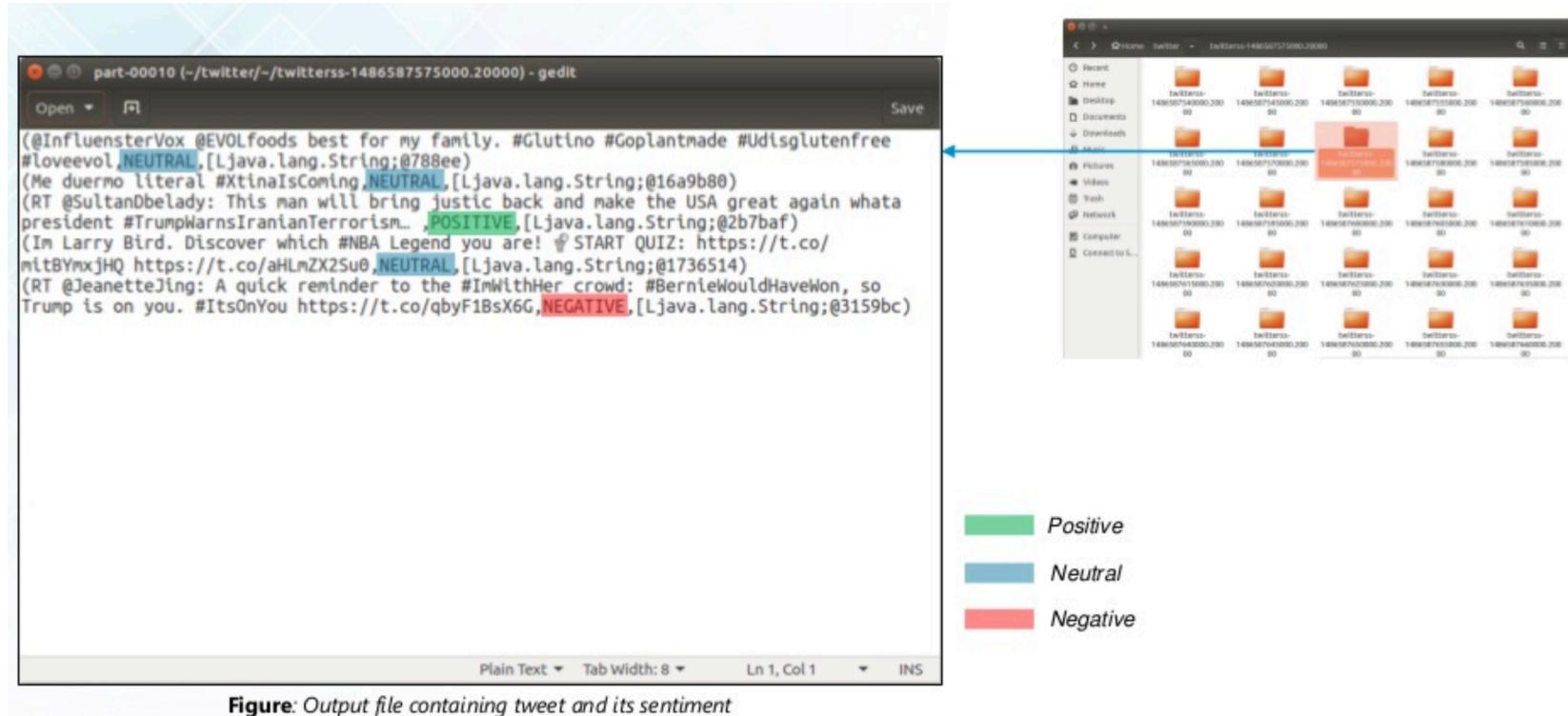


Figure: Output file containing Twitter username and timestamp

Output tweets and sentiments



Sentiments for Trump

The screenshot shows a Scala IDE interface. On the left, there are two tabs: 'mapr.scala' and 'earth.scala'. The 'mapr.scala' tab contains the following Scala code:

```
58     val tags = t.getText.split(" ").filter(_.startsWith("#")).map(_.toLowerCase)
59     tags.contains("#obama") && tags.contains("#NYC")
60   }
61 */
62 val tweets = stream.filter { t =>
63   val tags = t.getText.split(" ").filter(_.startsWith("Trump")).map(_.toLowerCase)
64   tags.exists { x => true }
65 }
66
67
68 val data = tweets.map { status =>
69   val sentiment = SentimentAnalysisUtils.detectSentiment(status.getText)
--
```

A yellow box highlights the line 'tags.exists { x => true }'. A blue arrow points from this box to the right margin, where the text 'Trump' Keyword is written.

The right margin also lists some Scala objects: 'import c', 'mapr', 'main(a)', 'cons1', 'cons2', 'acces', and 'X'. Below the code editor is a toolbar with icons for Markers, Properties, Servers, Data Source Explorer, Snippets, Console, and Scala Interpreter (Twitter). The 'Console' tab is selected.

The 'Console' tab displays the output of the application:

```
<terminated> mapr$ [Scala Application] /usr/lib/jvm/java-8-openjdk-i386/bin/java (09-Feb-2017, 6:06:24 PM)
ueuug: weighted: 3.0
-----
Time: 1486645210000 ms
-----
(#USA Trump Suggests That Supreme Court Nominee's Criticism of Him Misrepresented: Trump questioned whether...
https://t.co/1ZCtok4P43 #News, NEGATIVE, [Ljava.lang.String;@10f96b1)
( RT @WorldStarLaugh: Compilation of Donald Trump's greatest accomplishments as president https://t.co/
Got6efwiMH, POSITIVE, [Ljava.lang.String;@e5a4fe)
(BBCNewsnight: Should the UK roll out the red carpet for President Trump? Here's what Hillary Clinton's
campaign ma... https://t.co/hjKNuJJu3s, NEUTRAL, [Ljava.lang.String;@146dc81)
( RT @Jdxthompson: Ellen DeGeneres response to Donald Trump screening "Finding Dory" at The White House is
everything https://t.co/koQcPuH..., NEGATIVE, [Ljava.lang.String;@116c5fd)
( RT @calilelia: Trump: Ivanka "always pushing me to do the right thing." He needs a push to do the right
thing? @ananavarro @VanJones68 @Ch..., NEUTRAL, [Ljava.lang.String;@129dc11)
```

To the right of the console output is a legend:

- Positive (Green square)
- Neutral (Blue square)
- Negative (Red square)

Figure: Performing Sentiment Analysis on Tweets with 'Trump' Keyword

Applying sentiment analysis

- ❑ As we have seen from our Sentiment Analysis demonstration, we can extract sentiments of particular topics just like we did for 'Trump'.
- ❑ Hence Sentiment Analytics can be used in **crisis management**, **service adjusting** and **target marketing** by companies around the world.



Companies using **Spark Streaming** for Sentiment Analysis have applied the same approach to achieve the following:

1. Enhancing the customer experience
2. Gaining competitive advantage
3. Gaining Business Intelligence
4. Revitalizing a losing brand



References

- Zaharia, Matei, et al. "Discretized streams: an efficient and fault-tolerant model for stream processing on large clusters." *Presented as part of the*. 2012.
- Armbrust, Michael, et al. "Structured streaming: A declarative API for real-time applications in apache spark." *Proceedings of the 2018 International Conference on Management of Data*. 2018.



25
YEARS ANNIVERSARY
SOICT

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

Thank you
for your
attention!!!

