ISIT312 Big Data Management

Spark Stream Processing

Dr Fenghui Ren

School of Computing and Information Technology -
University of Wollongong

Spark Structured Data and Stream Processing

Outline

Spark Stream Processing Modules

Stream processing is a key requirement in many big data applications

As soon as an application computes something of value, for example, a report or a machine learning model, an organization may want to compute this result continuously in a production environment

This capability is lacked in Hadoop MapReduce framework due to slowness of hard-disk IO

In-memory computation implemented in Spark make stream processing possible

Spark Streaming based on its low-level API Resilient Distributed Dataset is available since Spark 1.2

Spark Structured Streaming based on the Spark SQL engine is available since Spark 2.1

Luckily, our VM has installation of Spark 2.1.1

Spark Structured Data and Stream Processing

Outline

Stream Processing

Stream processing is the act of continuously incorporating the new data in the stream to compute a result

Sample sources of streams

- Bank transactions
- Clicks on a website
- Sensor readings from IoT devices
- Scientific observations and experiments
- Manufacturing processes, and the others

Stream processing vs. batch processing

- Batch processing runs to a fixed set of data, but stream processing handles an unbounded set of data
- Batch processing has low timeliness requirement, but stream processing requires to work at near realtime

Stream Processing

Use cases of stream processing

- Notifications and alerting
- Real-time reporting
- Incremental ETL
- Update data to serve in real time
- Real-time decision making
- Online machine learning

Stream Processing

To see the challenges of stream processing, we consider the following example

Suppose we received the following data from a sensor

Sample data

```
{value: 1,  time: "2017-04-07T00:00:00"}
{value: 2,  time: "2017-04-07T01:00:00"}
{value: 5,  time: "2017-04-07T02:00:00"}
{value: 10, time: "2017-04-07T01:30:00"}
{value: 7,  time: "2017-04-07T03:00:00"}
```

What actions should be performed when receiving single values, say, `5` ?

How to react to a pattern, say, `2` -> `10` -> `5`

What if data arrives out-of-order, for example, `10` before `5`

Other issues: What if a machine in the system fails, losing some state? What if the load is imbalanced? How can an application signal downstream consumers when analysis for some event is done, and so on

Stream Processing

Main challenges of stream processing are the following

- Processing out-of-order data based on application timestamps (also called event time)
- Maintaining large amounts of states
- Supporting high-data throughput
- Processing each event exactly once despite machine failures
- Handling load imbalance and strugglers
- Responding to events at low latency
- Joining with external data in other storage systems
- Determining how to update output sinks as new events arrive
- Writing data transactionally to output systems
- Updating application business logic at runtime

Spark Structured Data and Stream Processing

Outline

Structured Stream Processing: Quick Start

Structured Streaming Processing suppose to provide fast, scalable, fault-tolerant, end-to-end exactly-once stream processing without the user having to reason about streaming

A streaming version of the word-count example

Reading a stream

```
val lines = spark.readStream
                 .format("socket")            // socket source
                 .option("host", "localhost") // listen to the localhost
                 .option("port", 9999)        // and port 9999
                 .load()
```

Importing methods

```
import spark.implicits._
```

sql

```
val words = lines.as[String].flatMap(_.split(" "))
```

Grouping

```
val wordCounts = words.groupBy("value").count()
```

Writing stream

```
val query = wordCounts.writeStream
                      .outputMode("complete") // accumulate the counting result
                      .format("console")      // use the console as the sink
                      .start()
```

Structured Stream Processing: Quick Start

The input is simulated by Netcat (a small utility found in most Unix-like systems) as a data server

Starting Netcat

```
nc -lk 9999
```

In a different Terminal, we start Spark-shell and input the Scala code from the previous slides

If we input in the first Terminal session

Starting Netcat

```
nc -lk 9999
apache spark
apache hadoop
...
```

Structured Stream Processing: Quick Start

Then we should see the right hand-side output in Spark-shell

```
-------------------------------------------
Batch: 0
-------------------------------------------
+------+-----+
| value|count|
+------+-----+
|apache|    1|
| spark|    1|
+------+-----+


-------------------------------------------
Batch: 1
-------------------------------------------
+------+-----+
| value|count|
+------+-----+
|apache|    2|
| spark|    1|
|hadoop|    1|
+------+-----+
...
```

Spark Structured Data and Stream Processing

Outline

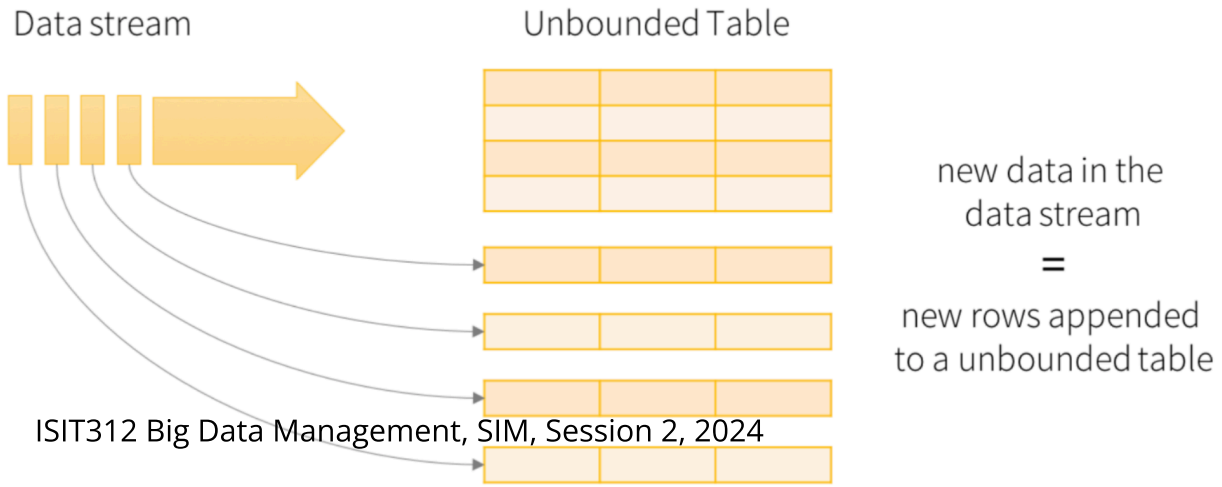
Programming Model

The key idea in Structured Streaming is to treat a live data stream as a table that is being continuously appended

This leads to a new stream processing model that is very similar to a batch processing model

Users can express the streaming computation as standard batch-like query as on a static table, and Spark runs it as an incremental query on the unbounded Input Table

A new data item arriving on the stream is like a new row being appended to Input Table



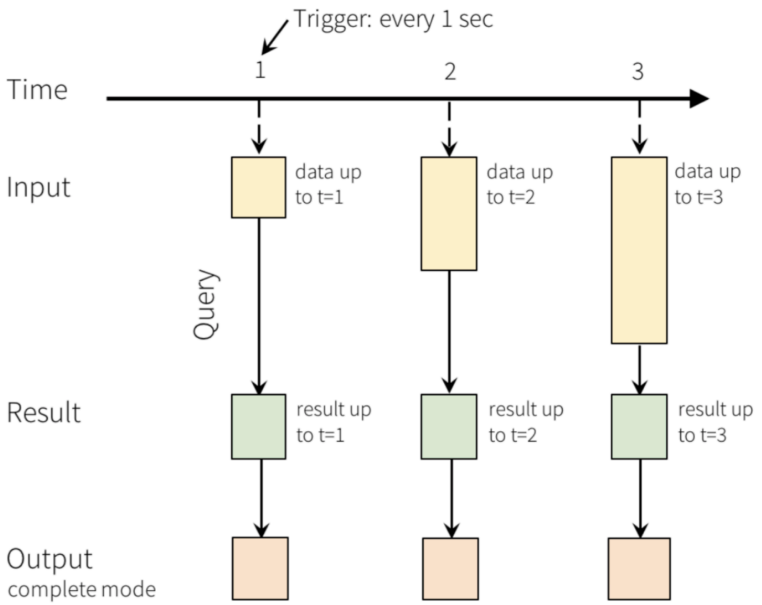
Programming Model

A query on the input will generate Result Table

Every trigger interval, let us say, every X seconds, the new rows get appended to Input Table

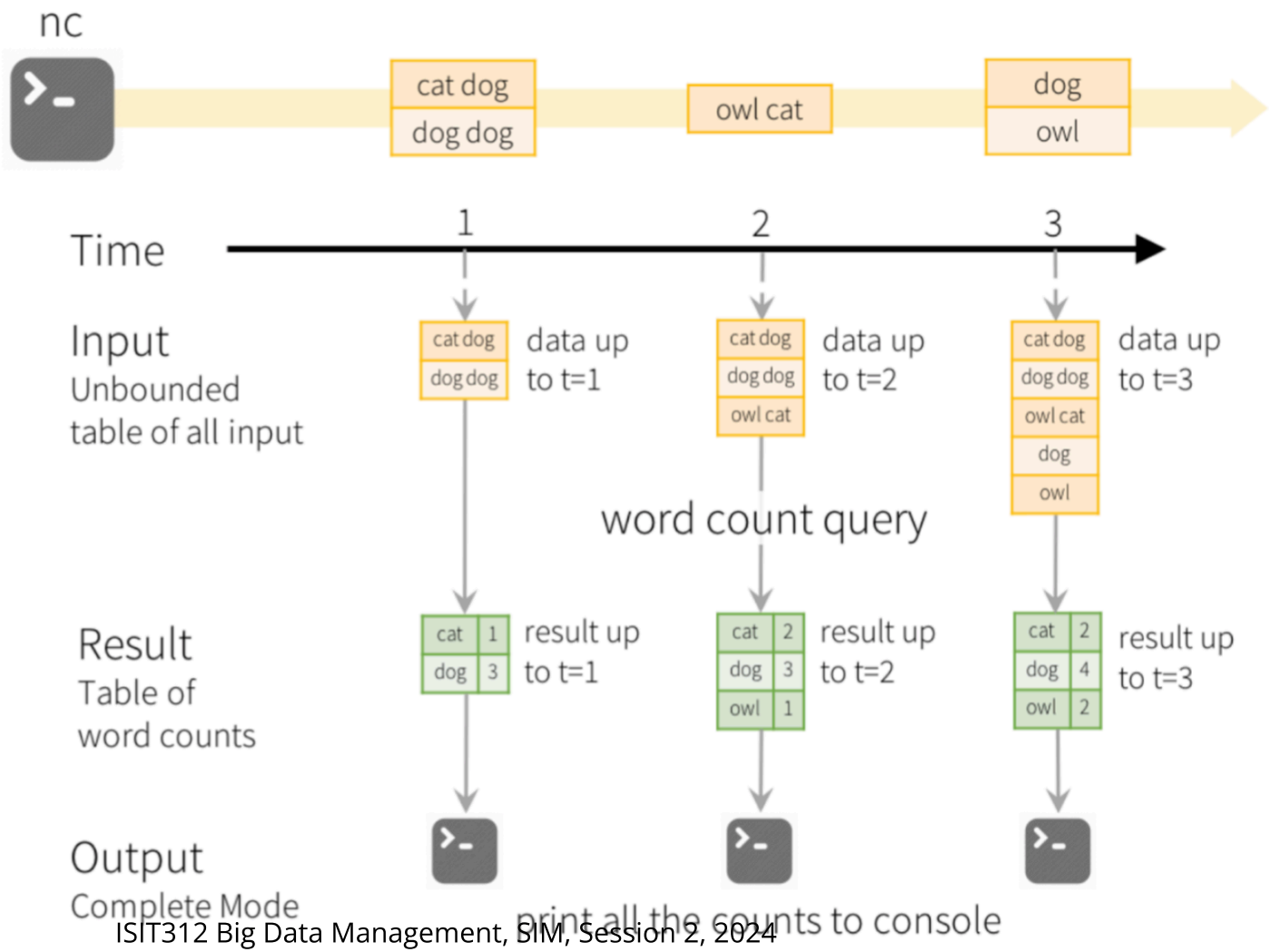
It will eventually updates Result Table

Whenever Result Table gets updated, we would want to write the changed result rows to an external sink



Programming Model

A complete process



References

A Gentle Introduction to Spark, Databricks, (Available in `READINGS` folder)

RDD Programming Guide

Spark SQL, DataFrames and Datasets Guide

Karau H., Fast data processing with Spark Packt Publishing, 2013 (Available from UOW Library)

Srinivasa, K.G., Guide to High Performance Distributed Computing: Case Studies with Hadoop, Scalding and SparkSpringer, 2015 (Available from UOW Library)

Chambers B., Zaharia M.,Spark: The Definitive Guide, O'Reilly 2017

Perrin J-G., Spark in Action, 2nd ed., Manning Publications Co. 2020