GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Exploratory Analysis of Spark Structured Streaming
[Work-in-Progress]

Todor Ivanov todor@dbis.cs.uni-frankfurt.de
Jason Taaffe jasontaaffe@yahoo.de

Frankfurt Big Data Lab
GOETHE UNIVERSITÄT

Goethe University Frankfurt am Main, Germany
http://www.bigdata.uni-frankfurt.de/

Motivation

- Increasing popularity and variety of Stream Processing Engines such as Spark Streaming, Flink Streaming, Samza, etc.
- Growing support for stream SQL features such as Spark Structured Streaming, Flink SQL, Samza SQL, etc.

Research Objectives

- Evaluate the features of Spark Structured Streaming
- Prototype a benchmark for testing stream SQL based on BigBench
- Perform experiments validating the prototype benchmark and exploring the Spark Structured Streaming capabilities/features

Spark Structured Streaming (introduced in Spark 2.0)

- *Structured Streaming provides* **fast, scalable, fault-tolerant, end-to-end exactly-once** *stream processing without the user having to reason about streaming.*

- built and executed on top of the Spark SQL engine

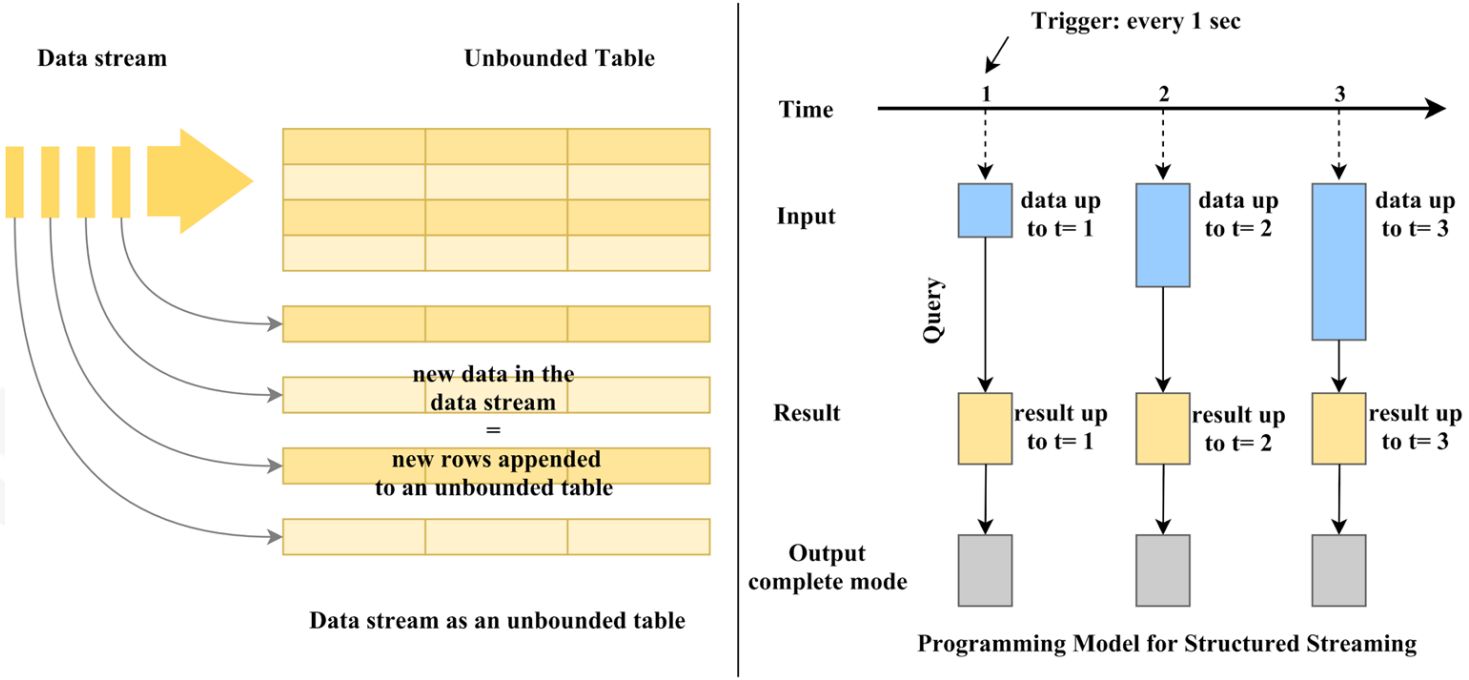
- use the Dataset/DataFrame API in Scala, Java, Python or R to express streaming aggregations, event-time windows, stream-to-batch joins, etc

- Micro-batch processing mode
  - processes data streams as a series of small batch jobs thereby achieving ***end-to-end latencies as low as 100 milliseconds*** and exactly-once fault-tolerance guarantees

- Continuous processing mode (introduced in Spark 2.3)
  - achieve ***end-to-end latencies as low as 1 millisecond*** with at-least-once guarantees (future work)

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

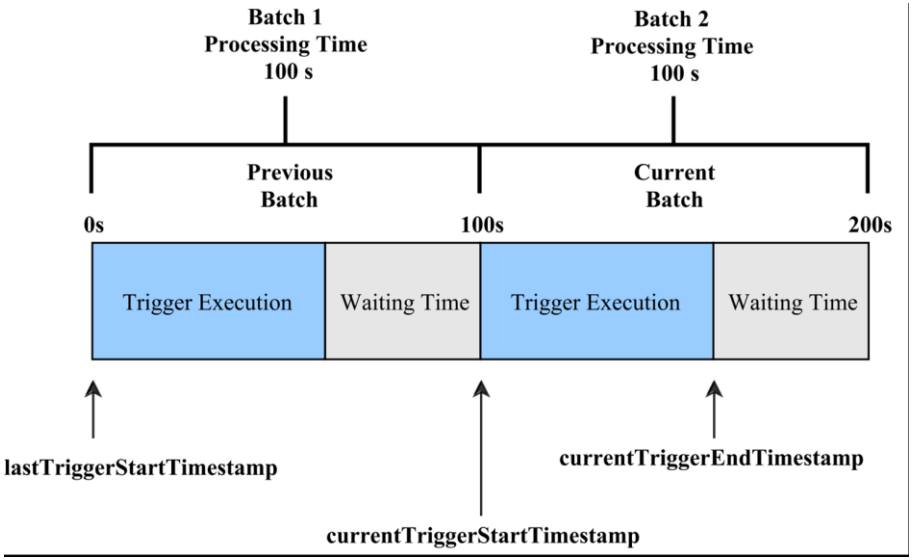
Spark Structured Streaming (2)



"The key idea in Structured Streaming is to ***treat a live data stream as a table*** that is being continuously appended. [...] Every data item that is arriving on the stream is like a row being appended to the Input Table." [https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html]

"... at any time, the output of the application is equivalent to executing a batch job on a prefix of the data." [https://databricks.com/blog/2016/07/28/structured-streaming-in-apache-spark.html].

Trigger Execution Metrics



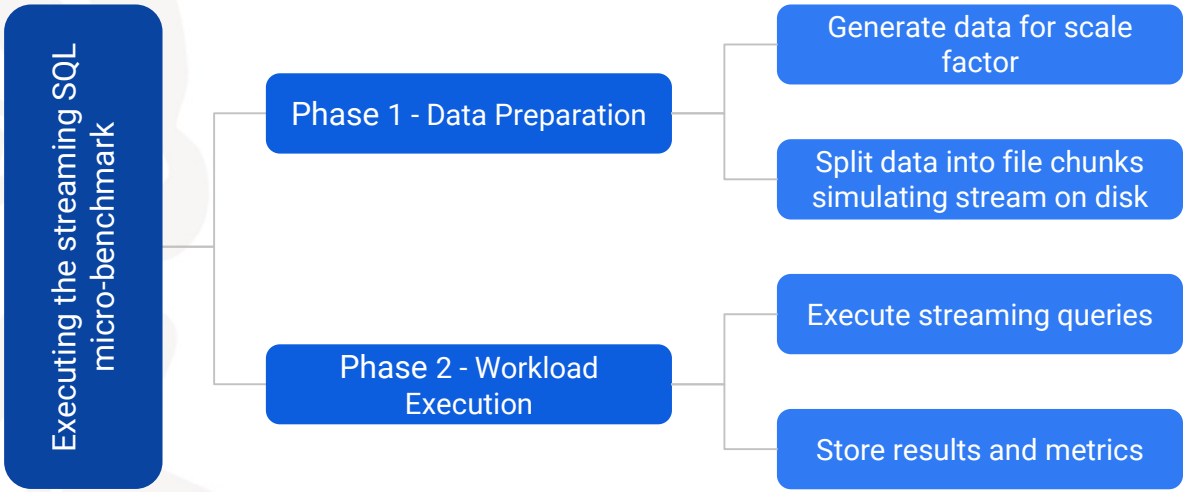
Name	Definition	Description
Latency	triggerExecution - The amount of time taken to perform various operations in milliseconds.	Describes the amount of time needed to perform the SQL query operations during one trigger interval.
InputRate-total	numRecords/inputTimeSec	Describes how many rows were loaded per second between the start of the last trigger and the start of the current trigger.
ProcessingRate-total	numRecords/processingTimeSec	Describes how many rows were processed per second during the start of the current trigger and the end of the current trigger.
InputTimeSec	currentTriggerStartTimestamp - lastTrigger-StartTimestamp	The period of time between the start of the current trigger and the start of the last trigger.
ProcessingTimeSec	currentTriggerEndTimestamp - currentTrigger-StartTimestamp	The period of time between the beginning and the end of a trigger period.
NumInputRows	NumRecords	The number of rows in a batch.

Structured Streaming Micro-Benchmark

- Popular streaming benchmarks such as HiBench, SparkBench and Yahoo Streaming Benchmark
  → do not stress test the streaming SQL capabilities of engines.

- Idea: use BigBench [1,2] as a base for new streaming SQL micro-benchmark

- Simplified 2 phase benchmark methodology
  - Phase 1: generate and prepare the data
  - Phase 2: execute the continuous workloads and validate the results



Workloads

- 5 queries were chosen for the evaluation

- 4 of them are from BigBench V2 and represent relevant streaming questions
  - Q05 : Find the 10 most browsed products in the last X seconds.
  - Q06 : Find the 5 most browsed products that were not purchased across all users (or only specific user) in the last X seconds.
  - Q16 : Find the top ten pages visited by all users (or specific user) in the last X seconds.
  - Q22 : Show the number of unique visitors in the X seconds.

- 1 new query performing simple monitoring operation:
  - Qmilk : Show the sold products (of a certain product or category) in the last X seconds.

Implementation and Limitations

- Q05, Q16 and Qmilk were successfully implemented.
  - Spark code: https://github.com/Taaffy/Structured-Streaming-Micro-Benchmark

```
var web_logs_05 = web_logsDF
.groupBy ( "wl _ item_id" ).count( )
.orderBy ( "count" )
.select ( "wl _item_id" , "count" )
.where ( "wl _ item _ id ␣ IS ␣ NOT ␣ NULL " )
```

```
var web_logs_16 = web_logsDF
.groupBy ( " wl_webpage_name " ) . count ( )
.orderBy ( "count " )
.select ( " wl_webpage_name " , " count " )
.where ( " wl_webpage_name ␣ IS ␣ NOT ␣ NULL "
)
```

```
var web _ sales _ milk = web_salesDF
.groupBy ( "ws _ product _ id") . count ( )
.orderBy ( "ws _ product_ id")
.select ( "ws _ product _ id " , " count ")
.where ( "ws_product_ID ␣ IS ␣ NOT ␣ NULL “)
```

- Q06 - performs ***join operations on two streaming datasets***, which was not supported
  - however, the latest version supports stream-stream joins (https://databricks.com/blog/2018/03/13/introducing-stream-stream-joins-in-apache-spark-2-3.html)
- Q22 - uses a ***distinct operation***, ***sorting operation*** (Order By) and ***Limit*** keyword, which were not supported at the time of implementation

Benchmarking Setup

- Hardware Setup:
  - Intel Core i5 CPU 760 @3.47GHz x 4
  - 8 GB RAM & 1 TB Hard Disk

- Software
  - Ubuntu 16.04 LTS OS
  - Java Version 1.8.0_131 & Scala Version 2.11.2
  - Apache Spark 2.3 - Standalone with Default parameters

Data Preparation and Execution

- Generate data with the BigBench V2 data generator for scale factor 1
  - ***web logs*** consisting of ***web clicks in JSON format*** (around 20GB)
  - ***web sales*** structured data (around 10MB)

- ***Web logs*** with variable file sizes ranging from 50MB to 2 GB were created.

- For every file size, 10 files were created with subsequent timestamps to simulate a stream of 10 data files.

- Q05 and Q16 were tested with varied ***web logs*** files.

- Qmilk was tested 10MB ***web sales*** file copied 10 times to ensure 10 files could be streamed successfully.

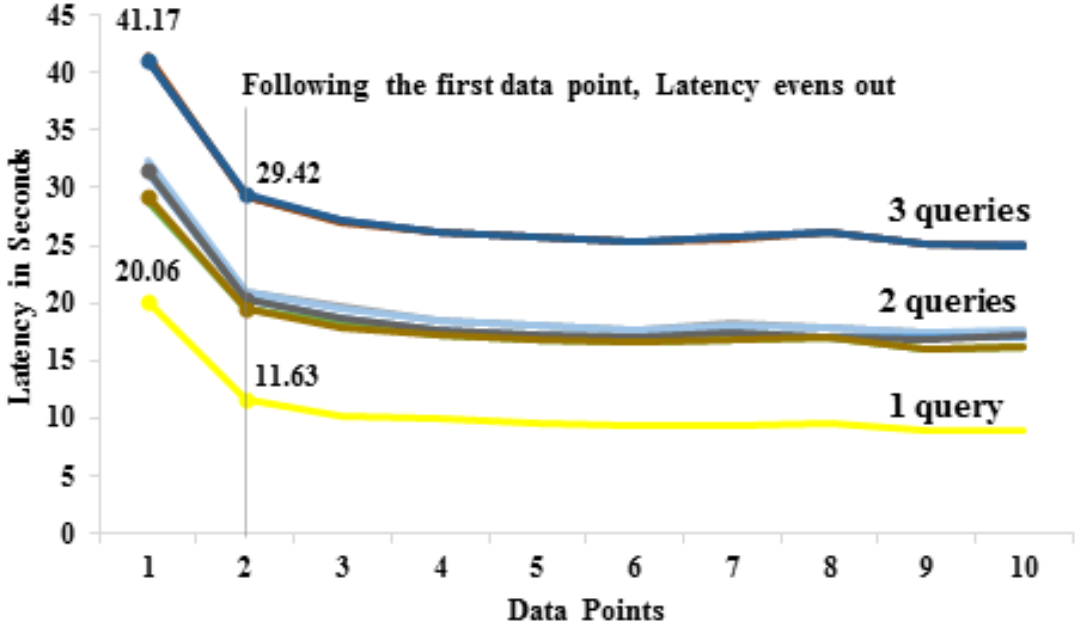
Performed Experiments

- Query and File Size Combinations (total of 31 combinations)

Query/File Size	10 MB*	50 MB	500 MB	1 GB	1.5 GB	2 GB
Q05		x	x	x	x	x
Q16		x	x	x	x	x
Qmilk	x (web sales)	x	x	x	x	x
Q05 | Q16		x	x	x	x	x
Q05 | Qmilk		x	x	x	x	x
Q16 | Qmilk		x	x	x	x	x
Q05 | Q16 | Qmilk		x	x	x	x	x

Exploratory Analysis

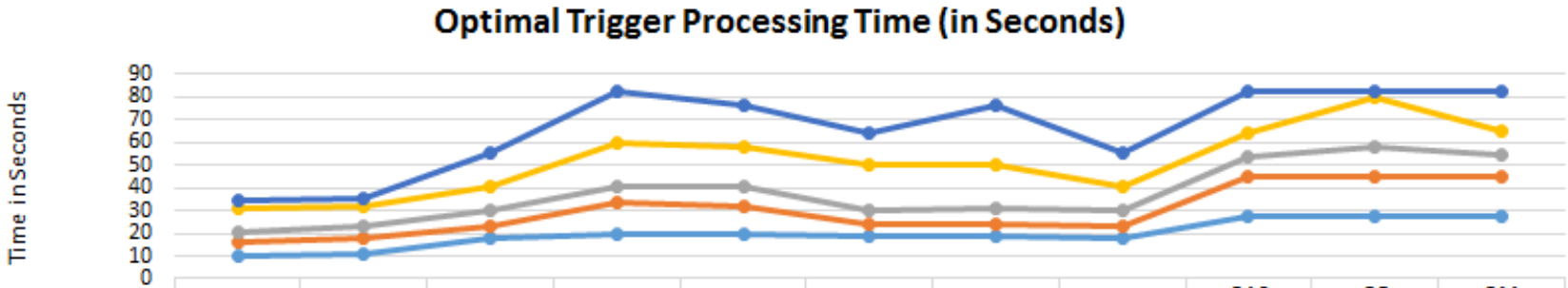
- Warmup run for all experimental executions taken into account. (excluded from measures)



- After the second execution the average query latency becomes constant in all 3 cases.

Average Processing Times

- 3 groups of queries:
  - single executions - Q05 and Q16
  - parallel pair executions - Q05, Q16 and Qmilk
  - triple parallel executions - Q05, Q16 and Qmilk



	Q16	Q5	Q16 (16/M)	Q16 (5/16)	Q5 (5/16)	Q5 (5/M)	QM (5/M)	QM (16/M)	Q16 (5/16/M)	Q5 (5/16/M)	QM (5/16/M)
50 MB	10	11	18	19	19	18	18	18	27	27	27
500MB	16	18	24	34	32	24	24	24	45	45	45
1000MB	21	23	30	40	40	31	31	30	54	58	54
1500MB	31	32	41	60	58	51	51	41	65	80	65
2000MB	34	35	56	82	76	65	76	55	82	83	82

- In general, it can be said that the more queries are run in parallel and the larger the file sizes used, ***the longer these take***, to be completed and ***the more resource intensive they are***.

Other Observations

- ***Average Latency*** increased by more than 200% between the 50MB and 2 GB file sizes.
- Comparing ***single query latency*** to triple query latency, it increased 2 to 3-fold.
- CPU utilization was the ***highest*** at the 50MB level and ***decreased every time*** the file size was increased.
- The JAVA heap size ***increased for larger file sizes***, making the memory size to be the limiting factor when using larger files.

Experimental Limitations

- File Creation Date Issue
  - It appeared that when all the streaming files were copied to the streaming directory and had the same file creation or modification date, Spark considered these to be the same file even if the files had different names (web-logs1, web-logs2, etc.).

- Files larger than 2GB and with more than 10 file streams should be investigated in future research for potential bottlenecks and peak performance.

- Spark in Cluster mode should be tested and evaluated with the end-to-end BigBench execution.

Lessons Learned

- Pros
  - Simple programing model and streaming API
  - Several built-in metrics
  - Several extraction and sink options for metrics
  - Possible to run queries in parallel

- Cons
  - Undocumented trigger process
  - Query limitations due to streaming API (Syntax)
  - Complicated metric extraction process

Future Work

- Extend the benchmark with more queries and implementations on other engines (Flink, Samza, etc.).
- Build automated benchmark kit with all executable components (as part of ABench).
- Perform experiments with larger file sizes on multi-node Spark cluster.

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Thank you for your attention!

References

[1] Ahmad Ghazal, Todor Ivanov, Pekka Kostamaa, Alain Crolotte, Ryan Voong, Mohammed Al-Kateb, Waleed Ghazal, and Roberto V. Zicari. 2017. BigBench V2: The New and Improved BigBench. In ICDE 2017, San Diego, CA, USA, April 19-22.

[2] Ahmad Ghazal, Tilmann Rabl, Minqing Hu, Francois Raab, Meikel Poess, Alain Crolotte, and Hans-Arno Jacobsen. 2013. BigBench: Towards An Industry Standard Benchmark for Big Data Analytics. In SIGMOD 2013. 1197–1208.