## [EXPEDIA GROUP TECHNOLOGY — DATA](https://medium.com/expedia-group-tech/data/home)

# **Apache Spark Structured Streaming — Watermarking (6 of 6)**

## Watermarking in Spark Streaming to handle late data

[IMG_256](https://medium.com/@bhadani.neeraj.08?source=post_page-----1187542d779f--------------------------------)

[IMG_257](https://medium.com/expedia-group-tech?source=post_page-----1187542d779f--------------------------------)

[Neeraj Bhadani](https://medium.com/@bhadani.neeraj.08?source=post_page-----1187542d779f--------------------------------)

·

[Follow](https://medium.com/m/signin?actionUrl=https://medium.com/_/subscribe/user/f9ed76fbd13a&operation=register&redirect=https://medium.com/expedia-group-tech/apache-spark-structured-streaming-watermarking-6-of-6-1187542d779f&user=Neeraj+Bhadani&userId=f9ed76fbd13a&source=post_page-f9ed76fbd13a----1187542d779f---------------------post_header-----------)

Published in

[Expedia Group Technology](https://medium.com/expedia-group-tech?source=post_page-----1187542d779f--------------------------------)

·

5 min read

·

Mar 11, 2021

94



Photo on unsplash.com by Dave Hoefler

In previous blogs, we discussed [input sources](https://medium.com/expedia-group-tech/apache-spark-structured-streaming-input-sources-2-of-6-6a72f798838c), [sinks](https://medium.com/expedia-group-tech/apache-spark-structured-streaming-output-sinks-3-of-6-ed3247545fbc), [checkpoints, triggers](https://medium.com/expedia-group-tech/apache-spark-structured-streaming-checkpoints-and-triggers-4-of-6-b6f15d5cfd8d) and [operations](https://medium.com/expedia-group-tech/apache-spark-structured-streaming-operations-5-of-6-40d907866fa7). In this post, we discuss watermarking in Apache Spark™️ Streaming.

You may also be interested in some of my other posts on Apache Spark.

[Apache Spark Structured Streaming — First Streaming Example](https://medium.com/expedia-group-tech/apache-spark-structured-streaming-first-streaming-example-1-of-6-e8f3219748ef)

[Apache Spark Structured Streaming — Input Sources](https://medium.com/expedia-group-tech/apache-spark-structured-streaming-input-sources-2-of-6-6a72f798838c)

[Apache Spark Structured Streaming — Output Sinks](https://medium.com/expedia-group-tech/apache-spark-structured-streaming-output-sinks-3-of-6-ed3247545fbc)

[Apache Spark Structured Streaming — Checkpoints and Triggers](https://medium.com/expedia-group-tech/apache-spark-structured-streaming-checkpoints-and-triggers-4-of-6-b6f15d5cfd8d)

[Apache Spark Structured Streaming — Operations](https://medium.com/expedia-group-tech/apache-spark-structured-streaming-operations-5-of-6-40d907866fa7)

[Deep Dive into Apache Spark DateTime Functions](https://medium.com/expedia-group-tech/deep-dive-into-apache-spark-datetime-functions-b66de737950a)

[Working with JSON in Apache Spark](https://medium.com/expedia-group-tech/working-with-json-in-apache-spark-1ecf553c2a8c)

[Deep Dive into Apache Spark Window Functions](https://medium.com/expedia-group-tech/deep-dive-into-apache-spark-window-functions-7b4e39ad3c86)

[Deep Dive into Apache Spark Array Functions](https://medium.com/expedia-group-tech/deep-dive-into-apache-spark-array-functions-720b8fbfa729)

[Start Your Journey with Apache Spark](https://medium.com/expedia-group-tech/start-your-journey-with-apache-spark-part-1-3575b20ee088)

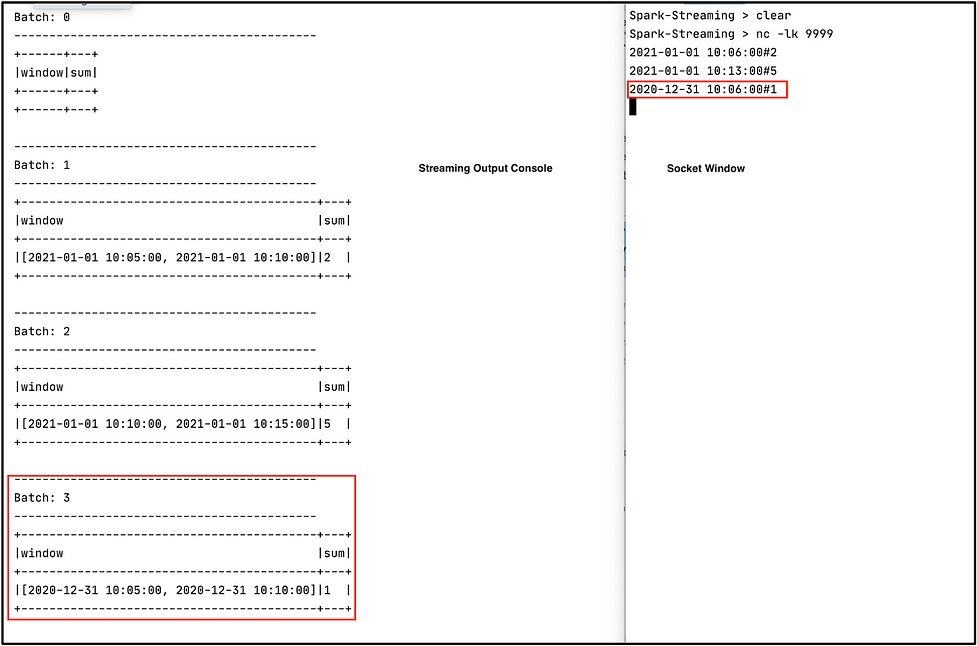
Let’s set-up a streaming application by reading data from a socket to understand the need and concept of watermarking with an example.

IMG_259

# **Setup**

We read data from a socket (127.0.0.1:9999) and pass along data which is # separated. Then we create a DataFrame with 2 columns event\_timestamp and val. More details on reading data from a socket can be found in [this](https://medium.com/expedia-group-tech/apache-spark-structured-streaming-input-sources-2-of-6-6a72f798838c) blog.

Now let’s open port 9999 and send some data, do the window aggregation and display the results on console. We kept the size of the tumbling window as 5 minutes. More details on window operations can be found in [this](https://medium.com/expedia-group-tech/apache-spark-structured-streaming-operations-5-of-6-40d907866fa7) blog.



On right window, we can see input on socket window. On left window, we can see output on console.

The first 2 records are from 1st Jan 2021 while the 3rd record is from the 31st Dec 2020. Let’s consider the third record *late data*. (Please be aware that the definition of late data varies from application to application). With window operations, a streaming application will process all data and put late data into an older window. At the same time, it maintains state for the old data which might not be required anymore. We can use watermarking to handle late data so that the system discards records automatically.

Now let’s add withWatermark() to our application and see how it works. We set watermark to delayThreshold = 10 minutes as well as windowDuration = 5 minutes using groupBy(). Watermark is set to max event time seen so far — delayThreshold. Let’s understand with an example.

Open the port 9999 , start our streaming application and send the same data again to the socket. Sample data can be found [here](https://github.com/NeerajBhadani/spark-streaming/blob/master/data/sampleWaterMarkData" \t "https://medium.com/expedia-group-tech/_blank). Let's discuss each record in detail.

****First record****: 2021–01–01 10:06:00#2 It will be processed since this is the first record.

Output:

-------------------------------------------  
Batch: 1  
-------------------------------------------  
+------------------------------------------+---+  
|window |sum|  
+------------------------------------------+---+  
|[2021-01-01 10:05:00, 2021-01-01 10:10:00]|2 |  
+------------------------------------------+---+

*Note: Batch:0 was empty.*

Since size of the window is 5 minutes we have a window value of [2021–01–01 10:05:00, 2021–01–01 10:10:00]

2. ****Second record**** : 2021–01–01 10:13:00#5

We have only processed one record so far, which means max event time = 2021–01–01 10:06:00. Hence, the watermark (max event time — delayThreshold) is 2021–01–01 09:56:00. For a window of size 5 minutes, the watermark window will be [09:55:00, 10:00:00]. So any data earlier than the start of the watermark window (2021–01–01 09:55:00) will be discarded. However, the event time (2021–01–01 10:13:00) for the second record is greater than the start time of the watermark window, so it will be processed.

*Note: in the watermark window, I only included time because date will always be 2021–01–01 in our dataset.*

-------------------------------------------  
Batch: 2  
-------------------------------------------  
+------------------------------------------+---+  
|window |sum|  
+------------------------------------------+---+  
|[2021-01-01 10:10:00, 2021-01-01 10:15:00]|5 |  
+------------------------------------------+---+

3. ****Third Record****: 2020–12–31 10:06:00#1

Our system’s current state is

max event time: [2021–01–01 10:13:00]

watermark: [2021–01–01 10:03:00]

watermark window: [2021–01–01 10:00:00, 2021–01–01 10:05:00]

For this event, the timestamp is less than the start of the watermark window, hence it will be treated as late data and discarded. We will get an empty batch.

-------------------------------------------  
Batch: 4  
-------------------------------------------  
+------+---+  
|window|sum|  
+------+---+  
+------+---+

4. ****Fourth Record****: 2021–01–01 10:08:00#8

max event time: [2021–01–01 10:13:00]

watermark: [2021–01–01 10:03:00]

watermark window: [10:00:00, 10:05:00]

This event’s time is greater than the start of the watermark window, hence it will be processed.

-------------------------------------------  
Batch: 5  
-------------------------------------------  
+------------------------------------------+---+  
|window |sum|  
+------------------------------------------+---+  
|[2021-01-01 10:05:00, 2021-01-01 10:10:00]|10 |  
+------------------------------------------+---+

5. ****Fifth Record****: 2021–01–01 10:00:01#7

max event time: [2021–01–01 10:13:00]

watermark: [2021–01–01 10:03:00]

watermark window: [10:00:00, 10:05:00]

This event’s time is slightly greater than the start of the watermark window, hence it will be processed.

6. ****Sixth Record****: 2021–01–01 9:59:00#3

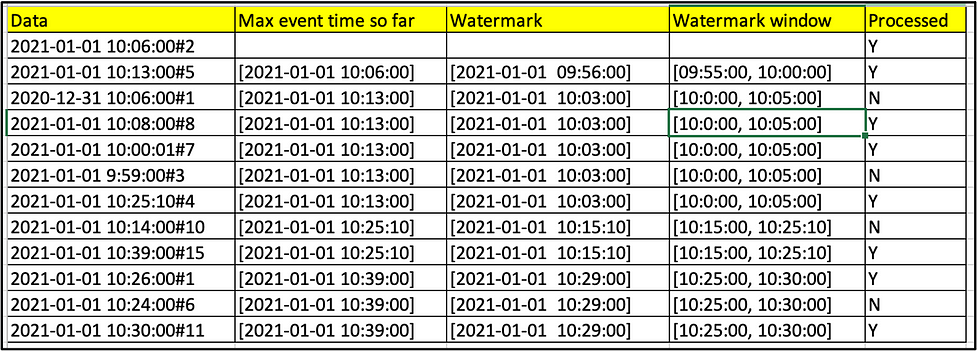
max event time: [2021–01–01 10:13:00]

watermark: [2021–01–01 10:03:00]

watermark window: [10:00:00, 10:05:00]

This event’s time is less than the start of the watermark window, hence it will not be processed.

Here is the consolidated list of all ingested events with max event seen so far, watermark, watermark window, and if they were processed or not.



An image that demonstrates watermarking with an example.

Ingest above data into our streaming application and verify the processed status. You can find the complete code on ***[GitHub](https://github.com/NeerajBhadani/spark-streaming/blob/master/src/main/scala/watermarkOperation.scala" \t "https://medium.com/expedia-group-tech/_blank)***.

I hope you have enjoyed learning about watermarking in Spark Streaming!

IMG_262

# **Reference**

[https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html#handling-late-data-and-watermarking](https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html" \l "handling-late-data-and-watermarking" \t "https://medium.com/expedia-group-tech/_blank)

[Programming](https://medium.com/tag/programming?source=post_page-----1187542d779f---------------programming-----------------)

[Software Engineering](https://medium.com/tag/software-engineering?source=post_page-----1187542d779f---------------software_engineering-----------------)

[Big Data](https://medium.com/tag/big-data?source=post_page-----1187542d779f---------------big_data-----------------)

[Spark](https://medium.com/tag/spark?source=post_page-----1187542d779f---------------spark-----------------)

[Streaming](https://medium.com/tag/streaming?source=post_page-----1187542d779f---------------streaming-----------------)