Watermarks

Measuring Time and Progress in Streaming Pipelines

Slava Chernyak  Google Inc
What is Cloud Dataflow?

Cloud Dataflow is a collection of SDKs for building parallelized data processing pipelines. Cloud Dataflow is a managed service for executing parallelized data processing pipelines.
Example: Auto-Completing Hash Tags

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Suggestions</th>
</tr>
</thead>
<tbody>
<tr>
<td>#ar</td>
<td>#argentina #aruglarocks, #argylesocks</td>
</tr>
<tr>
<td>#arg</td>
<td>#argentina #argylesocks #argonauts</td>
</tr>
<tr>
<td>#arge</td>
<td>#argentina #argentum #argentine</td>
</tr>
</tbody>
</table>
Tweets

Read

ExtractTags

Count

ExpandPrefixes

Top(3)

Write

Predictions

{#argentina scores!, watching #armenia vs #argentina, my #art project, …}

{argentina, armenia, argentina, art, …}

{argentina->5M, armenia->2M, art->90M, …}

{a->(argentina, 5M), a->(armenia, 2M), …, ar-> (argentina, 5M), ar-> (armenia, 2M), …}

{a->[apple, art, argentina], ar->[art, argentina, armenia], …}
Pipeline p = Pipeline.create(new PipelineOptions());

\[\text{p.begin()}\]

\[\text{.apply(TextIO.Read.from("gs://..."))}\]

\[\text{.apply(ParDo.of(new ExtractTags()))}\]

\[\text{.apply(Count.perElement())}\]

\[\text{.apply(ParDo.of(new ExpandPrefixes()))}\]

\[\text{.apply(Top.largestPerKey(3))}\]

\[\text{.apply(TextIO.Write.to("gs://..."))}\]

\[\text{.apply(TextIO.Read.from("gs://..."))}\]

\[\text{class ExpandPrefixes \{}\]

public void processElement(ProcessContext c) {
    String word = c.element().getKey();
    for (int i = 1; i <= word.length(); i++) {
        String prefix = word.substring(0, i);
        c.output(KV.of(prefix, c.element()));
    }
}

\[\text{\}}\]

p.run();
Pipeline p = Pipeline.create(new PipelineOptions());
  p.begin()
      .apply(TextIO.Read.from("gs://..."))
      .apply(ParDo.of(new ExtractTags()))
      .apply(Count.perElement())
      .apply(ParDo.of(new ExpandPrefixes()))
      .apply(Top.largestPerKey(3))
      .apply(TextIO.Write.to("gs://..."));
  p.run();
Let’s Stream It!

Age out old data

#ar*
rank

#argyle

#argentinagoal

#armeniarocks

game begins

armenia wins!

time
Pipeline p = Pipeline.create(new PipelineOptions());
    p.begin()
        .apply(PubsubIO.Read.topic("input_topic"))
        .apply(ParDo.of(new ExtractTags()))
        .apply(Count.perElement())
        .apply(ParDo.of(new ExpandPrefixes()))
        .apply(Top.largestPerKey(3))
        .apply(PubsubIO.Write.topic("output_topic"));
    p.run();
Pipeline p = Pipeline.create(new PipelineOptions());
    p.begin()
       .apply(PubsubIO.Read.topic("input_topic"))
       .apply(ParDo.of(new ExtractTags()))
       .apply(Count.perElement())
       .apply(ParDo.of(new ExpandPrefixes()))
       .apply(Top.largestPerKey(3))
       .apply(PubsubIO.Write.topic("output_topic"));
    p.run();
Pipeline p = Pipeline.create(new PipelineOptions());
p.begin()
   .apply(PubsubIO.Read.topic("input_topic"))
   .apply(ParDo.of(new ExtractTags()))
   .apply(Count.perElement())
   .apply(ParDo.of(new ExpandPrefixes()))
   .apply(Top.largestPerKey(3))
   .apply(PubsubIO.Write.topic("output_topic"));
p.run();
Pipeline p = Pipeline.create(new PipelineOptions());
  p.begin()
    .apply(PubsubIO.Read.topic("input_topic"))
    .apply(Window.into(SlidingWindows.of(
      Duration.standardMinutes(60))))
    .apply(ParDo.of(new ExtractTags()))
    .apply(Count.perElement())
    .apply(ParDo.of(new ExpandPrefixes()))
    .apply(Top.largestPerKey(3))
    .apply(PubsubIO.Write.topic("output_topic"));
  p.run();
That was easy, right?
Not So Easy - How To Window Data Into One Hour Windows?

- It's not enough to just aggregate data for one hour of wall time. How do we know if we saw all the data?
  - What if a source was delayed? Then we don’t have some of the data
  - What if we were reading from a backlog, at faster than realtime speed? Then we may have more data than we want.
Can we tell, by looking at the throughput of a pipeline, if it is keeping up with the input?

- Variability in input rate and processing delay make this hard.
- If even just one record is stuck, we can’t close a window.
Streaming Dataflow with Timestamped Data
Streaming Dataflow with Timestamped Data

Streaming Dataflow Pipeline

Recently Completed
In-Flight

Event Time

Input
Output
Streaming Dataflow with Timestamped Data

Event Time

Recently Completed

Recently Completed

In-Flight

Now (Real Time)

Input

Output

Streaming Dataflow Pipeline

Oldest Unprocessed Data

Oldest Unprocessed Data

Recently Completed

Recently Completed

In-Flight

In-Flight

Event Time

Event Time

Google Cloud Platform
What is a Watermark?

The *Low Watermark* of a streaming step is the timestamp of the oldest work not yet completed.

Why is this definition useful?
What is a Watermark?

The *Low Watermark* of a streaming step is the timestamp of the oldest work not yet completed.

Why is this definition useful?

- **Visibility**: If the step is stuck, or failing, the low watermark cannot advance. This lets us know that we are not making progress, and lets us identify the work that is causing stuckness.
- **Actionability**: A low watermark at time T guarantees that all (on-time) work before T has been finished. We can now aggregate.
What is a Watermark?

Watermarks can be defined at the edges of a streaming stage:

- **Input Low Watermark**: Oldest work not yet sent to this streaming stage.

- **Output Low Watermark**: Oldest work not yet completed by this streaming stage.
Watermarks At the Boundaries of Streaming Stages

- **Input Low Watermark**: Oldest work not yet sent to this streaming stage.
- **Output Low Watermark**: Oldest work not yet completed by this streaming stage.
What is a Watermark?

Watermarks are defined recursively upstream

\[
\text{OutputLowWatermark(Stage)} = \min \{ \text{InputLowWatermark(Stage)}, \text{OldestWork(Stage)} \}
\]

\[
\text{InputLowWatermark(Stage)} = \min \{ \text{OutputLowWatermark(Stage')}, \text{Stage' is upstream of Stage} \}
\]
Extending The Definition: Pipelines of Streaming Steps

Stage1: Filter
Stage2A: GroupBy KeyA
Stage2B: GroupBy KeyB
Stage3: Aggregate
Extending The Definition: Pipelines of Streaming Steps

Stage1: Filter
Stage2A: GroupBy KeyA
Stage2B: GroupBy KeyB
Stage3: Aggregate

What do we use for the watermark at the source?
Source Watermarks

- Data sources must provide a notion of the oldest work not yet sent to the streaming pipeline.
- We cannot (always) predict the future, sources may have old records that they don’t know about, etc.
- Such messages are labeled “late data”. Dataflow provides special semantics when handling late data.
Watermarks By Example

- Lets look at the tweets pipeline from the beginning of this talk to see how watermarks work.
- Assume that the source gives us perfect information about data not yet sent to the pipeline.
- Let's see how watermarks help us implement 1 hour windowing aggregations.
Watermarks Example

Stage 1: PubsubIO.Read

Stage 2: Window.into(Windows.of(60_Minutes))
Watermarks Example

**Stage 1:**
- **Input:** UNKNOWN
- **Process:** EMPTY
- **Output:** UNKNOWN

**Stage 2:**
- **Input:** UNKNOWN
- **Process:** EMPTY
- **Output:** UNKNOWN

**Pubsub:**
- **Input:** UNKNOWN
- **Output:** UNKNOWN

**Watermark Aggregator**
- **Pubsub Output:** UNKNOWN
- **Stage 1 Input:** UNKNOWN
- **Stage 1 Output:** UNKNOWN
- **Stage 2 Input:** UNKNOWN
- **Stage 2 Output:** UNKNOWN

**Stage 1:**
- **PubsubIO.Read**

**Stage 2:**
- **Window.into(Windows.of(60_Minutes))**
Stage 1:
Pubsub Output: UNKNOWN
Stage 1 Input: UNKNOWN
Stage 1 Output: UNKNOWN
Stage 2 Input: UNKNOWN
Stage 2 Output: UNKNOWN

Stage 2: Window.into(Windows.of(60_Minutes))
Watermarks Example

Watermark Aggregator
Pubsub Output: 1:00:00
Stage 1 Input: 1:00:00
Stage 1 Output: UNKNOWN
Stage 2 Input: UNKNOWN
Stage 2 Output: UNKNOWN

PubSub

Stage 1: PubsubIO.Read

Stage 2
Window.into(Windows.of(60_Minutes))

Stage 1:
Input: UNKNOWN
Process: EMPTY
Output: UNKNOWN

Stage 2:
Input: UNKNOWN
Process: EMPTY
Output: UNKNOWN
Watermarks Example

Watermark Aggregator
Pubsub Output: 1:00:00
Stage 1 Input: 1:00:00
Stage 1 Output: UNKNOWN
Stage 2 Input: UNKNOWN
Stage 2 Output: UNKNOWN

Stage 1: PubsubIO.Read
Input: 1:00:00
Process: EMPTY
Output: UNKNOWN

Stage 2
Window.into(Windows.of(60_Minutes))
Input: UNKNOWN
Process: EMPTY
Output: UNKNOWN
Watermarks Example

Watermark Aggregator
Pubsub Output: 1:00:00
Stage 1 Input: 1:00:00
Stage 1 Output: UNKNOWN
Stage 2 Input: UNKNOWN
Stage 2 Output: UNKNOWN

PubSub

Stage 1:
PubsubIO.Read

Stage 2
Window.into(Windows.of(60_Minutes))

Stage 1:
Input: 1:00:00
Process: EMPTY
Output: 1:00:00

Stage 2:
Input: UNKNOWN
Process: EMPTY
Output: UNKNOWN
Watermarks Example

Stage 1:
Input: 1:00:00
Process: EMPTY
Output: 1:00:00

Stage 1 Input: 1:00:00
Stage 1 Output: 1:00:00
Stage 2 Input: 1:00:00
Stage 2 Output: UNKNOWN

PubSub

Stage 2
Window.into(Windows.of((60_Minutes)))

Stage 1: PubsubIO.Read

Watermark Aggregator
Pubsub Output: 1:00:00
Stage 1 Input: 1:00:00
Stage 1 Output: 1:00:00
Stage 2 Input: 1:00:00
Stage 2 Output: UNKNOWN

PubSub: 1:00:00

Stage 1:
Input: 1:00:00
Process: EMPTY
Output: 1:00:00

Stage 2:
Input: UNKNOWN
Process: EMPTY
Output: UNKNOWN
**Watermarks Example**

**Stage 1:**
- **Input:** 1:00:00
- **Process:** EMPTY
- **Output:** 1:00:00
- **Pubsub:** 1:00:00

**Stage 2:**
- **Input:** 1:00:00
- **Process:** EMPTY
- **Output:** UNKNOWN

**Watermark Aggregator**
- **Pubsub Output:** 1:00:00
- **Stage 1 Input:** 1:00:00
- **Stage 1 Output:** 1:00:00
- **Stage 2 Input:** 1:00:00
- **Stage 2 Output:** UNKNOWN
Watermarks Example

Stage 1:
Input: 1:00:00
Process: EMPTY
Output: 1:00:00

Stage 2 Input: 1:00:00
Stage 2 Output: UNKNOWN

Stage 2:
Window.into(Windows.of(60_Minutes))

PubSub

Watermark Aggregator
Pubsub Output: 1:00:00
Stage 1 Input: 1:00:00
Stage 1 Output: 1:00:00
Stage 2 Input: 1:00:00
Stage 2 Output: UNKNOWN
Watermarks Example

Watermark Aggregator
- Pubsub Output: 1:00:00
- Stage 1 Input: 1:00:00
- Stage 1 Output: 1:00:00
- Stage 2 Input: 1:00:00
- Stage 2 Output: 1:00:00

Stage 1: PubsubIO.Read
- Input: 1:00:00
- Process: EMPTY
- Output: 1:00:00

Stage 2: Window.into(Windows.of(60_Minutes))
- Input: 1:00:00
- Process: EMPTY
- Output: 1:00:00
Stage 1:
Input: 1:00:00
Process: EMPTY
Output: 1:00:00

Stage 2
Window.into(Windows.of(60_Minutes))

Stage 1 Input: 1:00:00
Stage 1 Output: 1:00:00
Stage 2 Input: 1:00:00
Stage 2 Output: 1:00:00

Watermark Aggregator
Pubsub Output: 1:00:00
Stage 1 Input: 1:00:00
Stage 1 Output: 1:00:00
Stage 2 Input: 1:00:00
Stage 2 Output: 1:00:00

Message with timestamp 1:01:00 Arrives from PubSub

PubSub
Watermarks Example

**Stage 1:**
- **Input:** 1:00:00
- **Process:** EMPTY
- **Output:** 1:00:00

**Pubsub:**
- **Output:** 1:00:00

**Stage 2 Input:** 1:00:00

**Stage 2:**
- **Input:** 1:00:00
- **Process:** EMPTY
- **Output:** 1:00:00

**Message is read by Stage 1 and sent to Stage 2 for windowing**

**Watermark Aggregator**
- **Pubsub Output:** 1:00:00
- **Stage 1 Input:** 1:00:00
- **Stage 1 Output:** 1:00:00
- **Stage 2 Input:** 1:00:00
- **Stage 2 Output:** 1:00:00
Stage 1:
Input: 1:00:00
Process: EMPTY
Output: 1:00:00

Stage 2:
Input: 1:00:00
Process: 1:01:00
Output: 1:00:00

Watermark Aggregator
Pubsub Output: 1:00:00
Stage 1 Input: 1:00:00
Stage 1 Output: 1:00:00
Stage 2 Input: 1:00:00
Stage 2 Output: 1:00:00
Watermarks Example

**Watermark Aggregator**
- Pubsub Output: 1:02:00
- Stage 1 Input: 1:02:00
- Stage 1 Output: 1:02:00
- Stage 2 Input: 1:02:00
- Stage 2 Output: 1:01:00

**Stage 1:**
- PubsubIO.Read
- PubSub

**Stage 2**
- Window.into(Windows.of(60_Minutes))
- Stage 2 Input: 1:02:00
- Process: 1:01:00
- Output: 1:01:00
Stage 1:
Input: 1:59:00
Process: EMPTY
Output: 1:59:00

Pubsub: 1:59:00

Stage 2
Input: 1:59:00
Process: 1:01:00
Output: 1:01:00

Watermark Aggregator
Pubsub Output: 1:59:00
Stage 1 Input: 1:59:00
Stage 1 Output: 1:59:00
Stage 2 Input: 1:59:00
Stage 2 Output: 1:01:00

More messages arrive via PubSub and are processed...
Watermarks Example

**Stage 1:**
- **Input:** 2:00:00
- **Process:** EMPTY
- **Output:** 2:00:00

**Stage 2:**
- **Input:** 2:00:00
- **Process:** 1:01:00
- **Output:** 1:01:00

**Pubsub:**
- **Input:** 2:00:00

**Watermark Aggregator**
- **Pubsub Output:** 2:00:00
- **Stage 1 Input:** 2:00:00
- **Stage 1 Output:** 2:00:00
- **Stage 2 Input:** 2:00:00
- **Stage 2 Output:** 1:01:00
Watermarks Example

**Stage 1:**
- **Input:** 2:00:00
- **Process:** EMPTY
- **Output:** 2:00:00

**Stage 2 Input:** 2:00:00
**Stage 2 Output:** 1:01:00

**Pubsub:**
- **Pubsub Output:** 2:00:00

**Stage 1:**
- **PubsubIO.Read**

**Stage 2:**
- **Window.into(Windows.of(60_MINUTES))**

**Watermark Aggregator**
- **Pubsub Output:** 2:00:00
- **Stage 1 Input:** 2:00:00
- **Stage 1 Output:** 2:00:00
- **Stage 2 Input:** 2:00:00
- **Stage 2 Output:** 1:01:00

Watermark guarantees we have seen all messages before 2:00:00. Can emit window aggregation.
Watermarks Example

Watermark Aggregator
Pubsub Output: 2:00:00
Stage 1 Input: 2:00:00
Stage 1 Output: 2:00:00
Stage 2 Input: 2:00:00
Stage 2 Output: 1:01:00

Stage 1:
PubsubIO.Read
Input: 2:00:00
Process: EMPTY
Output: 2:00:00

Stage 2
Window.into(Windows.of(60_Minutes))
Input: 2:00:00
Process: EMPTY
Output: 2:00:00
Takeaways from Example:

- Any delay in a stage will cause a delay in the downstream stage’s watermarks. This is by design, it will ensure correct aggregations.
- This works well only if we have good watermark information for the data source.
- What about late data?
  - Watermark is monotonic - why? Otherwise we would need a way to roll back aggregations which may have side-effects
  - But we can’t predict the future - late data will happen
  - Answer: ensure monotonicity, detect late data, provide special semantics for how to handle it. (Triggers)
Watermarks And Windowing

- The SDK has a concept of windows. We saw an example of this with 60 minute windows earlier.

- Watermarks drive triggering these windows - when a watermark passes event time $T$, we trigger windows ending up to $T$.

- Lots of complex windowing and triggering concepts can be built on top of this - see Ken’s talk called “Triggers as Progressing Predicates” for much more details on these.
How Do We Establish Watermarks For Sources?

Let's look at two representative examples:

- Reading events from log files - data in each log file is (mostly) ordered, full set of files is (mostly) known.

- Reading events from an opaque source - events arrive in no particular order.
Watermark from Log Files

Low Watermark = 1:10
Watermark from Log Files - Refinements

- Events might be out of order - so we need to scan ahead, but can’t always emit results since we need skew control

- Files may not be completed, but instead grow very slowly. In this case they need to be dropped out of the estimate.

- Very old files may show up unexpectedly.
Watermark from PubSub

- PubSub is a message delivery system with no order guarantees. Data is published concurrently by many clients, and subscribed to concurrently by many shards of the streaming pipeline.

- PubSub does not know about application-level metadata such as event timestamps.
Watermark from PubSub

What we do know: oldest outstanding message publish timestamp. This is not the same as a watermark - why?
Watermark from PubSub

What we do know: oldest outstanding message publish timestamp. This is not the same as a watermark - why?

- Event time ≠ Publish Time, in fact they may be completely unrelated.
- If we read a message off disk that was a day old before publishing it to pubsub, then the pubsub bound is a day ahead of the actual message timestamp.
- Especially crucial when processing historical data sources. We want to window based on actual event time.
Watermark from PubSub

Base Subscription  Oldest Un-Ack’d (pt = 100)

<table>
<thead>
<tr>
<th>pt</th>
<th>et</th>
</tr>
</thead>
<tbody>
<tr>
<td>98</td>
<td>5</td>
</tr>
<tr>
<td>99</td>
<td>4</td>
</tr>
<tr>
<td>100</td>
<td>7</td>
</tr>
<tr>
<td>101</td>
<td>6</td>
</tr>
<tr>
<td>102</td>
<td>8</td>
</tr>
<tr>
<td>103</td>
<td>9</td>
</tr>
</tbody>
</table>

pt - publish time
et - event time
- ack’d message
- un-ack’d message
Watermark from PubSub

**Base Subscription**
- Oldest Un-Ack’d (pt = 100)
  - pt = 98, et = 5
  - pt = 99, et = 4
  - pt = 100, et = 7
  - pt = 101, et = 6
  - pt = 102, et = 8
  - pt = 103, et = 9

**Tracking Subscription**
- Oldest Un-Ack’d (pt = 104)
  - pt = 98, et = 5
  - pt = 99, et = 4
  - pt = 100, et = 7
  - pt = 101, et = 6
  - pt = 102, et = 8
  - pt = 103, et = 9

**Key Terms**
- **pt**: publish time
- **et**: event time
- **ack’d message**: a message that has been acknowledged
- **un-ack’d message**: a message that has not been acknowledged

**Legend**
- Black: ack’d message
- Red: un-ack’d message
Watermark from PubSub

Base Subscription

pt = 98  et = 5
pt = 99  et = 4
pt = 100 et = 7
pt = 101 et = 6
pt = 102 et = 8
pt = 103 et = 9

Tracking Subscription

pt = 98  et = 5
pt = 99  et = 4
pt = 100 et = 7
pt = 101 et = 6
pt = 102 et = 8
pt = 103 et = 9

Oldest Un-Ack’d (pt = 100)

Oldest Un-Ack’d (pt = 104)

Tracked Messages:

pt = 98  et = 5
pt = 99  et = 4
pt = 100 et = 7
pt = 101 et = 6
pt = 102 et = 8
pt = 103 et = 9

Estimate Min ET: 6

pt - publish time
et - event time
- ack’d message
- un-ack’d message
Watermarks allow us to reduce the complexity of tracking progress through a streaming pipeline to the complexity of tracking the completeness of the source.

For sources, the answer is highly specific to the source. No one size fits all answer. With greater information, we can improve the quality of the watermark.

We have found a number of approaches, heuristics, and tradeoffs that work well for common sources that streaming pipelines read from.
We’ve talked about watermarks based on event times of the data:
These can tell us about overall delay.

But what if we also compute watermarks based on the events the system is generating, based on the wall times of operations in the system?
These “system watermarks” allow us to distinguish data delay from system delay.
System Watermark Example

Consider the following graph of data watermark delay (Now - DataWatermark):

What is falling behind, the data or the system?
System Watermark Example

Two Cases Possible. Case 1: The system is stuck!
System Watermark Example

Two Cases Possible. Case 1: The system is stuck!
System Watermark Example

Two Cases Possible. Case 2: The data is stuck!
System Watermark Example

Two Cases Possible. Case 2: The data is stuck!

![Watermark Delay Graph]

- **Event Time**: Recently Completed
- **Streaming Dataflow Pipeline**: In-Flight
- **Data Watermark**: Blue line
- **System Watermark**: Red line
Watermarks are a way of defining progress and completeness semantics in streaming pipelines - allow us to know when we have “all” of the data.

We can define rich windowing semantics on top of watermarks, that allow for all sorts of complex aggregation patterns, without having to delve into the details of knowing when it is correct to emit aggregations.

Watermarks are only as accurate as the source is - the source has to tell us how old data is. In many cases we can estimate this pretty well.

System watermarks allow us to distinguish between data delays and system delays.
Questions ?