Deduplication: Where Beam Fits In
Beam Summit 2021

2021-08-04

Jeff Klukas
Sr. Staff Data Engineer, Firefox Data Platform
Who am I?

- Data engineer at Mozilla primarily working on the Firefox data pipeline
  - Deployed on Cloud Dataflow
  - 20 TB/day, 2 billion records/day
- Occasional poster on Beam mailing lists
- Author of several PRs in the Beam Java SDK, mostly in documentation and BigQueryIO
- Technical writing at https://jeff.klukas.net
- klukas on ASF Slack
- Email me: jeff@klukas.net
Objective

We'll compare the robustness, performance, and operational experience of deduplicating using built-in Beam transforms vs. storing IDs in an external Redis cluster and why Mozilla switched from one approach to the other in our streaming pipelines.

Mozilla’s pipeline is open source on GitHub: https://github.com/mozilla/gcp-ingestion
Agenda

1. Sources of Duplicate Messages
2. Beam’s Built-In Transforms for Deduplication
3. End-to-End Identifiers
4. Externalizing State
5. Comparison and Questions
Sources of Duplicate Messages
Built-In Transforms

The Java SDK provides two families of transforms relevant to this problem.

```java
PCollection<String> words = ...;
PCollection<String> deduplicatedWords = words.apply(Deduplicate.<String>values());

PCollection<String> words = ...;
PCollection<String> uniqueWords = words.apply(Distinct.<String>create());
```

“Distinct guarantees uniqueness of values within a PCollection but may support a narrower set of windowing strategies or may delay when output is produced” compared to Deduplicate.
Beam’s I/O machinery includes hooks for deduplication. For example, `PubsubIO.Read` calls `Deduplicate` under the hood to ensure each message is read only once.

The Dataflow runner even pushes PubsubIO to a separate service, so deduplication state does not consume worker resources.

This code snippet implicitly includes deduplication:
Readers and PubsubIO
Readers and PubsubIO

```java
public PCollection<PubsubMessage> expand(PBegin input) {
  return input //
      .apply(PubsubIO.readMessagesWithAttributesAndMessageId("".withIdAttribute(idAttribute)
      .fromSubscription(subscription))
```
End-to-End Identifier

The Firefox telemetry API requires that the client include a randomly generated UUID as part of the URL for each document sent.

We call this \textit{document\_id} and it serves as an \textit{end-to-end identifier} for the document.

\begin{itemize}
\item [/submit/\texttt{<namespace>}/<doctype>/\texttt{<version>}/\texttt{<document\_id>}]
\item [/submit/\texttt{eng\textunderscore workflow/\texttt{hgpush/1/2c3a0767-d84a-4d02-8a92-fa54a3376049}}]
\end{itemize}
End-to-End Identifier
Externalizing State

See Redis-based deduplication code in the gcp-ingestion repo
Batch Deduplication

Our “stable” tables for historical analysis in BigQuery are populated once per day, guaranteeing that each `document_id` is unique per table partition.

See full `copy_deduplicate` code in the mozilla/bigquery-etl repo

```sql
WITH
    numbered_duplicates AS (SELECT *,
                           ROW_NUMBER() OVER (PARTITION BY document_id
                                          ORDER BY submission_timestamp) AS _n
                          FROM live_table
                          WHERE DATE(submission_timestamp) = @submission_date)

SELECT *
  EXCEPT(_n)
FROM numbered_duplicates
WHERE _n = 1
```
<table>
<thead>
<tr>
<th></th>
<th>Distinct.java</th>
<th>Deduplicate.java</th>
<th>Dataflow PubsubIO</th>
<th>External State</th>
<th>Batch</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Time domain</strong></td>
<td>Event</td>
<td>Processing</td>
<td>Processing</td>
<td>Event</td>
<td>Event</td>
</tr>
<tr>
<td><strong>Duration</strong></td>
<td>Minutes</td>
<td>Minutes</td>
<td>10 min</td>
<td>Hours or days</td>
<td>Hours or days</td>
</tr>
<tr>
<td><strong>Built-in</strong></td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
<td>❌</td>
<td>✅</td>
</tr>
<tr>
<td><strong>No worker resource consumption</strong></td>
<td>❌</td>
<td>❌</td>
<td>✓</td>
<td>✓</td>
<td>❓</td>
</tr>
<tr>
<td><strong>Allows monitoring of duplicate rate</strong></td>
<td>❌</td>
<td>❌</td>
<td>❌</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Explore!

https://github.com/mozilla/gcp-ingestion
Thank you!

https://jeff.klukas.net

klukas on ASF Slack

jeff@klukas.net

https://github.com/mozilla/gcp-ingestion