Simplifying Speech-to-Text Processing with Apache Beam and Redis

Pramod Rao
Cloud Data Engineer
Google Cloud Consulting

Prateek Sheel
Data & Analytics Consultant
Google Cloud Consulting
Overview

Design Journey

Lessons Learned
01 Overview
Business Process

Users → Speech-to-text → Support Agent → Offer → Recommendation Engine

Speech-to-text

Support Agent

Offer

Recommendation Engine

Users

Pub/Sub → Dataflow → Vertex AI → Kafka
So, what’s the problem?

Events

First Agent Joins

Wait Time

Second Agent Joins

Warm Transfer

Overlap

First Agent Drops

Conversation

Second Agent Drops

User Joins

Multiple Call-Transfer Scenarios

Plus, additional business rules

Events

Key1

Key1|Key2

Transcript

Key2

Metadata

A B C D

DUPLICATES
OUT-OF-ORDER DATA
MISSING DATA

B B A C C

Key1|Key2
02

Design Journey
Design Approach #1

**Events**
- Key1

**Transcript**
- Key1|Key2

**Metadata**
- Key2

---

**Session Window**
- Conversation 1
- Conversation 2

**Re-key, and Window with key1 | key 2**

---

**Set 2 expiry timers**
- \( t_1 = t_0 + Xs \), \( t_2 = t_0 + Ys \)

**T1 expired**

**T2 expired**

---

**Expired**
- \( t_0 + 60s \) payload
- \( t_0 + 180s \) payload

---

**Conversation**
- Conversation 1
- Conversation 2
## Design 1 Trade Offs

<table>
<thead>
<tr>
<th>Dependencies</th>
<th>Latency</th>
<th>Completeness</th>
<th>Code Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>No state external to Dataflow. No external service dependencies.</td>
<td>Need to wait for the session to end and the timers to expire before the output payloads can be produced. Not ideal based on the business SLO.</td>
<td>In some cases all of the information required to creating the output payloads may not be available when the timers expire. This is due to the uncertain ordering of events.</td>
<td>Windowing allows for relatively simpler business logic implementation for creating the output payloads since re-keying produces outputs at the required granularity</td>
</tr>
</tbody>
</table>
Design Approach # 2

Events
- Key1

Transcript
- Key1|Key2

Metadata
- Key2

Session Window

Key1

Conversation 1

- Events
- Transcripts
- Metadata

Set 2 expiry timers: $t_1 = t_0 + X_s$, $t_2 = t_0 + Y_s$

T1 expired

T2 expired

$\text{t0 + 60s payload}$

$\text{t0 + 180s payload}$

$\text{t0 + Xs output}$

$\text{t0 + Ys output}$
<table>
<thead>
<tr>
<th>Dependencies</th>
<th>Latency</th>
<th>Completeness</th>
<th>Code Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>No state external to Dataflow. No external service dependencies.</td>
<td>Need to wait for the session to end and the timers to expire before the output payloads can be produced. Not ideal based on the business SLO.</td>
<td>In some cases all of the information required to creating the output payloads may not be available when the timers expire. This is due to the uncertain ordering of events.</td>
<td><strong>Granularity</strong> of outputs doesn’t match the inputs thereby increasing the business logic <strong>complexity</strong> required to produce the output payloads.</td>
</tr>
</tbody>
</table>
Design Approach # 3

Events
Key1

Transcript
Key1|Key2

Metadata
Key2

Dataflow

Session Window

Set 2 expiry timers \( t_1 = t_0 + X_s \), \( t_2 = t_0 + Y_s \)

T1 expired

T2 expired

Events

Transcripts

Metadata

Conversations 1

Conversations 2

Dataflow to Redis

Lettuce 6.1.8

Sorted Sets

Set expirations:
- \( t_0 + 60s \) payload
- \( t_0 + 180s \) payload
We rely on Redis sorted sets for accumulating the speech transcripts, we are able to maintain the order of the conversation as well as deduplicating the transcripts automagically.

Redis offers a simple approach to manage cleanup of stale data.

Low latency data store that dovetails well with streaming use cases.
## Design 3 Trade Offs

<table>
<thead>
<tr>
<th>Dependencies</th>
<th>Latency</th>
<th>Completeness</th>
<th>Code Complexity</th>
</tr>
</thead>
</table>
| **Dependency** on a managed Redis instance. This also results in additional **costs** to host a Redis instance in the Cloud environment. | No need for any additional wait time over and above the required timers.  
**Subsecond end-to-end latency for ML predictions.** | Least chance of incomplete outputs due to the **ordering** provided by Redis | Much **simpler processing** because complicated scenarios related to cross-referencing the three data sources are eliminated. Only need to “act” on events. |
## Latency Metrics

<table>
<thead>
<tr>
<th>Dataflow</th>
<th>Machine Type</th>
<th>PreProcessing</th>
<th>Redis</th>
<th>Predictions</th>
<th>End-To-End</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>n1-standard-2</td>
<td>1210.90</td>
<td>20.84</td>
<td>204.83</td>
<td>1441.75</td>
</tr>
<tr>
<td></td>
<td>t0+60s</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>n1-standard-2</td>
<td>1155.52</td>
<td>18.62</td>
<td>260.33</td>
<td>1441.72</td>
</tr>
<tr>
<td></td>
<td>t0+180s</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>n2d-standard-4</td>
<td>580.38</td>
<td>9.84</td>
<td>198.68</td>
<td>796.10</td>
</tr>
<tr>
<td></td>
<td>t0+60s</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>n2d-standard-4</td>
<td>596.54</td>
<td>9.98</td>
<td>260.54</td>
<td>874.35</td>
</tr>
<tr>
<td></td>
<td>t0+180s</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Speech-to-text Processing with Apache Beam and Redis

Final Solution
03

Lessons Learned
Lessons Learned

Order of data
Real world scenarios include out-of-order data, duplicates, and missing elements

Granularity of inputs
Business logic is greatly simplified if all inputs are at the same level of "granularity"

Representative test data
"Good" test data is imperative to shorten the development lifecycle and can be tricky to generate or acquire

Observability
Non functional requirements such as operational metrics and dead-letter queues are essential to gain insights into the processing state at any time

Configurability
Levers should be provided to change the processing characteristics without changing any code

Latency
Latency requirements dictate the nature of the final solution
Thank you!

https://cloud.google.com/consulting