Scaling machine learning to millions of users with Apache Beam

Tatiana Al-Chueyr
Principal Data Engineer @ BBC Datalab

Online, 4 August 2021
@tati_alchueyr

- **Brazilian** living in London **UK** since 2014
- Principal Data **Engineer** at the **BBC** (Datalab team)
- Graduated in **Computer Engineering** at Unicamp
- Software developer for **18 years**
- Passionate about **open-source**

**Apache Beam user** since **early 2019**
The knowledge in this presentation is the result of lots of teamwork within one squad of a larger team and even broader organisation.

Current squad team members:
- Darren Mundy
- Richard Bownes
- Tatiana Al-Chueyr

Previous squad team members:
- Bettina Hermant
- David Hollands
- Jana Eggink
- Marc Oppenheimer
some business context
business context goal

to build a **replacement** for an **external third-party** recommendation engine

to **personalise** the experience of **millions of users** of BBC Sounds
BBC Sounds has approximately

- **200,000** podcast and music episodes
- **6.5 millions** of users

The personalised rails (eg. Recommended for You) display:

- **9 episodes** (smartphones) or
- **12 episodes** (web)
business context problem visualisation

it is similar to finding the best match among 20,000 items per user x 65 million times
The **recommendations** must also comply to the BBC product and editorial **rules**, such as:

- **Diversification**: no more than one item per brand
- **Recency**: no news episodes older than 24 hours
- **Narrative arc**: next drama series episode
- **Language**: Gaelic items to Gaelic listeners
- **Availability**: only available content
- **Exclusion**: shipping forecast and soap-opera
technology & architecture overview
technology overview

- Python
- Google Cloud Platform
- Apache Airflow
- Apache Beam (Dataflow Runner)
- LightFM Factorisation Machine model
architecture overview

User activity
Extract & Transform
User activity features
Train
Model Artefacts
Predict
Predictions
Apply rules
Filtered Predictions

Content metadata
Extract & Transform
Content metadata features

historical data

future
risk analysis predict on the fly

A. On the fly
- API predicts & applies rules
  - user activity
  - model
  - content metadata

B. Precompute
- API retrieves pre-computed recommendations
  - cached recs

SLA goal
1500 reqs/s < 60 ms
## Risk Analysis Predict on the Fly

<table>
<thead>
<tr>
<th></th>
<th>On the fly</th>
<th>Precomputed</th>
<th>Precomputed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concurrent load tests requests/s</td>
<td>50</td>
<td>50</td>
<td>1500</td>
</tr>
<tr>
<td>Success percentage</td>
<td>63.88%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>Latency of p50 (success)</td>
<td>323.78 ms</td>
<td>1.68 ms</td>
<td>4.75 ms</td>
</tr>
<tr>
<td>Latency of p95 (success)</td>
<td>939.28 ms</td>
<td>3.21 ms</td>
<td>57.53 ms</td>
</tr>
<tr>
<td>Latency of p99 (success)</td>
<td>979.24 ms</td>
<td>4.51 ms</td>
<td>97.49 ms</td>
</tr>
<tr>
<td>Maximum successful requests per second</td>
<td>23</td>
<td>50</td>
<td>1500</td>
</tr>
</tbody>
</table>

A. On the fly

B. Precompute

SLA goal
1500 reqs/s
< 60 ms
**risk analysis** precompute recommendations

Estimate of time (seconds) to precompute recommendations

<table>
<thead>
<tr>
<th>Number of threads</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>4</th>
<th>4</th>
<th>4</th>
<th>8</th>
<th>8</th>
<th>8</th>
<th>16</th>
<th>16</th>
<th>16</th>
<th>30</th>
<th>30</th>
<th>30</th>
</tr>
</thead>
<tbody>
<tr>
<td>Candidate items \ Chunk of users</td>
<td>10k</td>
<td>60k</td>
<td>100k</td>
<td>10k</td>
<td>60k</td>
<td>100k</td>
<td>10k</td>
<td>60k</td>
<td>100k</td>
<td>10k</td>
<td>60k</td>
<td>100k</td>
<td>10k</td>
<td>60k</td>
<td>100k</td>
</tr>
<tr>
<td>1 user</td>
<td>0.01</td>
<td>0.05</td>
<td>0.05</td>
<td>0.01</td>
<td>0.01</td>
<td>0.02</td>
<td>0.02</td>
<td>0.02</td>
<td>0.03</td>
<td>0.01</td>
<td>0.02</td>
<td>0.03</td>
<td>0.01</td>
<td>0.01</td>
<td>0.03</td>
</tr>
<tr>
<td>100 users</td>
<td>0.32</td>
<td>2.45</td>
<td>4.21</td>
<td>0.16</td>
<td>1.11</td>
<td>1.86</td>
<td>0.14</td>
<td>0.86</td>
<td>1.45</td>
<td>0.12</td>
<td>0.76</td>
<td>1.28</td>
<td>0.11</td>
<td>0.73</td>
<td>1.20</td>
</tr>
<tr>
<td>1000 users</td>
<td>3.09</td>
<td>24.10</td>
<td>39.95</td>
<td>1.44</td>
<td>10.67</td>
<td>18.31</td>
<td>1.17</td>
<td>8.47</td>
<td>14.22</td>
<td>1.04</td>
<td>7.25</td>
<td>12.53</td>
<td>1.00</td>
<td>7.02</td>
<td>11.75</td>
</tr>
<tr>
<td>10000 users</td>
<td>31.05</td>
<td>231.13</td>
<td>409.41</td>
<td>14.24</td>
<td>109.31</td>
<td>184.96</td>
<td>11.40</td>
<td>84.50</td>
<td>141.85</td>
<td>9.99</td>
<td>73.89</td>
<td>125.3</td>
<td>9.75</td>
<td>69.73</td>
<td>116.58</td>
</tr>
</tbody>
</table>

analysis using c2-standard-30 (30 vCPU and 120 RAM) and LightFM

**cost estimate:** ~ US$ 10.00 run
risk **analysis** sorting recommendations

<table>
<thead>
<tr>
<th>Chunk of users \ Candidate items</th>
<th>10k</th>
<th>60k</th>
<th>100k</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 user</td>
<td>0.14</td>
<td>0.10</td>
<td>0.17</td>
</tr>
<tr>
<td>100 users</td>
<td>1.39</td>
<td>9.68</td>
<td>17.08</td>
</tr>
<tr>
<td>1000 users</td>
<td>16.07</td>
<td>105.35</td>
<td>180.60</td>
</tr>
<tr>
<td>10000 users</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

sort 100k predictions per user with pure Python did not seem efficient
architecture overview

User activity
Extract & Transform
User activity features
Train
Model Artefacts
Predict
Predictions
Apply rules
Filtered Predictions

Content metadata
Extract & Transform
Content metadata features

historical data

future
where we used Apache Beam
architecture overview

User activity data → Content metadata

Machine Learning model training → Predict recommendations

Business Rules, part I - Non-personalised
- Recency
- Availability
- Excluded Masterbrands
- Excluded genres

Business Rules, part II - Personalised
- Already seen items
- Local radio (if not consumed previously)
- Specific language (if not consumed previously)
- Episode picking from a series
- Diversification (1 episode per brand/series)

Precomputed recommendations

Machine Learning model training

Predict recommendations

Precomputed recommendations
precompute recommendations

pipeline evolution
pipeline 1.0 design & arguments

apache-beam[gcp]==2.15.0
--runner=DataflowRunner
--machine-type = n1-standard-1 (1 vCPU & 3.75 GB RAM)
--num_workers=10
--autoscaling_algorithm=NONE
pipeline 1.0 design

August 2020
pipeline 1.0 design
pipeline 1.0 error when running in dev & prod

Workflow failed. Causes: S05:Read non-cold start users/Read+Retrieve user ids+Predict+Keep best scores+Sort scores+Process predictions+Group activity history and recommendations/pair_with_recommendations+Group activity history and recommendations/GroupByKey/Reify+Group activity history and recommendations/GroupByKey/Write failed. The job failed because a work item has failed 4 times. Look in previous log entries for the cause of each one of the 4 failures. For more information, see https://cloud.google.com/dataflow/docs/guides/common-errors. The work item was attempted on these workers:

- beamapp-al-cht01-08141052-08140353-1tqj-harness-0k4v
  Root cause: The worker lost contact with the service.
- beamapp-al-cht01-08141052-08140353-1tqj-harness-ffqv
  Root cause: The worker lost contact with the service.
- beamapp-al-cht01-08141052-08140353-1tqj-harness-cjht
  Root cause: The worker lost contact with the service.
# pipeline 1.0 data analysis

<table>
<thead>
<tr>
<th>Data structure</th>
<th>(dev) size in disk</th>
<th>(prod) size in disk</th>
<th>(dev) size in memory</th>
<th>(prod) size in memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>model</td>
<td>1.74 GiB</td>
<td>3.99 GiB</td>
<td>2.49 GB</td>
<td>4.50 GB</td>
</tr>
<tr>
<td>candidate items</td>
<td>22.58 MiB</td>
<td>23.95 MiB</td>
<td>0.19 GB</td>
<td>0.20 GB</td>
</tr>
<tr>
<td>item features</td>
<td>832.11 KiB</td>
<td>1.26 MiB</td>
<td>0.04 GB</td>
<td>0.05 GB</td>
</tr>
<tr>
<td>mapping</td>
<td>357 MiB</td>
<td>404 MiB</td>
<td>1.30 GB</td>
<td>2.79 GB</td>
</tr>
<tr>
<td>consumed items</td>
<td>20.04 MiB</td>
<td>48.16 MiB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>consumption history</td>
<td>299 MiB</td>
<td>1.45 GiB</td>
<td>4.02 GB</td>
<td>7.54 GB</td>
</tr>
</tbody>
</table>
pipeline 1.0 attempts to fix (i)

1. Change machine type to a larger one
   - --machine_type=custom-1-6656 (1 vCPU, 6.5 GB RM) - 6.5GB RAM /core
   - --machine_type=m1-ultramem-40 (40 vCPU, 961 GB RAM) - 24GB RAM/core

2. Refactor the pipeline

3. Reshuffle => too expensive for the operation we were doing
   - Shuffle service
   - Reshuffle function

4. Increase the amount of workers
   - --num_workers=40

September 2020
5. Control the parallelism in Dataflow so the VM wouldn’t starve out of memory

--number_of_worker_harness_threads=1
--experiments=use_runner_v2
(or)
--sdk_worker_parallelism
--experiments=no_use_multiple_sdk_containers
--experiments=beam_fn_api

pipeline 1.0 attempts to fix (ii)
pipeline 1.0 attempts to fix (iii)

https://stackoverflow.com/questions/63705660/optimising-gcp-costs-for-a-memory-intensive-dataflow-pipeline
pipeline 1.0 attempts to fix (iii)

We've been struggling to reduce the costs of an @ApacheBeam pipeline in @googlecloud Dataflow: stackoverflow.com/questions/6370... Any ideas @datancoffee @rarokni?

Optimising GCP costs for a memory-intensive Dataflow Pi... We want to improve the costs of running a specific Apache Beam pipeline (Python SDK) in GCP Dataflow. We have bu... stackoverflow.com

https://twitter.com/tati_alchueyr/status/1301152715498758146
pipeline 1.0 attempts to fix (iii)

We are working on long-term solutions to these problems, but here is a tactical fix that should prevent the model duplication that you saw in approaches 1 and 2:

Share the model in a VM across workers, to avoid it being duplicated in each worker. Use the following utility

([https://github.com/apache/beam/blob/master/sdks/python/apache_beam/impls/shared.py](https://github.com/apache/beam/blob/master/sdks/python/apache_beam/impls/shared.py)), which is available out of the box in Beam 2.24 If you are using an earlier version of Beam, copy just the shared.py to your project and use it as user code.

answered Sep 9 '20 at 0:21

[Sergei](https://stackoverflow.com/users/21175367/sergei)

[201](https://stackoverflow.com/users/21175367/sergei) • 1 • 4

https://stackoverflow.com/questions/63705660/optimising-gcp-costs-for-a-memory-intensive-dataflow-pipeline
pipeline 1.0 attempts to fix (iii)

I don't think that at this moment there's an option to control the number of executors per VM, it seems that the closest that you will get there is by using the option (1) and assume a Python executor per core.

Option (1)

```bash
--number_of_worker_harness_threads=1 --experiments=use_runner_v2
```

To compensate on the cpu-mem ratio you need, I'd suggest using custom machines with extended memory. This approach should be more cost-effective.

For example, the cost of a running a single executor and a single thread on a `n1-standard-4` machine (4 CPUs - 15GB) will be roughly around 30% more expensive than running the same workload using a `custom-1-15360-ext` (1 CPU - 15GB) custom machine.

https://stackoverflow.com/questions/63705660/optimising-gcp-costs-for-a-memory-intensive-dataflow-pipeline
pipeline 2.0 design & arguments

apache-beam== 2.24
--runner=DataflowRunner
--machine-type = custom-30-460800-ext
--num_workers= 40
--autoscaling_algorithm=NONE
pipeline 2.0 business outcomes

- **+59% increase** in interactions in Recommended for You rail
- **+103% increase** in interactions for under 35s
pipeline 2.0 **issues**

- but **costs were high...**

<table>
<thead>
<tr>
<th>Resource metrics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current vCPUs</td>
</tr>
<tr>
<td>Total vCPU time</td>
</tr>
<tr>
<td>Current memory</td>
</tr>
<tr>
<td>Total memory time</td>
</tr>
<tr>
<td>Current HDD PD</td>
</tr>
<tr>
<td>Total HDD PD time</td>
</tr>
<tr>
<td>Current SSD PD</td>
</tr>
<tr>
<td>Total SSD PD time</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Elapsed time</th>
<th>2 hr 42 min</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encryption type</td>
<td>Google-managed key</td>
</tr>
</tbody>
</table>

£ **279.31** per run

September 2020
pipeline 2.0 issues

OSError: [Errno 28] No space left on device During handling

<table>
<thead>
<tr>
<th></th>
<th>successful</th>
<th>unsuccessful</th>
</tr>
</thead>
<tbody>
<tr>
<td>compute-predictions-21628eba</td>
<td>2021-02-24_20_06_12-17726452853028084617 25 February 2021</td>
<td>2021-02-28_20_01_56-16233421279575625541 1st March 2021</td>
</tr>
<tr>
<td>consumption-history/</td>
<td>1.87 GiB</td>
<td>1.88 GiB</td>
</tr>
<tr>
<td>consumed-items/</td>
<td>55.63 MiB</td>
<td>55.68 MiB</td>
</tr>
<tr>
<td>candidate-items/rfy/</td>
<td>25.06 MiB</td>
<td>25.09 MiB</td>
</tr>
<tr>
<td>xantus.pkl</td>
<td>4.67 GiB</td>
<td>4.69 GiB</td>
</tr>
<tr>
<td>item_features.npz</td>
<td>1.73 MiB</td>
<td>1.73 MiB</td>
</tr>
<tr>
<td>mapping.json</td>
<td>473.84 MiB</td>
<td>476.01 MiB</td>
</tr>
</tbody>
</table>

March 2021
If a batch job uses Dataflow Shuffle, then the default is 25 GB; otherwise, the default is 250 GB.
pipeline 2.0 issues

```
apache-beam== 2.24
--runner=DataflowRunner
--machine-type = custom-30-460800-ext
--num_workers= 40
--autoscaling_algorithm=NONE
--experiments=shuffle_mode=appliance
```
<table>
<thead>
<tr>
<th>1. Administer pain relief</th>
<th>2. Hook up to bypass</th>
<th>3. Heart surgery</th>
</tr>
</thead>
<tbody>
<tr>
<td>➔ Attempt shared memory</td>
<td>➔ Mid week delta (only compute mid week for users with activity since Sunday’s run)</td>
<td>➔ Split pipeline</td>
</tr>
<tr>
<td>➔ Attempt FlexRS</td>
<td></td>
<td>➔ Major refactor</td>
</tr>
<tr>
<td></td>
<td></td>
<td>➔ SCANN vs LightFM.score()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>➔ etc.</td>
</tr>
</tbody>
</table>

Timebox: 1 week

Timebox: 2 weeks

Timebox: 1 month

April 2021
pipeline 3.0 design

apache-beam== 2.24
--runner=DataflowRunner
--machine-type = custom-30-460800-ext
--num_workers= 40
--autoscaling_algorithm=NONE
--experiments=shuffle_mode=appliance

April 2021
pipeline 3.0 shared memory & FlexRS strategy

- Used production-representative data (model, auxiliary data structures)
- Ran the pipeline for 0.5% users, so the iterations would be cheap
  - 100% users: £ 266.74
  - 0.5% users: £ 80.54
- Attempts
  - Shared model using custom-30-460800-ext (15 GB/vCPU)
  - Shared model using custom-30-299520-ext (9.75 GB/vCPU)
  - Shared model using custom-6-50688-ext (8.25 GB/vCPU)
    - 0.5% users: £ 18.46 => -77.5% cost reduction!
pipeline 3.0 shared memory & FlexRS results

- However, when we tried to run the same pipeline for 100%, it would take hours and not complete.
- It was very inefficient and costed more than the initial implementation.
pipeline 4.0 heart surgery

- Split compute predictions from applying rules
- Keep the interfaces to a minimal
  - between these two pipelines
  - between steps within the same pipeline
pipeline 4.1 precompute recommendations

apache-beam== 2.29
--runner=DataflowRunner
--machine-type = n1-highmem-16
--flexrs-goal = COST_OPTIMIZED
--max-num-workers= 64
--number-of-worker-harness-threads=7
--experiments=use_runner_v2

+ Batching
+ Shared memory

pipeline 4.1 precompute recommendations

Cost to run for 3.5 million users:
- 100k episodes: £48.92 / run
- 300 episodes: £3.40
- 18 episodes: £0.74

July 2021
pipeline 4.2 apply business rules

- Implemented rules natively
- Created minimal interfaces and views of the data

```
apache-beam==2.29
--runner=DataflowRunner
--machine-type=n1-standard-1
--experiments=use_runner_v2
```
pipeline 4.2 apply business rules

Cost to run for 3.5 million users:
- £0.15 - 0.83 per run

July 2021
pipeline 4.0 heart surgery

- We were able to reduce the cost of the most expensive run of the pipeline from **£ 279.31** per run to less than **£ 50**
- Reduced the costs to -82%

July 2021
takeaways
1. plan based on your **data**
2. an **expensive** machine learning pipeline is better than none
3. reducing the **scope** is a good starting point to **saving money**
   ○ Apply non-personalised rules before iterating per user
   ○ Sort top 1k recommendations by user opposed to 100k
4. using **custom machine types** might limit other **cost savings**
   ○ Such as FlexRS (schedulable preemptible instances in Dataflow only work)
5. to use **shared memory** may not lead to cost savings
6. **minimal interfaces** lead to more **predictable behaviours** in Dataflow
7. **splitting the pipeline** can be a **solution to costs**
Thank you!

@tati_alchueyr