Caching in Dataflow using Beam SDK

By Zeeshan Khan | Google

BEAM SUMMIT Austin, 2022

What is a cache

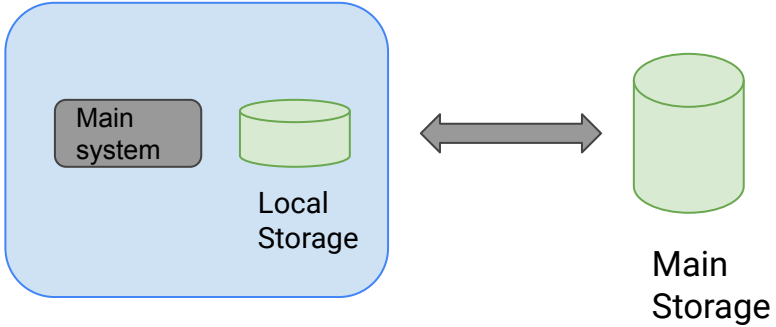
“software component that stores data so that future requests for that data can be served faster; the data stored in a cache might be the result of an earlier computation or a copy of data stored elsewhere” - Wikipedia

Why have a cache ?

- Calls to external systems are expensive
- Lower latency
- Better throughput

Trade offs :

- Stale data



Options in Beam SDK

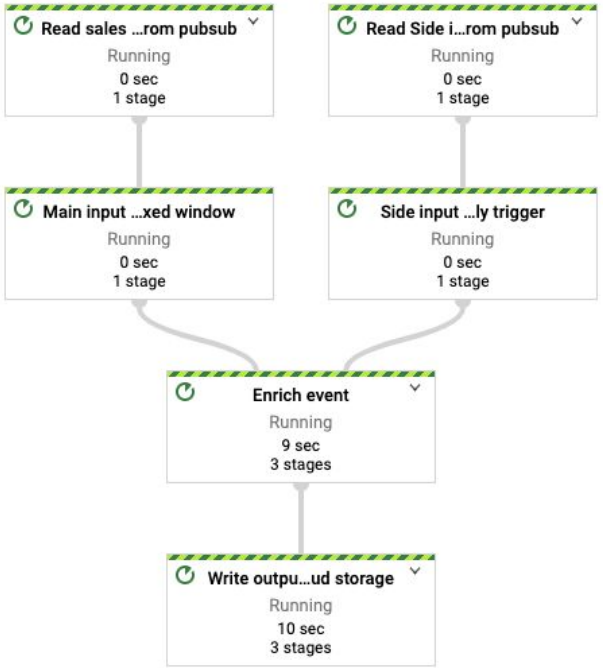
In memory caches
- Side Inputs
- Shared class

Stateful DoFn

In memory cache : Side Input

“A side input is an additional input that your DoFn can access each time it processes an element in the input PCollection”

This cache is a copy of data store elsewhere for example BigQuery, text files on GCS etc.



In memory cache : Side Input

```
main_input = (
    pipeline
    | 'MpImpulse' >> beam.Create(sample_main_input_elements)
    | 'MapMpToTimestamped' >> beam.Map(lambda src: TimestampedValue(src, src))
    | 'WindowMpInto' >> beam.WindowInto(window.FixedWindows(main_input_windowing_interval)))
```

In memory cache : Side Input

```
input = (
    pipeline
    | 'MpImpulse' >> beam.Create(sample_main_input_elements)
    | 'MapMpToTimestamped' >> beam.Map(lambda src: TimestampedValue(src, src))
    | 'WindowMpInto' >> beam.WindowInto(window.FixedWindows(main_input_windowing_interval)))

output = (
    input
    | 'ApplySomeFunction' >> beam.FlatMap(some_function))
```

In memory cache : Side Input

```
main_input = (
    pipeline
    | 'MpImpulse' >> beam.Create(sample_main_input_elements)
    | 'MapMpToTimestamped' >> beam.Map(lambda src: TimestampedValue(src, src))
    | 'WindowMpInto' >> beam.WindowInto(window.FixedWindows(main_input_windowing_interval)))

side_input = (pipeline
    | 'PeriodicImpulse' >> PeriodicImpulse(  first_timestamp, last_timestamp, interval, True)
    | 'MapToFileName' >> beam.Map(lambda x: src_file_pattern + str(x))
    | 'ReadFromFile' >> beam.io.ReadAllFromText())
```

In memory cache : Side Input

```
main_input = (
    pipeline
    | 'MpImpulse' >> beam.Create(sample_main_input_elements)
    | 'MapMpToTimestamped' >> beam.Map(lambda src: TimestampedValue(src, src))
    | 'WindowMpInto' >> beam.WindowInto(window.FixedWindows(main_input_windowing_interval)))

side_input = (pipeline
    | 'PeriodicImpulse' >> PeriodicImpulse(  first_timestamp, last_timestamp, interval, True)
    | 'MapToFileName' >> beam.Map(lambda x: src_file_pattern + str(x))
    | 'ReadFromFile' >> beam.io.ReadAllFromText())

output = (
    main_input
    | 'ApplySomeFunction' >> beam.FlatMap(some_function, rights=beam.pvalue.AsIter(side_input))))
```

In memory cache : Side Input

“A side input is an additional input that your DoFn can access each time it processes an element in the input PCollection”

Can be passed as :

- Singleton
- List
- Iterable
- Map
- Multimap

Technical consideration on Dataflow

- Streaming jobs without Streaming Engine store side input in memory. For Java pipelines, there is 1 copy per worker, while for Python there is 1 copy per vCPU.
- Streaming jobs using Streaming Engine have a limit of 80MB as max size of side input
- For best performance, side inputs should be small (less than 1GB)

Beam's Shared Class

`apache_beam.utils.shared.Shared`

Shared class

- Provides a way to sharing in-memory data object across multiple threads/DoFn with a process to improve space and access efficiency.

DoFn lifecycle

- **setup** — Invoked after creation of an instance.
- **Repeatedly process bundles:**
  - **start_bundle** — Invoked before processing of each bundle.
  - **Repeatedly process elements**
    - **process** method
  - **finish_bundle** — Invoked after processing of each bundle
- **teardown** — Invoked before instance is discarded and used for any clean up

Shared class

- There can be hundreds of instances on Dofn on a Dataflow worker

	Batch job	Streaming job	
		Without SE	With SE
Java	1 DoFn/vCPU	300 DoFn/vCPU	500 DoFn/vCPU
Python	1 DoFn/vCPU	12 DoFn/vCPU	12 DoFn/vCPU

Shared class

- Beam provides the Shared class (apache_beam.utils.shared.Shared) to share object across threads. One object per class

```
class GetNthStringFn(beam.DoFn):
  def __init__(self, shared_handle):
    self._shared_handle = shared_handle

  def process(self, element):
    def initialize_list():
      # Build the giant initial list.
      return [str(i) for i in range(1000000)]

    giant_list = self._shared_handle.acquire(initialize_list)
    yield giant_list[element]

p = beam.Pipeline()
shared_handle = shared.Shared()
(p | beam.Create([2, 4, 6, 8])
     | beam.ParDo(GetNthStringFn(shared_handle)))
```

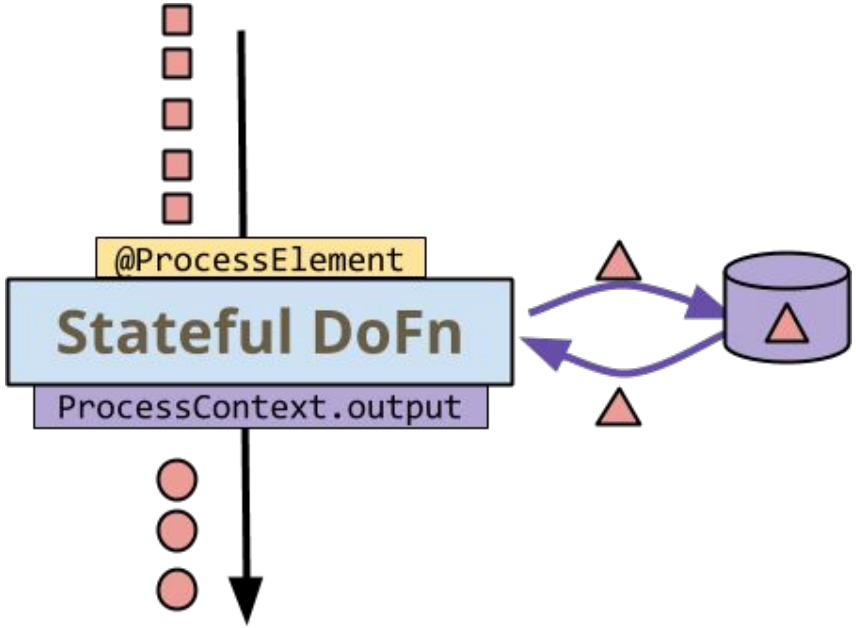
Stateful DoFn

Stateful DoFn

- I

- With state, a `DoFn` has the ability to access persistent mutable state while processing each input element.

- State is persisted per key per window

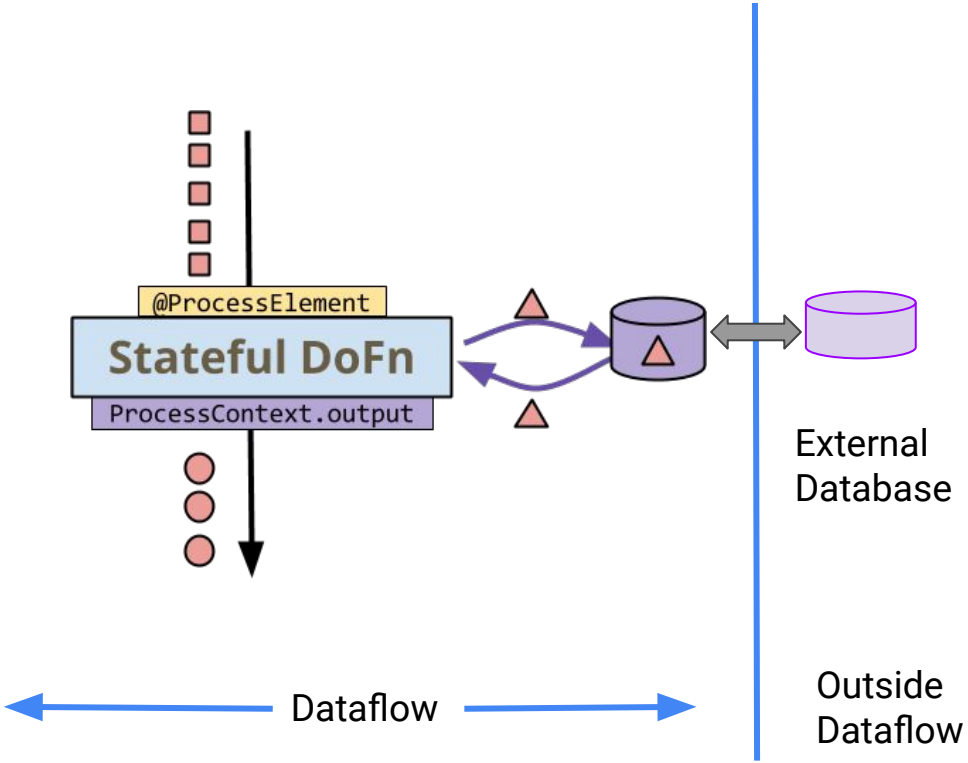


Stateful DoFn

- I

- With state, a `DoFn` has the ability to access persistent mutable state while processing each input element.

- State is persisted per key per window



Technical consideration on Dataflow

- For Batch jobs, state is stored in worker memory.
- For Streaming jobs using Streaming Engine its stored in Streaming Engine.
- For Streaming jobs without Streaming Engine, all the state is stored on the worker's disk locally. If the state size exceeds the disk capacity you may encounter a “*No space left on device error*”

External Cache

Memorystore

External Cache

- Set up an external cache to Dataflow
- Use service like Memorystore : Managed Redis and Memcached version

Acknowledgement

- Prathap Kumar Parvathareddy, Cloud Data Engineer, Google PSO

Questions?

BEAM SUMMIT Austin, 2022