

Error handling Apache Beam, and Asgarde Library

By Mazlum TOSUN



Group Bees





BEAM
SUMMIT

About me

Mazlum TOSUN



- ❖ Head of data and co founder at  Group Bees
- ❖ Tech lead GCP and data
- ❖ Passionate about Google Cloud, data, craft and functional programming
- ❖  Fan

<https://github.com/tosun-si>



<https://twitter.com/MazlumTosun3>



@MazlumTosun3

<https://www.linkedin.com/in/mazlum-tosun-900b1812/>



Asgarde



- ❑ Library allows simplifying error handling with Apache Beam
- ❑ Proposed for Apache Beam Java and Python
- ❑ A Wrapper is also proposed in Kotlin

Beam Java : native error handling ParDo and DoFn



- ❑ Beam recommends treating errors with a dead letter queue
- ❑ It means catching errors in the flow and, using side outputs, sinking errors to a file, database or any other output...
- ❑ Beam suggests handling side outputs with `TupleTags` in a `DoFn` class, example :

Beam Java : native error handling ParDo and DoFn



```
// Failure object.
public class Failure implements Serializable {
    private final String pipelineStep;
    private final Integer inputElement;
    private final Throwable exception;

    public static <T> Failure from(final String pipelineStep,
                                  final T element,
                                  final Throwable exception) {
        return new Failure(pipelineStep, element.toString(), exception);
    }
}

// Word count DoFn class.
public class WordCountFn extends DoFn<String, Integer> {

    private final TupleTag<Integer> outputTag = new TupleTag<Integer>() {};
    private final TupleTag<Failure> failuresTag = new TupleTag<Failure>() {};

    @ProcessElement
    public void processElement(ProcessContext ctx) {
        try {
            // Could throw ArithmeticException.
            final String word = ctx.element();
            ctx.output(1 / word.length());
        } catch (Throwable throwable) {
            final Failure failure = Failure.from("step", ctx.element(), throwable);
            ctx.output(failuresTag, failure);
        }
    }

    public TupleTag<Integer> getOutputTag() {
        return outputTag;
    }

    public TupleTag<Failure> getFailuresTag() {
        return failuresTag;
    }
}
```



Beam Java : native error handling ParDo and DoFn

```
// In Beam pipeline flow.
final PCollection<String> wordPCollection....

final WordCountFn wordCountFn = new WordCountFn();

final PCollectionTuple tuple = wordPCollection
    .apply("ParDo", ParDo.of(wordCountFn)
        .withOutputTags(wordCountFn.getOutputTag(), TupleTagList.of(wordCountFn.getFailuresTag())));

// Output PCollection via outputTag.
PCollection<Integer> outputCollection = tuple.get(wordCountFn.getOutputTag());

// Failures PCollection via failuresTag.
PCollection<Failure> failuresCollection = tuple.get(wordCountFn.getFailuresTag());
```

With this approach we can, in all steps, get the output and failures result PCollections.



Beam Java : native error handling MapElements FlatMapElement

Beam also allows handling errors with built-in components like `MapElements` and `FlatMapElements`

```
// In Beam pipeline flow.  
final PCollection<String> wordPCollection....  
  
WithFailures.Result<PCollection<Integer>, Failure> result = wordPCollection  
    .apply("Map", MapElements  
        .into(TypeDescriptors.integers())  
        .via((String word) -> 1 / word.length()) // Could throw ArithmeticException  
        .exceptionsInto(TypeDescriptor.of(Failure.class))  
        .exceptionsVia(exElt -> Failure.from("step", exElt))  
    );  
  
PCollection<String> output = result.output();  
PCollection<Failure> failures = result.failures();
```

Comparison between approaches : usual Beam pipeline



In a usual Beam pipeline flow, steps are chained fluently:

```
final PCollection<Integer> outputPCollection = inputPCollection
    .apply("Map",
        MapElements .into(TypeDescriptors.strings()).via((String word) -> word + "Test"))
    .apply("FlatMap",
        FlatMapElements
            .into(TypeDescriptors.strings())
            .via((String line) -> Arrays.asList(Arrays.copyOfRange(line.split(" "), 1, 5))))
    .apply("Map ParDo", ParDo.of(new WordCountFn()));
```


Comparison between approaches : Usual Beam pipeline with error handling



Here's the same flow with error handling in each step:

```
WithFailures.Result<PCollection<String>, Failure> result1 = input
    .apply("Map", MapElements
        .into(TypeDescriptors.strings())
        .via((String word) -> word + "Test")
        .exceptionsInto(TypeDescriptor.of(Failure.class))
        .exceptionsVia(exElt -> Failure.from("step", exElt)));

final PCollection<String> output1 = result1.output();
final PCollection<Failure> failure1 = result1.failures();

WithFailures.Result<PCollection<String>, Failure> result2 = output1
    .apply("FlatMap", FlatMapElements
        .into(TypeDescriptors.strings())
        .via((String line) -> Arrays.asList(Arrays.copyOfRange(line.split(" "), 1, 5)))
        .exceptionsInto(TypeDescriptor.of(Failure.class))
        .exceptionsVia(exElt -> Failure.from("step", exElt)));

final PCollection<String> output2 = result2.output();
final PCollection<Failure> failure2 = result2.failures();

final PCollectionTuple result3 = output2
    .apply("Map ParDo", ParDo.of(wordCountFn)
        .withOutputTags(wordCountFn.getOutputTag(),
            TupleTagList.of(wordCountFn.getFailuresTag())));

final PCollection<Integer> output3 = result3.get(wordCountFn.getOutputTag());
final PCollection<Failure> failure3 = result3.get(wordCountFn.getFailuresTag());

final PCollection<Failure> allFailures = PCollectionList
    .of(failure1)
    .and(failure2)
    .and(failure3)
    .apply(Flatten.pCollections());
```

Comparison between approaches : Usual Beam pipeline with error handling



Problems with this approach:

- ❑ We lose the native fluent style on apply chains, because we have to handle output and error for each step.
- ❑ For MapElements and FlatMapElements we have to always add exceptionsInto and exceptionsVia (can be centralized in a wrapper class).
- ❑ For each custom DoFn, we have to duplicate the code of TupleTag logic and the try catch block (can be centralized in a wrapper class).
- ❑ The code is verbose.
- ❑ There is no centralized code to concat all the errors, we have to concat all failures (can be centralized in a wrapper class).

Comparison between approaches : Usual Beam pipeline with error handling using Asgarde Java



Here's the same flow with error handling, but using Asgarde library instead:

```
final WithFailures.Result<PCollection<Integer>, Failure> resultComposer = CollectionComposer.of(input)
    .apply("Map", MapElements.into(TypeDescriptors.strings()).via((String word) -> word + "Test"))
    .apply("FlatMap", FlatMapElements
        .into(TypeDescriptors.strings())
        .via((String line) -> Arrays.asList(Arrays.copyOfRange(line.split(" "), 1, 5))))
    .apply("ParDo", MapElementFn.into(TypeDescriptors.integers()).via(word -> 1 / word.length()))
    .getResult();
```

Comparison between approaches : Usual Beam pipeline with error handling using Asgarde



Purpose of the library:

- ❑ Wrap all error handling logic in a composer class.
- ❑ Wrap exceptionsInto and exceptionsVia usage in the native Beam classes MapElements and FlatMapElements.
- ❑ Keep the fluent style natively proposed by Beam in apply methods while checking for failures and offer a less verbose way of handling errors.
- ❑ Expose custom DoFn classes with centralized try/catch blocks and Tuple tags.
- ❑ Expose an easier access to the @Setup, @StartBundle, @FinishBundle, @Teardown steps of DoFn classes, while error handling.
- ❑ Allow to concat all the failures occurred in the flow.
- ❑ Expose a way to handle errors in filtering logic (currently not available with Beam's Filter.by).

Comparison between approaches : Usual Beam pipeline with error handling using Asgarde Kotlin



Extensions are proposed to use Asgarde with Kotlin:

```
import fr.groupbees.asgarde.*

val result: Result<PCollection<Int>, Failure> = CollectionComposer.of(words)
    .map("Map") { word -> word + "Test" }
    .flatMap("FlatMap") { Arrays.asList(*Arrays.copyOfRange(it.split(" ").toTypedArray(), 1, 5)) }
    .mapFn("ParDo", { word -> 1 / word.length })
    .result
```

Pipeline example native error handling Python



```
# In Beam pipeline.
input_teams: PCollection[str] = p | 'Read' >> beam.Create(team_names)

outputs_map1, failures_map1 = (input_teams | 'Map to team with country' >> ParDo(MapToTeamWithCountry())
                              .with_outputs(FAILURES, main='outputs'))

outputs_map2, failures_map2 = (outputs_map1 | 'Map to team with city' >> ParDo(MapToTeamWithCity())
                              .with_outputs(FAILURES, main='outputs'))

outputs_filter, failures_filter = (outputs_map2 | 'Filter France teams' >> ParDo(FilterFranceTeams())
                                   .with_outputs(FAILURES, main='outputs'))

all_failures = (failures_map1, failures_map2, failures_filter) | 'All Failures PCollections' >> beam.Flatten()
```

Pipeline example Asgarde Python



```
# Beam pipeline with Asgarde library.
input_teams: PCollection[str] = p | 'Read' >> beam.Create(team_names)

result = (CollectionComposer.of(input_teams)
         .map('Map with country', lambda tname: TeamInfo(name=tname, country=team_countries[tname], city=''))
         .map('Map with city', lambda tinfo: TeamInfo(name=tinfo.name, country=tinfo.country, city=team_cities[tinfo.name]))
         .filter('Filter french team', lambda tinfo: tinfo.country == 'France'))

result_outputs: PCollection[TeamInfo] = result.outputs
result_failures: PCollection[Failure] = result.failures
```

Error handling code demo real application



Links to example projects



<https://github.com/tosun-si/teams-league-java-dlq-native-beam-summit>

<https://github.com/tosun-si/teams-league-java-dlq-asgarde-beam-summit>

<https://github.com/tosun-si/teams-league-kotlin-dlq-asgarde-beam-summit>

<https://github.com/tosun-si/teams-league-python-dlq-native-beam-summit>

<https://github.com/tosun-si/teams-league-python-dlq-asgarde-beam-summit>

Links of Asgarde projects



<https://github.com/tosun-si/asgarde>

<https://github.com/tosun-si/pasgarde>

<https://twitter.com/AsgardeBeam>

<https://www.linkedin.com/company/asgardebeam>

Help us, contributing to the projects and supporting us with Github stars



Thank you :)

