Oops, I Wrote a Portable Runner in Go

Robert Burke
@lostluck
Didn’t Finish Writing

Oops, I Wrote a Portable Runner in Go

Robert Burke
@lostluck
How to Implement an SDK Feature?

1. Look at how other SDKs do it.
2. Implement.
3. Unit test.
4. Integration testing.
Capability Matrix

<table>
<thead>
<tr>
<th>What is being computed?</th>
<th>Google Cloud Dataflow</th>
<th>Apache Flink</th>
<th>Apache Spark (RDD/DSStream)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParDo</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GroupByKey</td>
<td>✓</td>
<td>✓</td>
<td>~</td>
</tr>
<tr>
<td>Flatten</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Combine</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Composite Transforms</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
</tbody>
</table>

[https://beam.apache.org/documentation/runners/capability-matrix/](https://beam.apache.org/documentation/runners/capability-matrix/)
Map Side Inputs

ProcessElement(..., lookup func(K) func(*V) bool,...){
...
vals := lookup(key)
var val V
for vals(&val) { ... }
Map Side Inputs

ProcessElement(..., lookup func(K) func(*V) bool,...)
...
vals := lookup(key)
var val V
for vals(&val) { ... }

Declare the side input
Look up an iterator for a key
Iterate over associated values

FEATURE IMPLEMENTED!
Cross Bundle Cache

ProcessElement(..., lookup func(K) func(*V) bool,...){
...
vals := lookup(key)
var val V
for vals(&val) { ... }
import "github.com/apache/beam/sdks/v2/go/pkg/beam/util/harnessopts"

func main() {
    flags.Parse()

    harnessopts.SideInputCacheCapacity(*cacheKeyCount)

    beam.Init()
    ...
}
Cross Bundle Cache

```go
ProcessElement(..., lookup func(K) func(*V) bool,...){
...

timestamps.Update(ctx, time.Now().UnixNano())
vals := lookup(key)
var val V

for vals(&val) { ... }
timestamps.Update(ctx, time.Now().UnixNano())
```
Cross Bundle Cache

ProcessElement(..., lookup func(K) V...)
...

timestamps.Update(ctx, time.Now().UnixNano())
vals := lookup(key)
for vals(&val) { ... }
timestamps.Update(ctx, time.Now().UnixNano())
## Capability Matrix

### What is being computed?

<table>
<thead>
<tr>
<th>ParDo</th>
<th>Google Cloud Dataflow</th>
<th>Apache Flink</th>
<th>Apache Spark (RDD/DStream)</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GroupByKey</td>
<td>✓</td>
<td>✓</td>
<td>~</td>
</tr>
<tr>
<td>Flatten</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Combine</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Composite Transforms</td>
<td>~</td>
<td>~</td>
<td>~</td>
</tr>
</tbody>
</table>

[https://beam.apache.org/documentation/runners/capability-matrix/](https://beam.apache.org/documentation/runners/capability-matrix/)
Cross Bundle Cache

ProcessElement(..., lookup func(K) func(*V) bool,...){
...
timestamps.Update(ctx, time.Now().UnixNano())
vals := lookup(key)

for vals(&val) { ... }
timestamps.Update(ctx, time.Now().UnixNano())

FEATURE IMPLEMENTED ...?

Caching Observed?
Portable Worker

SDK

Runner

FnAPI
Beam has a Testing Problem
My Runner’s Goals

- SDK Testing focused
- Low Bar: Supercede Go Direct Runner
- Local & Single Machine
- In Memory
- Per-Pipeline Configurable
- High Bar: Run Java and Python pipelines

- Contribute it to the repo
Names are hard

Name possibilities

local: emphasises it’s not a distributed computation, but limits growth in that direction.
fake: tongue-in-cheek reference to test fakes which represent using a full implementation but may take some shortcuts that aren't appropriate in production
beam: It is what it is, but confusing to all future runners.
model: Clear, unambiguous that it represents implementation of the beam model. Might be confusing when discussion the model though (does model implement foo…)
unit: Emphasises its suitability for unit testing, in unoptimized mode.
portable: Might get conflated with the Python portable runner.
universal: Might get conflated with the decommissioned Java universal runner.
teach: emphasises the customizability and pedagogical benefits of the runner.
comp: emphasises the ability to validate different components.
collab: emphasises how the runner collaborates with SDKs to make testing things easier. But easily mistaken for Google Colab.
sdk: improved reference point runner -> each sdk has a Direct runner, so the term is confusing. Calling the the sdk runner or the gosdk runner avoids the confusion, and makes it rather clear about the implementation of it is. Bad point: then the packaging can get confusion: is the problem in the Go sdk or the gosdk?

<some not generic name>: care needs to be taken against being confused with other products and features in the data processing space.

handlebar: it gives control to pipelines for how it will execute (might be confused with the related moustache templates extension)
prism: It splits the Beam into its component parts, makes it visible.
lens: It focuses Beam into a pinpoint for consistent testing.
What I’ve got so far

https://github.com/lostluck/experimental/tree/master/local/internal
Loopback

https://github.com/lostduck/experimental/tree/master/local/internal

```go
func Execute(ctx context.Context, p *beam.Pipeline) (beam.PipelineResult, error) {
    if *jobopts.Endpoint == "" {
        // One hasn't been selected, so lets start one up and set the address.
        // Conveniently, this means that if multiple pipelines are executed against
        // the local runner, they will all use the same server.
        s := internal.NewServer(0)
        *jobopts.Endpoint = s.Endpoint()
        go s.Serve()
    }
    if !jobopts.IsLoopback() {
        // log.Info(ctx, "Environment type: %v, forcing loopback, as the local runner can
        *jobopts.EnvironmentType = "loopback"
    }
    return universal.Execute(ctx, p)
}
```
Single PTransform Execution

https://github.com/lostluck/experimental/tree/master/local/internal

```go
func() {
    // Send nil to start, Impulses won't require parental translation.
    processed <- nil
    for b := range toProcess {
        // Metrics?
        j.metrics.ContributeMetrics(-b.Resp)
        // Send back for dependency handling afterwards.
        processed <- b
    }
    close(processed)
}
```

```go
coders := map[string]pipepb.Coder{
    transforms := map[string]*pipepb.PTransform{
        tid: t, // The Transform to Execute!
    }
}
```

```go
reconcileCoders(coders, pipeline.GetComplete)
```

```go
desc := &fnpb.ProcessBundleDescriptor{
    Id:    bundID,
    Transforms: transforms,
    WindowingStrategies: pipeline.GetComplete,
    Pcollections: pipeline.GetComplete,
    Coders:    coders,
    StateApiServiceDescriptor: &pipepb.A
    Url: wk.Endpoint(),
}
GBKs

https://github.com/lostluck/experimental/tree/master/local/internal
What I’ve got so far

https://github.com/lostluck/experimental/tree/master/local/internal

- One PTransform at at time.
- ParDos
- GBKs
- Unlifted Combines
- Metrics
Example: CombineFns
CombineFn

Input I -> Accumulator A -> Output O

- CreateAccumulator () -> A
- Add Input (I, A) -> A
- Merge Accumulators (A, A) -> A
- Extract Output (A) -> O
CombineFn

Input $I$ == Accumulator $A$ == Output $O$

- CreateAccumulator $() \rightarrow A$
- Add Input $(I, A) \rightarrow A$
- Merge Accumulators $(A, A) \rightarrow A$
- Extract Output $(A) \rightarrow O$
Combiner Lifting

Unlifted

GBK

Merge

Lifted

GBK

Merge

Merge
**CombineFn**

Input \( I \) != Accumulator \( A \) == Output \( O \)

- CreateAccumulator \( () \rightarrow A \)
- Add Input \( (I, A) \rightarrow A \)
- Merge Accumulators \( (A, A) \rightarrow A \)
- Extract Output \( (A) \rightarrow O \)
Combiner Lifting

**Unlifted**

- Add Input
- Extract Output
- GBK

**Lifted**

- Add Input
- Merge
- Extract Output
- GBK
Combiner Lifting

- Used in Batch execution
  - Bigger bundles, more benefit
- Not used in Streaming execution
  - Small bundles, negligible benefit
Example:
SplittableDoFns
Splittable DoFns

- CreateInitialRestriction
- CreateTracker
- SplitRestriction
- RestrictionSize
- ...and more
My Runner’s “Secret” Goals

Production Ready?

- Modular
- Production features
  - Optimized Execution Graph
  - Disk Spillover
- Automation of Testing SDK Features
Oops, I Wrote a Portable Runner in Go

Robert Burke
@lostluck
Related Talks

**Writing a native Go streaming pipeline**
Tuesday 16:15-16:40 CDT, Room 203
with Danny McCormick and Jack McCluskey
https://2022.beamsummit.org/sessions/native-go-pipeline/
Direct
Runners are
Bad
Direct Runners are Bad