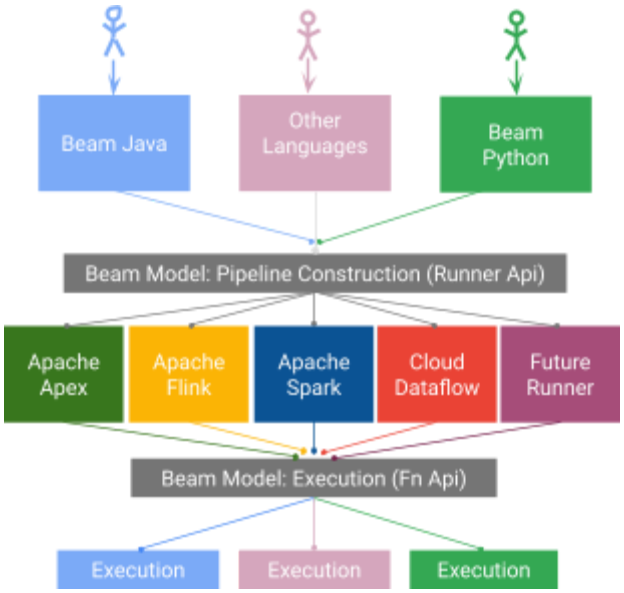
Apache Beam Fn API Overview

https://s.apache.org/beam-fn-api
Lukasz Cwik (lcwik@google.com)

The Beam technical vision includes the ability to mix and match SDKs and runners. This is accomplished through two portability layers:

1. The Runner API provides an SDK-and-runner-independent definition of a Beam pipeline
2. The Fn API allows a runner to invoke SDK-specific user-defined functions



Specifically, the Runner + Fn API allow:

1. new SDKs to run on every runner
2. new runners to run pipelines from every SDK
3. (eventually) mixing transforms from different SDKs in one pipeline, hence leveraging each language's connector libraries, etc

For the Fn API to be as useful as possible, it has two major design criteria:

1. It should be easy for an SDK to get started with a minimal Fn API implementation
2. It should be possible for an SDK to be performant by investing additional effort

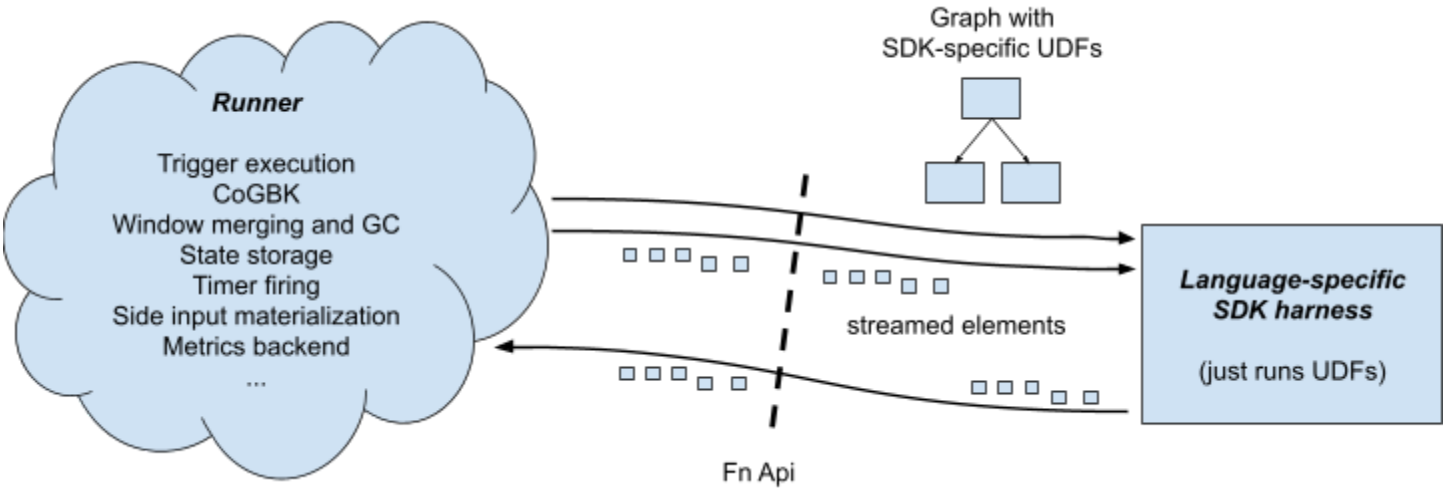
Overview

Executing a user's pipeline can be broken up into the following categories:

1. Perform any grouping and triggering logic including keeping track of a watermark
2. Execute user definable functions (UDFs) including custom sources/sinks, splittable DoFns, regular DoFns, window merging, ...
3. Support the execution of user definable functions with side inputs, state & timers, counters, ...
4. Pipeline execution management such as polling for counter information, job status, ...

Executing UDFs is the only category which requires a specific language-specific SDK context to execute in. Moving the execution of UDFs to language-specific SDK harnesses and using an RPC service between the two allows for a cross language/cross Runner portability story.

Executing UDFs one by one in this manner would be terribly slow, so the basic design is to execute small subgraphs over streamed bundles of elements. This diagram summarizes:



Components

This sketch proposes using a docker container to encapsulate the language-specific SDK harness. The runner would be responsible for launching, managing and tearing down the docker container. Supporting the execution of UDFs are the following gRPC service definitions:

- Control: A rich low bandwidth service used to tell the SDK which UDFs to execute and when to execute them. Also handles bundle processing progress, splitting, and counters.
- Data: A high bandwidth low latency service used to move data between the language specific SDK harness and the runner, modelled as a logical stream of bytes.
- State: A mid to high bandwidth service used to support user state, side inputs (using the paged read, half of the read/write state API), and group by key reiteration (ditto).
- Logging: A low to mid bandwidth service used to aggregate logging information from the language specific SDK harness.

API & Code References

Runner API Protobuf definition
Fn API Protobuf definition

Stories

How to process a bundle using the Fn API
How to send and receive data over the Fn API
How to access side inputs, access remote references, and support user state over the Fn API
How to report bundle processing progress over the Fn API
How a users container is initialized/shutdown over the Fn API
Breaking the fusion barrier: Deep splitting of Beam instruction graphs
Optimize combiners by lifting them before the GBK
Handling user and system metrics
Modeling, scheduling and executing timers
How to finalize bundles

Preliminary Results

A preliminary implementation was created to test Dataflow with the components above. Pipeline execution time was increased by approximately 15% when used with a pipeline which read 10 GiBs of data, followed by a group by key and then an identity ParDo. Pipeline execution time was measured from when the first byte was read to when the last byte was processed in the ParDo and did not include any VM startup overhead. Note that approximately 80% of the overhead came from encoding and decoding the data across the Fn Api and could be significantly reduced by optimizing encoding/decoding paths. Also note that this benchmark pipeline is atypical as user processing time was minimal while normal pipelines have non-trivial ParDos so this relative overhead decreases as user processing time increases.

Benefits

By using Docker combined with a gRPC interface, the Fn API as sketched here will deliver the following benefits:

- Providing clean APIs clarifying and reducing the knowledge needed to get an SDK started.
- Enables multi-language pipelines, allowing reuse of connectors and libraries (ML, etc) between SDKs.
- Gives a portability story for the Python SDK (or any Beam SDK) on any runner supporting the Fn API: given a way to construct a runner-specific execution plan with embedded UDFs (the Runner API will be the runner-agnostic way to do this) the runner can then execute UDFs in any language-specific SDK.
- Reduces the risk when upgrading or patching portions of a runner due to the reduction in coupling.
- Solves dependency issues caused when user code requires an environment different than the harnesses.
    - Key for Python users for customizing the user environment.
    - Relevant for Java users when dependencies conflict.
    - Also helpful when users want to configure their runtime environment by installing additional libraries or including large static data files.