Improving the performance of your pipeline

Israel Herraiz

Strategic Cloud Engineer, Google

http://twitter.com/herraiz

Performance - Planning ahead

With good pipeline design it is often possible to avoid performance issues before they arise.

In these sections we will look at

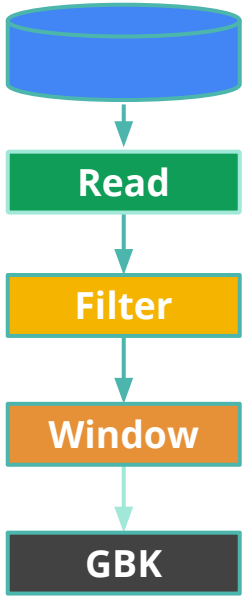
- Pipeline Design Considerations
- Effects of Data Shape
- Interactions with Source, Sinks and external systems
- Runner specific performance options
- Performance Debug
- Runner specific aids
    - Dataflow

Design : Topology

Filter first:

Always place transformations that reduce the volume of data as high up on the graph as possible.

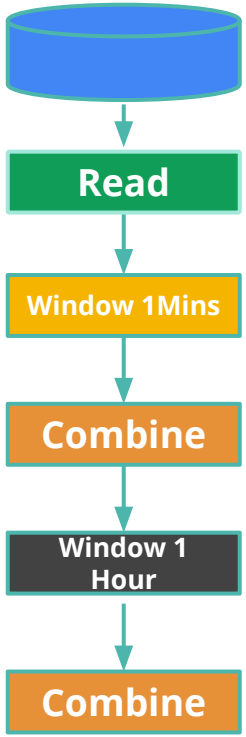
This includes above Window operations, even though the Window transform itself does nothing more than tag elements in preparation of the next aggregation step in the DAG..



Design: Window Configurations

Reducing the effects of large sliding windows

If the period of the sliding window is X seconds, you can create Window + Combine before the main sliding window to reduce the volume of elements to be processed when the Window slides.



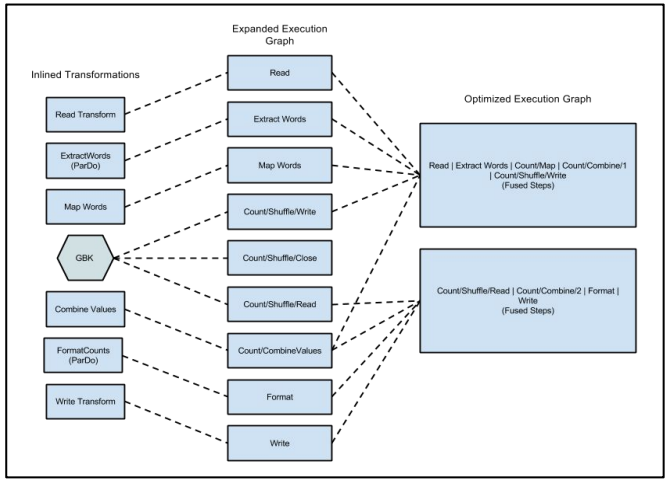
Design: Graph Optimization Preventing Fusion

Runners may support Fusion as part of Graph optimization. Such optimizations can include fusing multiple steps or transforms in your pipeline's execution graph into single steps.

There are a few cases in your pipeline where you may want to prevent the Cloud Dataflow service from performing fusion optimizations.

The primary example where fusion is not desirable is Large Fanout transforms, where a single element can output hundreds or thousands of times as many elements.

- You can insert a Reshuffle after your first ParDo. The Cloud Dataflow service never fuses ParDo operations across an aggregation.
- You can pass your intermediate PCollection as a side input to another ParDo. The Cloud Dataflow service always materializes side inputs.



Design: Coders

Choose coders that provide good performance. For example in the Java SDK:
Do not use SerializableCoder, choose a more efficient coder for example:

- ProtoCoder
- Schemas

Pro Tip:
Encoding/decoding is a large source of overhead. Thus if you have a large blob but only need part of it structured you could selectively decode just that part.

For example with protobufs com.google.protobuf.FieldMask is perhaps an easy way to do that for protos. So instead of parsing full blob to proto and then extracting a key to group by you could decode to Key, full blob and then just output blob to the key.

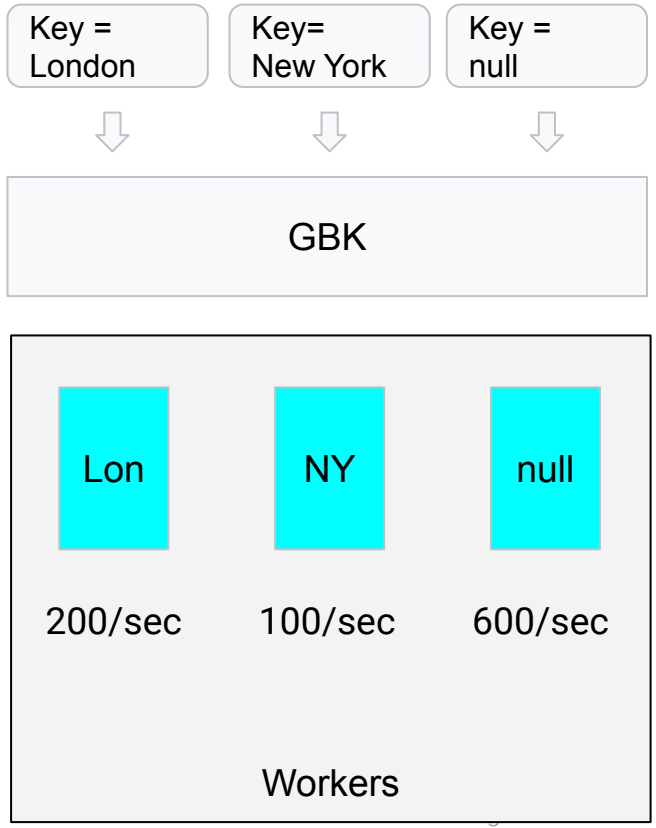
Design: Logging

One of the common service ticket resolutions for performance comes from that old favourite too much logging!

- In the Dataflow runner Logs are sent from all workers to a central location in Stackdriver.
- 1000 machines all pushing 100 logs per sec can cause massive back pressure!!!
- Log.info should nearly always be avoided against PCollection Element granularity. These will rarely be useful in logs.
- Log.error should also be carefully considered, will a Deadletter pattern followed by a Count per window of 5 mins be better suited for reporting data errors for example?

Data Shape: Data Skew

During a GBK keys will be shuffled to workers, keys will be sent to the same machine throughout the process.



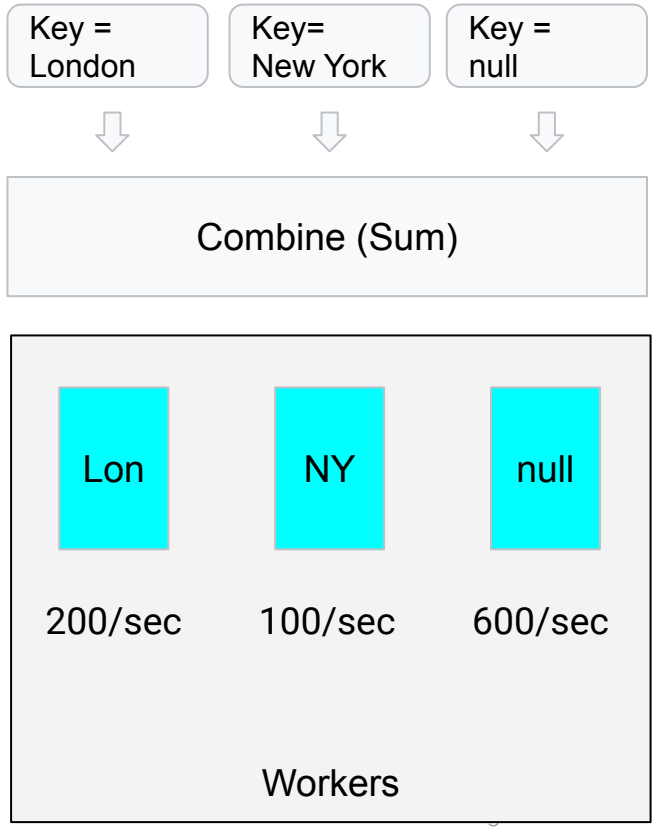
* Tip: Oftens columns used as keys which are @nullable end up being hotkeys

Data Shape: Data Skew

This will also be true in the last phase of a combine.

withFanout(int)
Allows for the defining of intermediate workers before the final combine step.

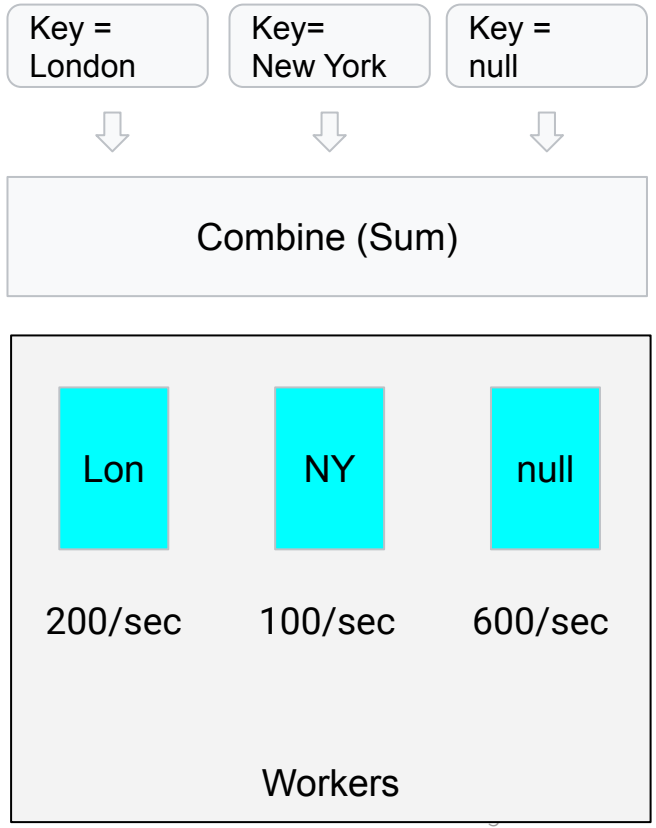
withHotKeyFanout(Sfn)
Available for Combine.perkey and allows for a function to determine intermediate steps.



Data Shape: Data Skew

This will also be true in the last phase of a combine.

Use Cloud Dataflow Shuffle (Batch)
Uses the Dataflow service rather than workers for the shuffle operation.



Data Shape: Key Space & Parallelism

If there are a limited number of keys in a dataset which are going through a non transitive and associative computation;

The amount of parallelism will == the number of keys. More machines will not be able to do anymore work.

* Pro Tip! If windows are distinct, the window can be added as part of the key to shard work across more workers. Adding the window to the key improves the ability of the system to parallelize processing since those keys can now be processed in parallel on different machines since they are now recognized as unrelated.

Data Shape: Key Space & Parallelism

Not a hard requirement, but general guideline:

- Too few keys: bad - hard to shard workload, and per-key ordering will kill performance
- Too many keys: can be bad too - overhead starts to creep in. if the keyspace is very large, consider using hashes separating keys out internally. This is especially useful if keys carry date/time information. In this case you can "re-use" processing keys from the past that are not active anymore essentially for free.

Sources, Sinks and External Systems

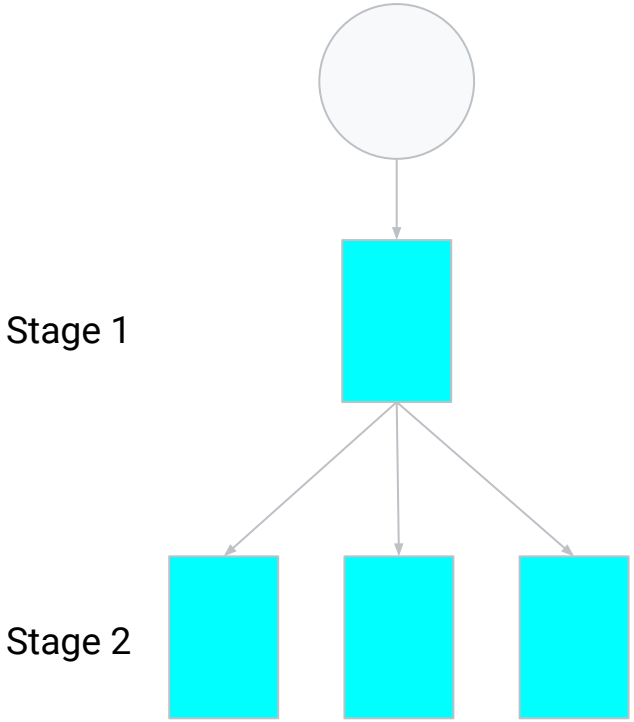
Most sources and sinks should abstract the user from the need to deal with the Read stage parallelism.

Common Issues:

text.gz files being read via TextIO; as TextIO can not read to an offset, a single thread will deal with each file. This will have three negative effects:

1. Only one machine can do the, normally slow, read I/O
2. After the Read Stage all fused stages will need to run on the same worker that read the data.
3. In any shuffle stage a single machine will need to push all the data from the file to all other machines. The single host network becomes the bottleneck.

Switch to uncompressed files or switch to compressed Avro.



Sources, Sinks and External Systems

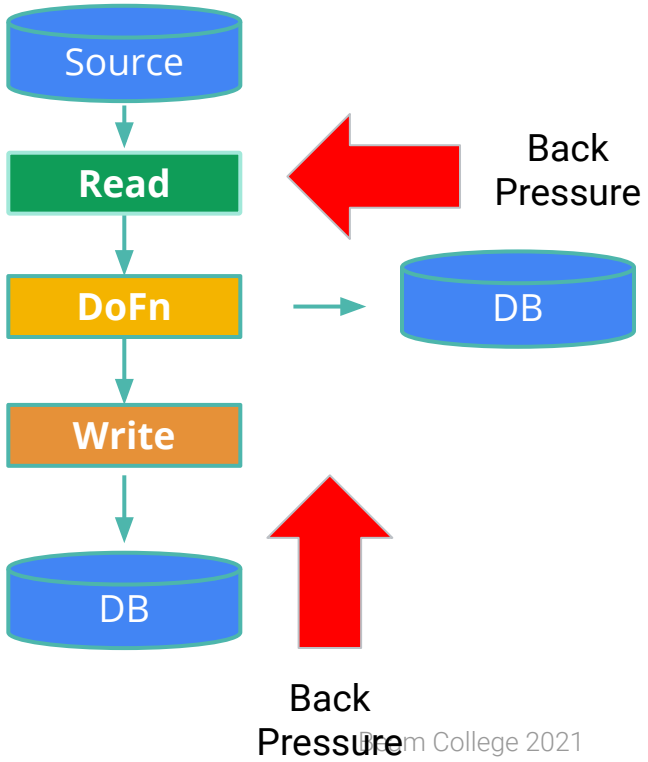
Back Pressure:

Beam runners are designed to be able to rapidly chew through parallel work. They can spin up many threads across many machines to achieve this goal.

- This can easily swamp an external system.
- This is an issue for both batch and stream pipelines, the effects on the external systems are often more pronounced in batch or during backlog processing in a stream pipeline.

For Side-effects from a DoFn where the external system can not deal with the qps flowing through the DoFn:

- Make use of batching mechanism in the call to the external system. You can use mechanism like GroupIntoBatches transforms or @StartBundle, @FinishBundle.



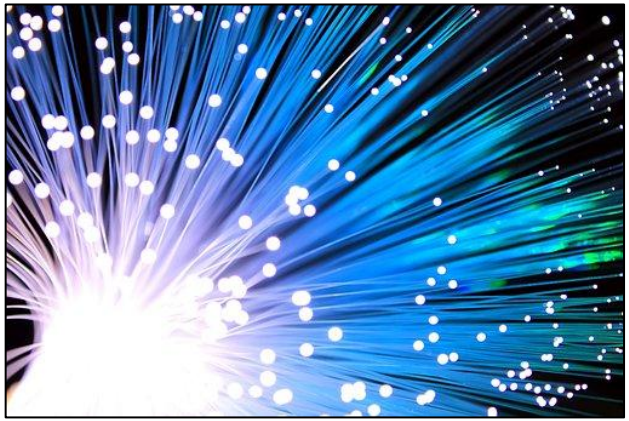
Sources, Sinks and External Systems

Speed of light .... yup still an issue

The physical location of sources and sinks versus the zone that workers are started is important. The Google backbone network is awesome, but Speed of light.... Quantum entanglement maybe? :-)

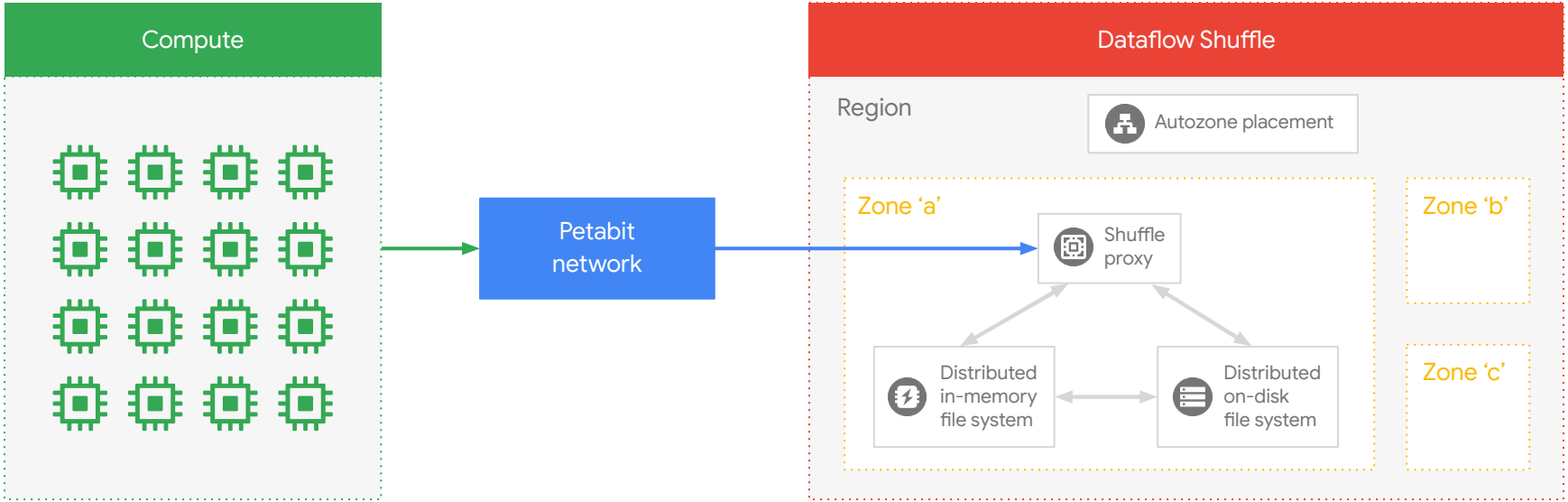
Also consider the Dataflow regional end point vs the worker zones as there are more worker zones than regional endpoints.

https://cloud.google.com/dataflow/docs/concepts/regional-endpoints



https://pixabay.com/photos/fiber-optic-cable-blue-network-2749588/

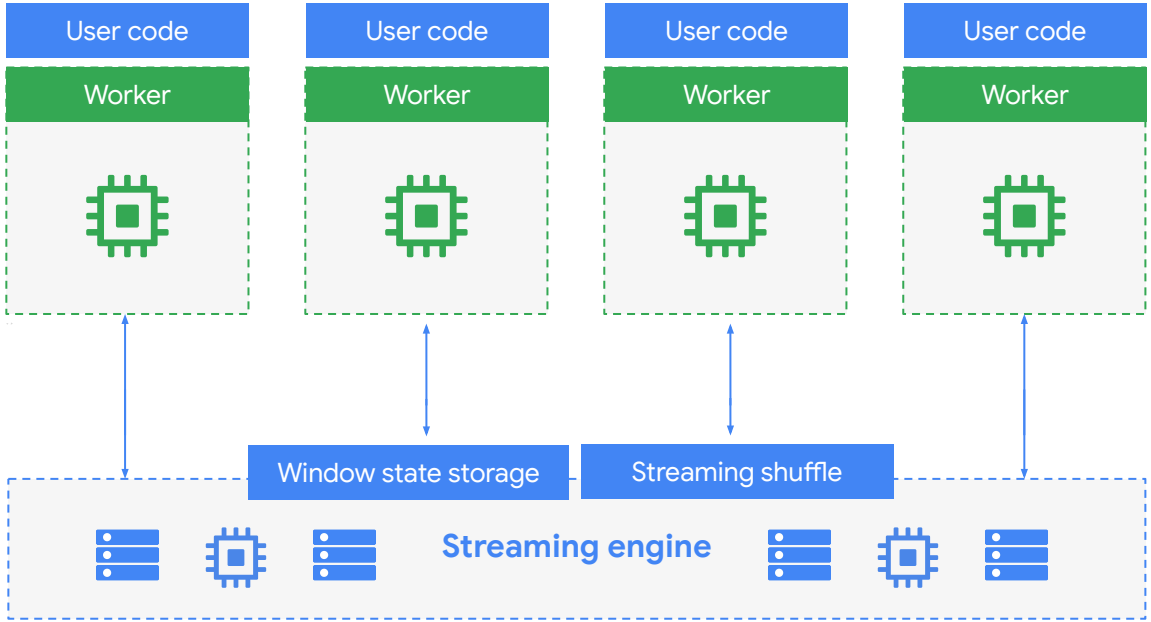
Runner specific options - Dataflow Shuffle Service - Batch



Runner specific options - Dataflow Streaming Engine

Benefits

- Smoother autoscaling
- Better supportability
- Less worker resources



Summary

1. Design Considerations.
2. Data Shape.
3. Source, Sinks and external systems.
4. Runner specific performance options.

Thank you!

Questions?