Guidelines for the Butcher

This document is intended to help you understand the purpose the Butcher plays in the system, and how to implement it properly/safely. We provide some sample module-packages in the repository for reference. The idea is that it should be relatively easy to set this up while providing enough flexibility for a wide range of applications.

You are ultimately responsible for the correctness and the strategies chosen to implement each component.

Any application that uses this framework requires what we call a *Butcher*. It is responsible for segmenting your inputs and merging your outputs in a way that allows for compatibility with the Parallel network. It consists of at least 2 required files, *"Split.py"* and *"Merge.py"*, you may include any other helper files. Split will provide the ability for the chief to serve data fragments on demand. Merge will combine your sub results into a combined final-result.

Split.py

The main concept is to use python generators to serve your inputs on demand. What is a generator you may ask. Put simply generators *yield* data instead of returning it like a normal function. That means specifically that the generator loads one smaller chunk of data into memory at a time. The generator saves its internal state before yielding the data and iterating to the next state. With Python this is actually super easy to do in practice.

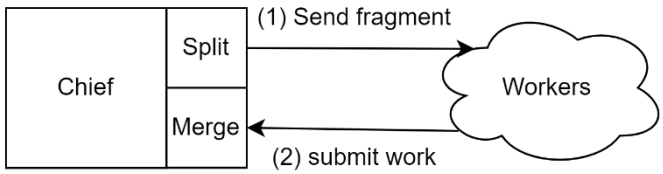
It involves defining a function called *splitter()* that segments your data or defines some kind of iterable, **finite** input space. For example, when you normally move data to some kind of buffer in a loop you can instead just yield it to the caller.

Merge.py

Merge takes all the work submitted back to the chief and combines the data however the user sees fit. Results are merged in causal order and are cached by the chief. Your merger method receives fragments that are causally related to each other, so you do not have to worry about ordering when implementing the merger. So, if you receive 1 then 4, the chief will not merge 4 into the result until it receives 2 and 3. This is an option that could be turned off if it is not applicable to you.

This involves defining a function called *merger()* that takes a result segment and maps it to a final result. This method has access to the results directory and can use it however they want. The chief will tar this and send it back to you after completing the job.

Our backend will use the methods you defined to segment data and merge results lazily as it receives requests from workers.



This diagram represents the processing loop, it starts when the client starts a job, using the splitter to distribute the first round of fragments. As each worker finishes their task, they submit their work back to the chief and it gets processed by the merger in turn causing another the next fragment to be sent out to that worker. After the input space has been exhausted the chief will stop posting data to the workers and wait for all the sub result.

Requirements

- NO malicious applications
- NO component should exceed some maximum recursion depth enforced by the chief.
- NO trying to break out of the docker container or gain access to the local system.
- NO running a server or proxy through workers.
- NO creating docker containers or virtual machines.
- NO access to external networks, like the internet.
- NO component may exceed some maximum concurrency parameter enforced by the chief. Such as opening a large number of threads or sub processes.
- The *Splitter()* method defines a python generator using the *yield* keyword. No other method in your package should yield anything.
- Your splitter does not produce infinite sequences.
- Your merger does write an amount of data that exceeds some maximum size enforced by the chief
- The fragment size is at most the maximum size enforced by the chief.

Consequences

- Your application will not work, and the network will refuse to execute the job.
- You will waste time writing an application that is not compatible with our architecture.
- Suspicious activity might trigger containers to reset and purge your code.
- You will most likely just waste your own resources and you may be blocked from accessing the system.

Assistance

We suggest if you are having trouble picturing certain implementation details that you look at the sample modules provided in the repository and play with them. We also encourage you to use chat-GPT to help write this module if needed and you can provide it with the requirements. To learn more about python generators you can read this article.

https://realpython.com/introduction-to-python-generators/