Moving Compute To Data

This document contains discussions / plans about moving computation towards data.

Moving the computation, i.e. data manipulation or analysis code, closer to the data is becoming a much more frequently utilized approach when dealing with large data sets. For example, if A hosts a data set and the analysis code on that data is running on machine B, as the size of the data gets larger it becomes increasingly impractical to move the data from A to B for the analysis to run. The more frequently used alternative in these cases, especially as portable containerized code has become more practical with technologies such as docker, is to move the containerized analysis code over to the the machine hosting the data and executing it their as opposed to moving the data (given that the containers are significantly smaller than the datasets and assuming some computational resource is also available on or near the server hosting the data). With the dockerization of extractors and converters as part of the previous milestone we now address this optional means of carrying out transformations within Brown Dog on larger datasets locally.

Scenario

A user wishes to convert and then extract some statistics on a locally hosted 8 GB weather dataset. The user is using the Brown Dog command line interface to do this. Below we outline the modifications needed by each of the components in order to allow these transformations to be carried out locally (i.e. moving the compute to the local data).

DTS (Clowder)

- An endpoint (see this page for more details) needs to be created either in Clowder or in Fence which returns the list of docker containers containing each extractor that would be fired off given a file/dataset of a given type. Parameters will be the mime type (requiring the client to determine this) or the file extension (requiring the server side to fill in the full mime type for Clowder).
- A minor modification will be needed to the extractors such that if the extractor is executed with a single argument (i.e. the path to the locally stored file to operate on) the extractor will bypass connecting to rabbitmq, use the argument as the input, run on it, print the generated JSON to the screen, and immediately exit. Might be possible to add a single method to pyClowder to handle this overall (will need to explore).
- The default entrypoint.sh file for the dockerized extractor will need to be modified to allow for an optional argument to be passed in.

DAP (Polyglot)

- An endpoint needs to be created which when given an input file type and an output file type returns an ordered list of docker containers and output formats to be generated by each in order to get to the desired target format. This ordered list will be the path generated by the I/O-graph.
- When given command line arguments containing a path to a locally stored file, a specific tool, as well as a desired output format the Software Server will execute a conversion using that tool to the desired output format, printing the path to the output file, and immediately exit as opposed to connecting to rabbitmq and waiting for jobs. This functionality may already exist, need to verify.
- The dockerfiles will need to be modified to allow for optional arguments to be passed in.

BD CLI

- A flag will need to be added trigger the movement of the tool containers as opposed to the data to the server. An other optional feature would be to determine whether to do this automatically based on the file size.
- A new extract method will need to be created utilizing the above endpoint, downloading then executing the identified tool containers.
- A new convert method will need to be created utilizing the above endpoint, downloading then executing the identified tool containers. Note, this will need to be a different method from the extract method as the behavior is different (e.g. conversion will require an ordered execution of the tools along with a parameter defining the output format at each step).

Once completed the bd command line interface might be utilized as follows in order to carry out the desired data transformations:

```
bd -o --bigdata pecan.zip region.narr.zip | bd -v --bigdata
```

Development Notes

DTS endpoint to determine needed containers:

1. Queries the RabbitMQ server to get all the available the queues (/api/queues/**vhost**). If vhost is not set in the config file, it uses the default one (%2F).
2. Then, for each of the queues, it again queries the server for receiving the bindings (/api/queues/***vhost***/***name***/bindings), where vhost (default is %2F) is obtained from the config file and name (i.e. queue name) is obtained from the previous step.
3. The bindings returned for a particular queue are searched for matching MIME types in the routing key. If this is found, the corresponding queue name is appended to the result array.
4. Finally, when all the queues have been traversed, the result array is returned to the user in JSON format.

DAP:

1. Query DAP for conversion path, http://bd-api-dev.ncsa.illinois.edu/dap/path/<output>/<input>, get path back in JSON, e.g. nc to pdf would return:
    a. `[{"input":"nc","application":"ncdump","output":"txt"},{"input":"txt","application":"unoconv","output":"pdf"}]`
2. Pull docker containers for applications specified in conversion path

    a. Requires ~~BD-1243~~ - Redeploy dap-dev DONE
3. Modify https://opensource.ncsa.illinois.edu/bitbucket/projects/POL/repos/polyglot/browse/docker/polyglot/entrypoint.sh so that if rabbitmq url is not set it instead runs the local version ~~BD-1312~~ - Modify entrypoint in Polyglot for local processing DONE :
    a. SoftwareServer.sh -nocache <Application> <operation> <output format> <input file>