Introduction:

The eau2 is a distributed key value store. Nodes are run withing clients from our client server network w e created previously. Each node holds a section of the columns of a inserted dataframe, and can run row er operation on its section. How it works: Store a dataframe by putting it into the key value store. The ke y value store then splits the columns of the dataframe into subcolumns. The subcolumns are serialized i nto a byte array and each byte array is sent to a different client. The clients then deserialize and reconstr uct the subset dataframe. The reconstructed dataframe is inserted into the node's internal key value sto re alongside the key that the put message arrived with.

When a dataframe get call is made to the key value store, the store sends a get message to each client. T he clients then pull the dataframes out of their nodes, serialize them, and send them back to the server. The server deserializes each group of subcolumns, puts them back together, and reconstructs the origin al dataframe from them.

The eau2 has functionality to run distributed calculations (via rowers) as well. After storing a dataframe, the server can send action messages to trigger the clients to run a specific rower on the node's stored su bdataframes. The clients then serialize and send the rowers back to the server, which joins them togeth er to produce the resulting data.

Architecture:

Server - tells the NetworkServer what to send/gets stuff from clients through NetworkServer

NetworkServer - communicates with clients

Client - runs the nodes in distributed network, communicates with server

Node - client class sends received messages to Node, actual KV pairs stored here

Rower - runs operations on the DataFrame

Implementation:

On Server

Create a NetworkServer

Create a Server which takes an instance of a NetworkServer

Create a DataFrame

Upload DataFrame to distributed network by calling Server->put(Key* key, DataFrame* df)

Run operations on the DataFrame by creating a Rower and calling Server->run_rower(Rower* r, DataFrame* df)

Get a DataFrame from the distributed network by calling Server->get(Key* key)

On Client

Create a Client

Run Client->register_ip() to register client to network

Run Client->be_client() to start client

Use cases:

Makefile commands:

make run_[word_count] - runs the word count application in a non-distributed manner

make run_[linus/server/client] - run the server (or linus as server), then the client in separate processes. Server cli: server_ip num_clients. Client cli: client_ip server_ip.

make build_[...] - builds a specific test case

make run_[...] - builds and runs a specific test case

make valgrind_[...] - builds then runs valgrind (with --leak-check=full) on a specific test case

See linus.cpp and test_client.cpp for how to set up and run the degrees of linus program.

Currently the makefile specifies that the linus program will run with 3 clients, but this can be changed either in the makefile or by running from the command line.

Set the configuration for the degrees of linus program at the top of linus.cpp.

The 3 PATHS must be set!

To run (in separate terminal clients):

- make run_linus

- make run_client

- make run_client_2

- make run_client_3

- ./test_client.out <address>:<port> <server_addr>:<server_port>

Open questions:

How do we efficiently debug a program that takes upwards of 10 minutes to get to the crashing point? If running on big data causes the problems how should we avoid waiting for the program to crash every ti me we try to run it?

Status:

- Eau2 seems to work entirely.

- When running on large (> 1GB) data and more than 3 clients, some clients crash at the end.

  - Unsure of why, but it seems that cpu does not multithread that high?

  - Some clients tend to take turns running the rower instead of all going at the same time.

- All data leaks (except a small fixed size one) have been eradicated.

- Degrees of linus currently has to be run from multiple processes, there is no single file demo.

TODOs:

- create util file for repeated functions (parse_messages, encode).

- write more unit level tests.