Introduction

Eau2 is a distributed in-memory key-value store which allows users to perform analysis on data in terms of Dataframe abstraction. The system is designed to handle arbitrary sized data by distributing storage across several storage nodes. Its distributed nature is hidden from the user, who primarily work with dataframes and save intermediate results by assigning keys to them.

To run the prepared example program that comes with the repository, please use `make docker_run`. To run the tests and memory check, please use `make valgrind`.

Architecture

The system is divided into three layers: - an Application layer on the top - a Key-Value store in the middle - a network layer on the bottom

Application Layer

The application layer is composed of a range of classes that provides the user the tools to perform their computations.

Among the key functionality that it supports, we have: - a way to configure the cluster underneath (e.g. number of nodes) - a mechanism to load data into the system - a mechanism to store/retrieve data - an overridable function that allows users to implement their logic

System Configuration

To configure the underlying system that gets started, the Application will take in command line arguments at the start-up of the application. Among the configurations, we have the number of nodes in the cluster, max chunk size, the ip addresses of node and server, the file to read the data from...

Loading Data

Currently, the only file format supported is .sor through the `fromFile()` method provided by the `Application` superclass. However, it is possible to parse other file formats through the `fromVisitor()` method, by providing a user defined visitor.

Data Storage & Retrieval

The storage of the data is handled according to the following strategy: - Fixed sized chunks of data of the same type (can think of them as sections of an array) are created and associated a key. - These chunks are then distributed across the nodes of the network making sure that all the sections belonging to the same row are stored in the same node.

This strategy provides performance benefits by permitting local map operations without the necessity to request data over the network.

User defined DataFrames are always stored locally to the node that created it, and the associated key is broad-casted over the network. The DataFrames are not chunked, since they only contain keys that are associated to actual chunks of data.

User Defined Logic

The Application provides an overridable function where users can to implement and run their own business logic on the data. To implement distributed computation, each node in the network comes with associated identifier that the user can use to run computation on that specific node. The identifiers start at 0 (always the node that is designated to be the server) all the way to `N-1`, where `N` is the number of nodes.

Key-Value Store (Middle-Layer)

The Key-Value Store contains the actual logic to store & retrieve data. This includes: - managing the local storage - handling all the network communication - caching

Since the Key-Value Store is the only access point to the underlying network, multiple classes on the top level (currently DataFrame and Application) has access to it.

Node (Bottom-layer)

The node encapsulates all the networking logic and functionality used by the above Key-Value Store. When an Eau2 application is started, the node is responsible for starting and connecting the nodes in the network according to some `NodeConfiguration` that is passed down from the application level.

The current initialization strategy changes depending on the node being a client or a server.

The server initialization follows these steps: - start server - listen to incoming connection - register the required number of nodes (all of them) - broadcast their addresses to all the other nodes

The client initialization follows these steps: - start client - connect to server (at the known address) - wait for the address book from the server - connect to all the other nodes

Implementation

Following are the description of the main classes on which Eau2 is built. We describe them from in order from top layer to bottom layer.

Application Layer

Application

Application is a the top layer class that a user would extend to and override to create a custom program. The class has the following fields:

```
// the unique id of the node
int id;

// the Key-Value Store used to store and retrieve
KVStore store;

// the number of items in each chunk of data
size_t chunkItems;
```

Among the methods, we have:

```
// creates a dataframe given a .sor file
fromFile(Key* k, const char* file);

// creates a dataframe using a writer visitor
fromVisitor(Key* k, const char* types, Writer* v);

// creates a dataframe from a single value
// there can be multiple overloaded version of this method to created
// dataframes from different types values
fromScalar(Key* k, ...);

// an overridable method for users to implement business logic
void run();
```

```
// gets the id of this node
getNodeId();

// retrieves a dataframe from the store. Does not wait its creation.
get(Key* k);
// retrieves a dataframe from the store. Waits its creation.
waitAndGet(Key* k);
```

Writer & Reader

Two super classes extended by user defined visitors to implement custom logic to create dataframes or compute on them.

Shared: Application layer & Middle layer

The following classes are shared among the top and middle layer.

Key

The Key class is used to encapsulate metadata associated with a value. Currently the metadata are: - a unique string identifier - the node location of the data the key refers to.

Value

The Value class is used to encapsulate the actual data. Currently it stores the data and the size of the data. The data is stored as a `char*`.

DataFrame

The DataFrame class is the data structure used to interact with data in Eau2. It stores metadata such as the data type of the columns, the number of columns, and number of rows. The current data types supported are: - 'B': boolean - 'I': integer - 'D': double - 'S': String

Since the underlying data is distributed across the network, the DataFrame is responsible for keeping track of where the data is located. To do so, it holds a `DistributedColumn` object which is a wrapper around a table of `Key`s. The actual logic to retrieve the data is delegated to the `DistributedColumn` class which has access to the Key-Value Store underneath.

DataFrame is a read-only class and does not provide methods to change its data.

Middle-Layer

Classes in the middle-layer include:

KVStore (Key-Value Store)

The KVStore is responsible for storing & retrieving the data. It has access to the network and contains serialization/deserialization as well as caching logic.

There are two kinds of data stored in the KVStore: - chunks of actual data - serialized dataframes

Both are stored in the `store` field, an `std::unordered_map` of `Key`s and `Value`s.

The KVStore also stores user defined keys in a separate array of keys, to keep track of where that data is stored.

Among the primary methods, we have:

```
// Stores the given value associated with the given key to the
// node specified by the key
// Note: this method steals the arguments!
put(Key* k, Value* v);

// Gets the value associated with the given key
get(Key* k);

// Gets the value associated with the given key.
// Wait for the key to be broadcasted if not yet in the server.
waitAndGet(Key* k);
```

Network layer

This layer is primarily comprised of the `Node` class.

Node

The Node class encapsulated all the networking logic necessary for the Eau2 to work. Among the functionalities, we have: - connect to nodes - send messages - receive messages

Use Cases

To use Eau2, the user will create a class that extends the `Application` and override the `run()` method, which will contain the user's business logic. A basic implementation and flow of the `run()` method is as follows:

General usage example:

```
// In DemoApplication class

void run() override {
    // server node is 0
    if (this->getNodeId() == 0) {
        delete this->fromFile(Key("data"), "file.txt");
    }

    Dataframe* df = this->waitAndGet(Key("data"));
    long long sum = 0;
    for (size_t i = 0; i < df->width(); i++) {
        for (size_t j = 0; j < df->length(); j++) {
            sum += df->getInt(i, j);
        }
    }
    // N is the local node id
    delete this->fromScalar(Key("local_sum_N"), sum);
    if (this->getNodeId() == 0) {
        // check that sum is equal in all nodes
        for (size_t i = 1; i < numberOfNodes; i++) {
            Dataframe* df2 = this->waitAndGet(Key("local_sum_i"));
            assert(sum == df2->getInt(0,0));
        }
    }
}
```

where we have a node reading data into the system, then retrieve the data in all nodes and do some computation on each one of them, and then do some merging operation on one of them.

Open Questions

- How to connect multiple computers to run Eau2

Status

Currently working on: - fixing memory leaks - find a way to connect multiple computers over the network - implementing linus - writing tests

Estimated Time of Implementation

Fixing current memory leaks: 5hours

Connect multiple computers over the network: 10hours

Implementing linus: 20+ hours

Writing tests: 6hours

Total time left so far: 40+ hours