Design Document
Team 16
WeatherPipe

By:
Stephen Harrell
Lala Vaishno De
Hanqi Du
Xiaoyang Lin

I. Purpose:

Difficult-to-use data resources or expensive-to-access data have often confined weather researchers to limited data sets restricting their sample sets for testing their hypothesis and theories. The partnership of National Oceanic and Atmospheric Administration (NOAA) with Amazon that would make it openly accessible for anyone to access past radar data could potentially transform the way this problem has been approached thus far. We plan to write a general-purpose historical weather data analyzer that would employ MapReduce to load datasets from Amazon S3 and directly run the required analyses. This application would thereby, be able to create novel historical analyses with dramatically less work and cost than was possible in the past.

Definitions

1. **MapReduce**: A programming model used to process very large datasets across many computers and return a much smaller dataset.
    a. https://en.wikipedia.org/wiki/MapReduce
2. **AWS (Amazon Web Service)**: A web based api that can provision many types of web services including S3 and EMR instances.
    a. https://aws.amazon.com/
3. **S3 (Simple Storage Service)**: A web based object store used with AWS services.
    a. https://aws.amazon.com/s3/
4. **EMR (Elastic Map Reduce)**: Dynamically provisioned Map Reduce clusters in AWS.
    a. https://aws.amazon.com/elasticmapreduce/
5. **NetCDF**: A file format used for array-oriented scientific data.
    a. http://www.unidata.ucar.edu/software/netcdf/
6. **NEXRAD**: A network of high resolution radar stations in the US operated by the National Weather Service.
    a. https://en.wikipedia.org/wiki/NEXRAD
    b. Map of Stations: https://www.roc.noaa.gov/wsr88d/Images/WSR-88DCONUSCoverage1000.jpg
7. **JSON (JavaScript Object Notation)**: A markup language for storing simple data structures such as arrays, dictionaries, integers and strings.
    a. https://en.wikipedia.org/wiki/JSON

Functional Requirements:

1. Analysis is done on NEXRAD data set.
    a. Analysis done on many files at once to be reduced based on user provided analysis
        i. Data set is split up into 5 minute increments for each weather station
    b. Analysis is done with EMR to split up the work between many workers
        i. Users will use their own AWS credentials to access EMR
        ii. The user will provide the analysis and output code for the EMR job
    c. User interface will be a command line tool with a config file
        i. User will define the inputs and outputs for the specific analysis in the config file
        ii. User will define the AWS credentials in a hidden file in their home directory
        iii. User will define the output code and any files the output should go to in the config file
        iv. Generalized inputs to the command line tool will be time inputs and location inputs
            1. Location inputs will be either a radar station name or a shape file
                a. If a shape file is given the code will determine which radar stations are relevant to the shape and analyze all of them.
            2. Time inputs will be given as either a block of time or slices in time
                a. An entire day is an example of a block of time
                b. Every October for the 1970 to 1980 is example of slices in time

II. Design outline:

1. Command Line Tool

    a. Programmatic MapReduce Job Script Generation
        i. Generate Hive script based on config file and analysis code
    b. Job submission to Amazon Elastic MapReduce
        i. Use Config file to define inputs to analysis and analysis/ouput code
        ii. Analysis/output code lives in java file that gets compiled at submit time
        iii. Analysis/Output code will be subclasses of superclasses that handle data marshalling and general EMR bootstrap tasks
    c. Write Results out to Files
        i. Different analyses will have different outputs
            1. Integer - json format
            2. Arrays - json format
            3. Dictionaries - json format
            4. 3D spaces - NETCDF files

2. Different types of analyses

    a. Average of a specific variable at a specific point in space over time
        i. Inputs: Time block, point in space, variable name
        ii. Outputs: Integer
    b. Histogram of specific variable at a specific point over time
        i. Inputs: Time block, point in space, variable name
        ii. Outputs: Array of histogram
    c. Multivariate Histogram
        i. Inputs: Time block, point in space, multiple variable names
        ii. Outputs: Dictionary w/ keys of variable names and values of arrays of histograms
    d. 3D Space average
        i. Inputs: Time block, shape file, variable name
        ii. Outputs: NetCDF file of the average of each point in 3d space over the time

III. Design issues:

Functional Issues:

Issue 1: API Decision

Option 1: use the original AWS-S3 APIs

Option 2: write a series of new WeatherPipe APIs

Decision:

Option 2 is chosen, because sometimes we need to develop other interfaces and do more complex work, including pluggable analyses.

Issue 2: What user interface should be chosen?

Option 1: Command Line Tool with flags

Option 2: Command Line Tool with config files

Option 3: Web interface

Decision:

Option 2 is chosen because the pluggable nature of the analyses require dynamic inputs and outputs. The config file provides an easy way to track this somewhat complex procedure.

Issue 3: Types of data from the data set that can be chosen by the user

Option 1: Allow users to choose a single date

Option 2: Allow users to choose a range of consecutive dates

Option 3: Allow users to choose a selection of dates over several years

Decision:

We agreed on implementing all the options because it would enable the user to use the data set in many more ways increasing the functionality of the system. Option 1 would be be the simplest to implement.

Issue 4: Storing the result of an analysis for future use

Option 1: Do not allow the saving of results

Option 2: Allow saving the results in a simple text-based format

Option 3: Allow saving the results in an archive that would be searched before every analysis to see if it has been computed before

Decision:

Option 2 is chosen because it is permits saving results while not making it unreasonably complex and expensive to store the results in an archive that has

to be searched using a key. Moreover, Option 2 would permit the results of previous analyses to be sent or printed.

Non-Functional Issues:

Issue 5: What format of output shall we return?

Option 1: return integers

Option 2: return arrays of integers

Option 3: return as json format

Option 4: return netcdf

Decision:

Option 3 and 4 are chosen, because such simple data, like integers and arrays of integers cannot represent all the information of the results after the analyses. In addition, different output formats depend on different kinds of analyses.

Issue 6: How should we obtain Users information and credentials for our analyzer?

Option 1: Allow people to register by giving them authority to key in information and credentials.

Option 2: Pull information and credentials directly from myPurdue.

Decision:

Option 2 is chosen, because our main users are atmospheric researchers at Purdue University. It will save their time to use our analyzer, and it makes the information of the user safer.

Issue 7: Handle individual errors within an analysis

Option 1: ignore the errors and produce results from the working parts of the analysis

Option 2: try re-running the individual analysis causing errors

Option 3: terminate the entire analysis and produce error report

Decision:

Option 2 is chosen because it would allow the system to solve the errors without looking for user's help. The individual erroneous analysis will be run 5 times and if it continues to fail, the user will be given the option to either ignore those errors or terminate the process completely.

IV. Design details:

Class design:

1. **UserInterfaceAPI:** This will be an API that provides all functionality for the command line tool. This will allow us to expand to other types of interfaces such as web interfaces if the researchers prefer.

2. **AWSInterface:** This class will handle all interactions with AWS such as creating new jobs, checking credentials, submitting jobs and any statistics from AWS such as cost of job.

3. **AnalysisSuper:** A super class for researchers to use a start for their analysis code. This will handle all of the data grabbing per individual analysis, unzipping and creating accessibility of the specific data. It will also provide an generic interface to transfer data output data to the Output class running on the user's machine.

4. **OutputSuper:** A super class for researchers to marshall all output data and write to one of two different file types. Either a json text file or a NetCDF file.

5. **AnalysisOutputCompiler:** This class will compile the output and analysis subclasses that are written by the researcher.

6. **HiveScriptGenerator:** This class will generate the hive script and will call the DataMarshall class to determine the files that are relevant for the simulation.

7. **PayloadUploader:** This will use the HiveScriptGenerator and AnalysisOutputCompiler to gather the necessary files, put them in the necessary places to launch an EMR job.

8. **DataMarshall:** This class will determine which files are relevant for the simulation based on the time and location inputs.

9. **FileWriter:** The Output classes will use this class to write JSON or NetCDF files.

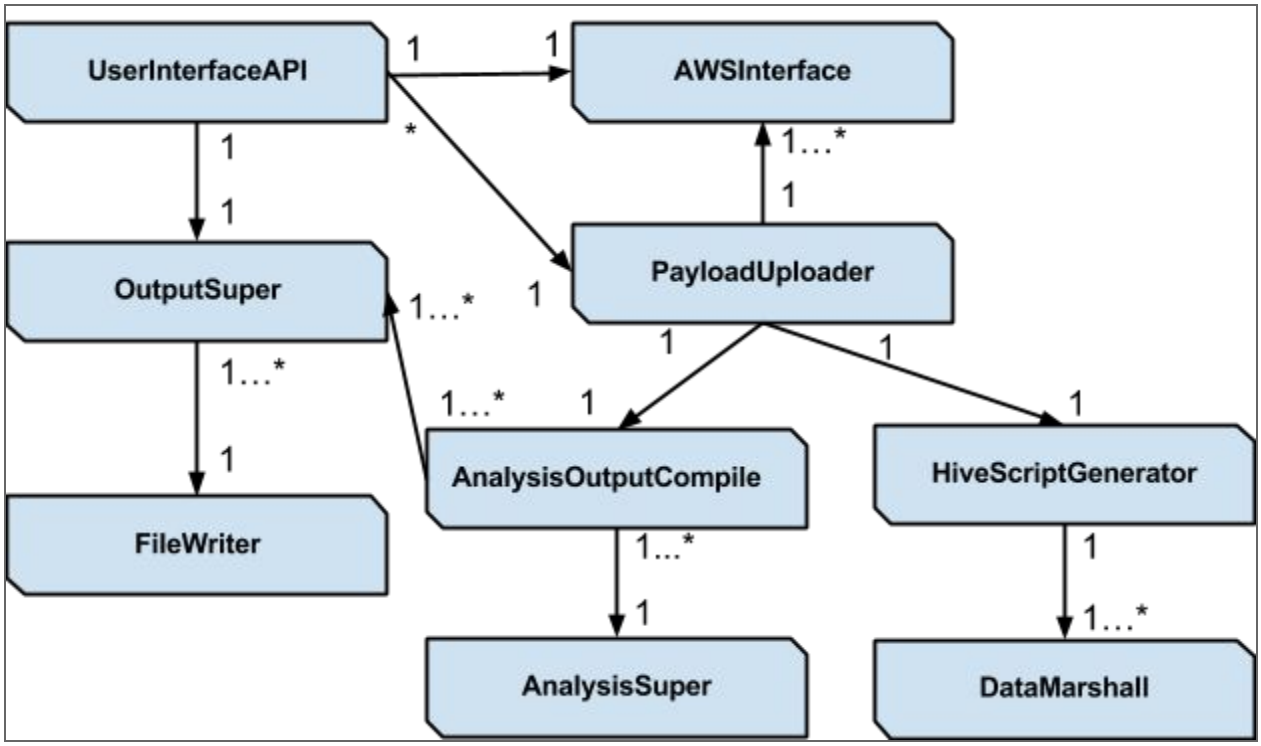

Figure 1.1 : Class Diagram: illustrates the relationship between the different classes and the flow of data and object instances as the pipeline functions.

Sequence diagrams:

1. User-Interaction Diagram

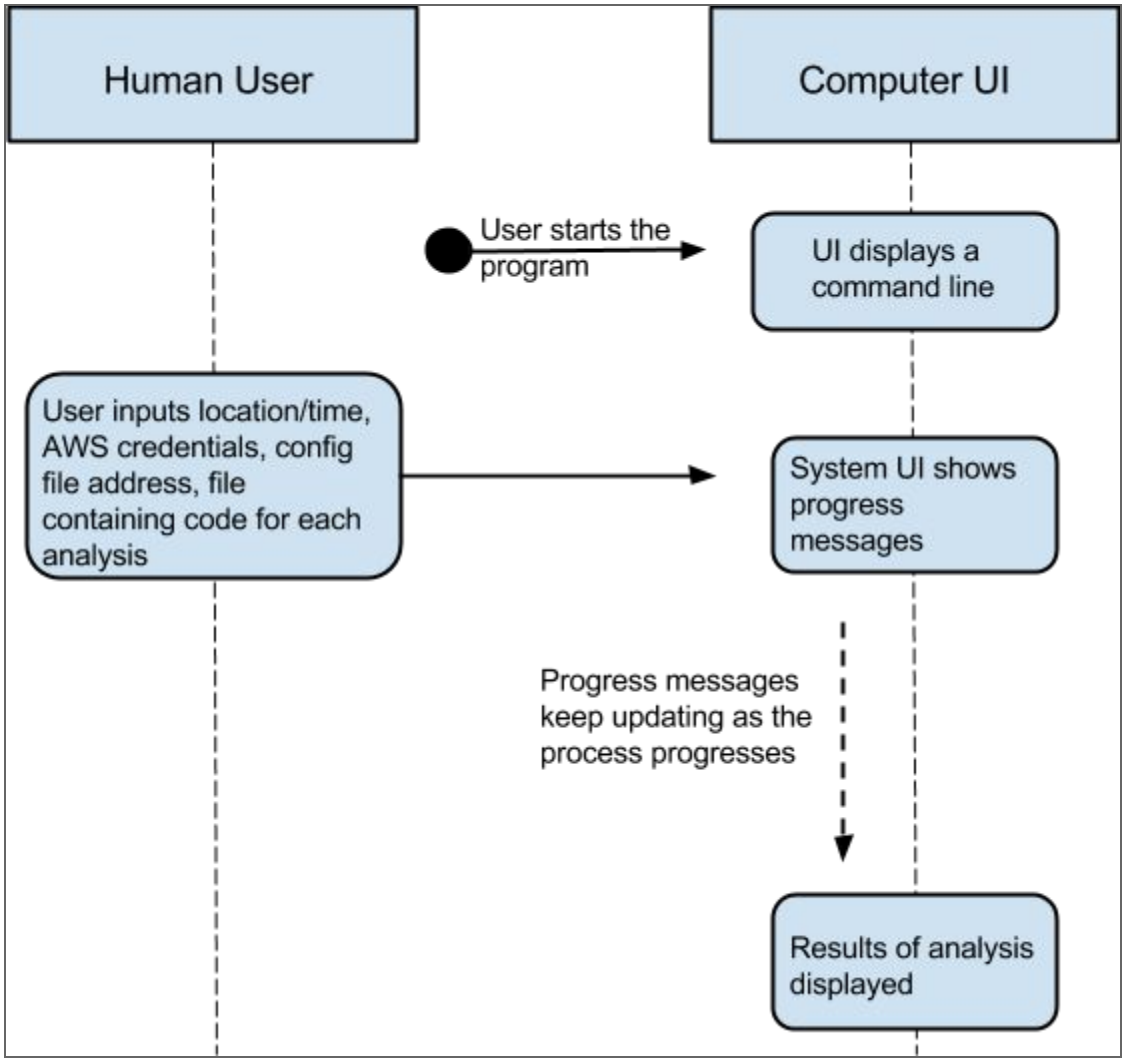

Figure 1.2: illustrates the process of user interaction between the user and the User Interface (UI) from when the program is started to when the results are printed on the screen. Once the program has accepted the inputs, it displays progress messages on the screen that would correspond to the step that the program is currently performing.

2. Data flow diagram

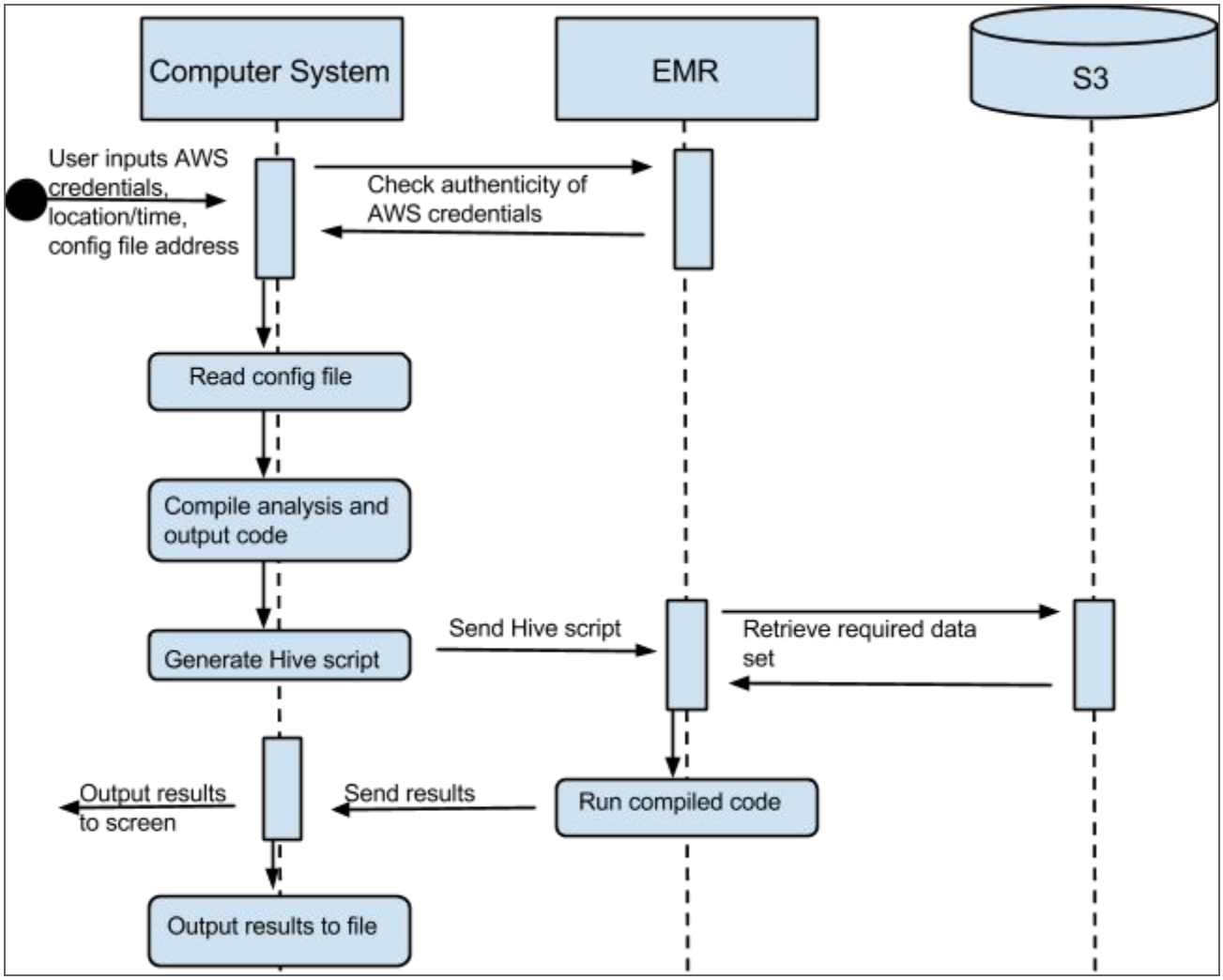

Figure 1.3: illustrates the flow of data inside the program. There are three types of data that flow within the program: 1) Data that is inputted by the user: this consists of the AWS credentials, the location/time and the config file. The config file contains the address of the file that contains the code for the analysis and flags that determine the form of output the user would like the results to be presented as. 2) Data that is retrieved from S3 depending upon the location and time parameters inputted by the user. 3) The data produced as a result of running the analysis on the data retrieved from S3. This data is saved on a file and displayed on the screen depending upon the type of output selected.