Group 3 Project Document

Wallace Donahoe, Joseph Spitaleri, Matthew Marsh
Brandon Carey, Mingzhi Yu

December 3, 2014

Contents

1 Group Web Page Link

http://cisc475-3.cis.udel.edu/

2 Change Log

- Version 0.6

  We have a working version of the Floyd-Warshall algorithm in HAMR, Java. We are currently working on the Spark version of the algorithm. We are also creating minor test cases on the Java verion of the algorithm and then comparing them with the HAMR and Spark versions.

- Version 0.5

  A Testing section was added to describe that aspect of the project.

  Descriptions regarding HAMR and Spark, were revised to emphasize that they are completely unrelated.

- Version 0.4

  We have a complete working prototype of the Floyd-Warshall algorithm written in Java code (not HAMR). Initial have passed and initial results look encouraging.

- Version 0.3

  The modular design of the machine was heavily reworked to account for new information regarding the HAMR and Spark software.

  A prototype system for taking an input and creating a Matrix out of it was created. It should be relatively easy to adapt to HAMR or Spark.

  Test program WordCount is now compiling on our Virtual Machine, as is a prototype of ShortestPath

- Version 0.2

  HAMR has been installed onto our Virtual Machine

3 Introduction

Big Data has become an increasingly important part of our world, and many tools have been developed in an effort to learn from this data and make informed decisions on it. One of the most widely used paradigms is MapReduce, which is based on the common functions *map* and *reduce* from functional programming. However, MapReduce goes beyond this, parsing out operations on to each node in the network, where the advantages of parallelization make data processing much quicker. While this paradigm has been wildly successful, it is now being pushed beyond its limits of usefulness.

HAMR is being developed in an effort to overcome many of the limitations common to MapReduce implementations, including relying only on batch processing and complexities in user-interaction. Our project aims to test an implementation of the Floyd-Warshall algorithm in HAMR and other Big Data tools in an effort to evaluate the efficiency of HAMR in large graph problems in comparison to competitors.

4 Definition of Concepts

- **HAMR** is a distributed big data tool written in java, developed by ET International capable of pulling data from multiple sources simultaneously, while processing in batch or real-time streaming modes. Note: It is merely a framework to work in. The implementation is still up to us, but we will use this to help distribute our algorithm amongst multiple PCs for processing.
- Next four items are other sources that will be used along with HAMR.
- **Hadoop** is an open-source software framework for storage and large-scale processing of data-sets on clusters of commodity hardware. HAMR utilizes portions of Hadoop in the Resource Reader packages.
- **Spark** is an open-source system for distributing the execution of a program amongst many computers at once, similar to HAMR.
- **Floyd-Warshall Algorithm** is a graph-analysis algorithm that finds the shortest path from each vertex to every other vertex in a weighted graph.

```
1       FloydWarshall(V,E)
2         for each vertex v in V
3           dist[v][v] = 0
4         end for
5         for each edge (u,v) in E
6           dist[u][v] = w(u,v)
7         end for
8         for k from 1 to size(V)
9           for i from 1 to size(V)
10            for j from 1 to size(V)
11              if dist[i][j] > dist[i][k] + dist[k][j]
12                dist[i][j] = dist[i][k] + dist[k][j]
13              end if
14            end for
15          end for
16        end for
17        return(dist)
18      end
```

5 Stakeholders

- ET International: Needs to evaluate the HAMR system in order to compare with competitors and evaluate the strengths and weaknesses of the system.
- Group Members: Want to be challenged by a project that is in-line with their interests and that furthers their career goals.
- ET International's clients: Need the best possible Big Data solution, and our project ties into the larger goal of making HAMR the best system possible for Big Data problems. "Best", in this case, is defined by the software which most effectively balances ease of use with performance and flexibility.
- Professor Siegel would like to see this project be successful in order to help sell future clients on outsourcing problems to students taking CISC 475.

6 Scope of the Project

Our machine will run tests on the Floyd-Warshall algorithm by using several big data frameworks including HAMR and Spark on a distributed system (which will include at least 4 virtual machines on our course server, provided by EECIS department). At the current stage, we are also prepared to run tests on our own computers through a small WLAN network connected by a router. This will be a backup plan in case the virtual machines are not a viable option.

7 Problem Domain

- Our problem domain consists of multiple components, one of which is a weighted undirected graph. Graph, in this case, refers to the standard mathematical definition of G = (V,E), where V is a set of vertices and E is a set of Edges. Weighted means that each edge of the graph has a numerical value, this value is considered the length of a route, the capacity of a line, the energy required to move between locations along a route, or whatever is being represented by the graphical layout. Since the graph is undirected, it means that the edges have no orientation. The edges go both ways between two vertices, they are bidirectional. This graph will be presented as a text file, which will be fed into HAMR. The text file will display the graph as a series of triples in the form (node, weight, node); these are the representation of the graphs edges. The triple will inform the system which two nodes are being attached by the edge, and the weight of that edge. As a result, the list of vertices will be implied by the list of edges. It is assumed that none of the vertices will have 0 edges connected to it. In order to interface with the framework that ETI implemented for HAMR, the HAMR API will be greatly used. The HAMR API consists of 20 different packages and 123 different classes. Several classes that we will implement for this project consist of ResourceReader, Edge, Node and Job.

  ResourceReader is a specialized edge Flowlet for reading from external data sources. This class uses a modular framework to provide flexibility in choosing how the data from the external data source is formatted as it is read into HAMR. It has two parameters, K-the key type and V-the value type. The framework for ResourceReader consists of

  RecordReader provides the factory and services for creating the objects that perform conversion of data from the data exchange format into key value pairs.

  PartitionRecordReader performs the conversion of data from either ByteBuffer or Tuple data exchange format into key value pairs.

  ReaderProvider provides the factory and services for creating the objects that read data from the external data source and convert it into the data exchange format.

  PartitionReader reads data from the external data source and converts it into the data exchange format, either ByteBuffer or Tuple.

- All of these are other classes that are in the HAMR API.

The methods for ResourceReader consist of int getConcurrentTaskLimit(), IPushProducer¡K,V¿ out(), void resetConcurrentTaskLimit() and void setConcurrentTaskLimit(intconcurrentTaskLimit).

The Edge class is an Object that represents a directed edge between two IBindable instances. (IBindable serves as a common interface which must be implemented by all bindable instances. This interface is implemented in its entirety by Bindable, which is the common base for all concrete bindables in HAMR. It is extended, however, by various functional interfaces which provide additional capabilities for push and pull operations). The methods in Edge are GroupNode getLocalDomain(),IBindable getTarget(),GroupNode getTargetDomain() and boolean isLocal().

The Node class is an Object that is the common superclass for all first-order graph elements, including Flowlet and the Job, and defines several common attributes, runtime hooks, and operations common to all of them. Out of the many methods in the Node class, the methods Aggregator getAggregator(),GroupNode getDomain(),Job getJob(),int getPartitionCount() and Partitioner getPartitioner().

Job is a container for a single complete workflow. This means that a job represents a single HAMR workflow, and serves as the ultimate root element for all other Nodes and Flowlets comprising that workflow. The methods for the Job class are Boolean containsCycles(), String getName(),int getNodeCount(),HAMR getRuntime(),int getThreadCount().

Another element of the problem domain is the cluster, which we will be using to test the distributive power of the two frameworks. We will be a 4 node cluster, granted to us by the University of Delaware for the purpose of this project.

8 Client Requirements

- **Test HAMR Framework** - An implementation of the Floyd-Warshall algorithm must be coded, using the HAMR framework. This means that it will utilize HAMR-exclusive packages and classes. This will require its own unique implementation, as use of these packages precludes a modular ability to swap frameworks in and out.

- **Compare results using Spark** - An implementation of Floyd-Warshall must also be written using Spark. This will be used as a basis for comparison. As with HAMR, use of Spark-exclusive packages means that Spark requires its own implementation.

- **Post-mortem Report** - At the end of our project, the client requests a detailed report about the results. This report must include, but may not necessarily be limited to, commentary regarding the ease of use of the HAMR framework, the usefulness of the online tutorials on HAMR, and performance metrics like overall runtime and CPU Usage.

8.1 System Design

- All coding will be done in Java. This is because HAMR is written in Java, so compatibility will not be an issue.
- The input will be a list of edges with positive weights (between 0 and 1), formatted as two node names, and followed by the weight of the edge between them. For example an edge from vertex 7 to vertex 42 with weight 0.5 could look like "7 0.5 42". It will be a simple .txt file.
- The output file will also be a .txt file, in the format of an NxN matrix of vertices and their minimal distances to every other vertex. This matrix will be represented by a 2D array in data. For example, if you wish to figure out the shortest path between nodes 2 and 5, then looking up "answer[2][5]" or "answer[5][2]" should produce the correct path weight. (This matrix should be diagonally symmetric.
- All other implementation details that are not specified above are left solely to the discretion of our team. The client may or may not choose to impart advice regarding implementation details, but nothing more will be required beyond that which is specified above.

9 Testing

As discussed in the Client Requirements section, our primary responsibility is to test the HAMR framework using our implementations of the Floyd-Warshall algorithm. To that end, the client has provided us with a file called "weighted10" representing a graph with 1024 nodes and a large quantity of edges between those nodes. There are three implementations being tested.

- Java: A sequential implementation of Floyd-Warshall using the accuracy of the below two distributed Floyd-Warshall implementations.
- HAMR: A distributive version of the algorithm written in Java, using the packages in the HAMR framework.
- Spark: A distributive version of the algorithm written in Java, using the packages in the Spark framework.

Further, we are testing each implementation on the following criteria.

- Performance Time: The time it takes to fully execute the algorithm from start to finish.
- CPU Usage: The difference between baseline CPU Usage of the tested machines and the CPU Usage with the algorithm running. This will be averaged out to account for multiple machines in a distributive network.
- Ease of Implementation: Our subjective analysis on how easy or difficult implementing each version of Floyd-Warshall was. Unlike the other two criteria, this is not quantifiable, and subject to our personal biases.

Our final submission will be a written document with the information specified above for each implementation. The client will use this information to further refine HAMR.

10 Basic Design

- **Moduler**: Each of our two test cases (HAMR and Spark implementations) will be written separately. However, the two systems will be written modularly, breaking up the system based on what is likely to be the most complex and thus most subject to change. In theory, this will make coding the second test case significantly easier because the code will be easy to rewrite. However, it must be noted that the Java, HAMR, and Spark implementation are **completely separate**, and unrelated to each other.
- **System**: Distributive System. In order to properly test a distributive framework, it is required
- **User cases**: User cases won't be applied to this program.
- **Data flow**: All data will run on framework directly through the distributed system. Therefore, the data flow is dependent on the framework partition data.
- **Test file**: It is a weighted, undirected graph. Each edge is represented as a triple. For example:
  0 0.170 1
  1 0.401 2
  2 0.513 3
  With source vertex, weight, and destination vertex.
- As the machine is performing all of these tasks, the runtime and overall performance must be recorded by a separate module.

11 Implementation Details

11.1 Java Implementation

Because the sequential implementation of Floyd-Warshall is relatively simple, there are two Java classes in the code that implements it.

- **Matrix** is the class which takes in the file (desribed in the Requirements section) and converts the input into a usable java objects. There are three objects in this class.

  inputLine: A string[], representing the line currently being processed by the parsing engine. This object will be one single object, repeatedly changed in order to reduce the memory and processing requirements by saving on the costs of instantiating multiple objects.

  Edge e: An Edge is an object representing two nodes (as integers representing their ID numbers) and a double representing the weight of the edge between them. The specific Edge e object represents the edge that is represented by the line being processed (inputLine). There is only one single edge, being repeatedly rewritten over and over again with each iteration. This is to preserve processing power and memory that would be wasted on generating multiple objects.

HashMap¡Integer, HashMap¡Integer, Double¿¿ edgeMatrix: This is the object which all edges will be written to. The first Integer key represents the first Node in the Edge object. The second Integer key represents the second Node in the Edge. The Value Double represents the weight of the edge between them. This structure is put in place for it's fast lookup times and ability to handle potentially large data sets.

In addition, the Java packages Buffered Reader, File, and FileReader were imported to assist with the parsing of the input file.

- **FloydWarshall** uses the edgeMatrix object created by the Matrix class, and performs the FloydWarshall algorithm the data. There are a few objects in the FloydWarshall class.

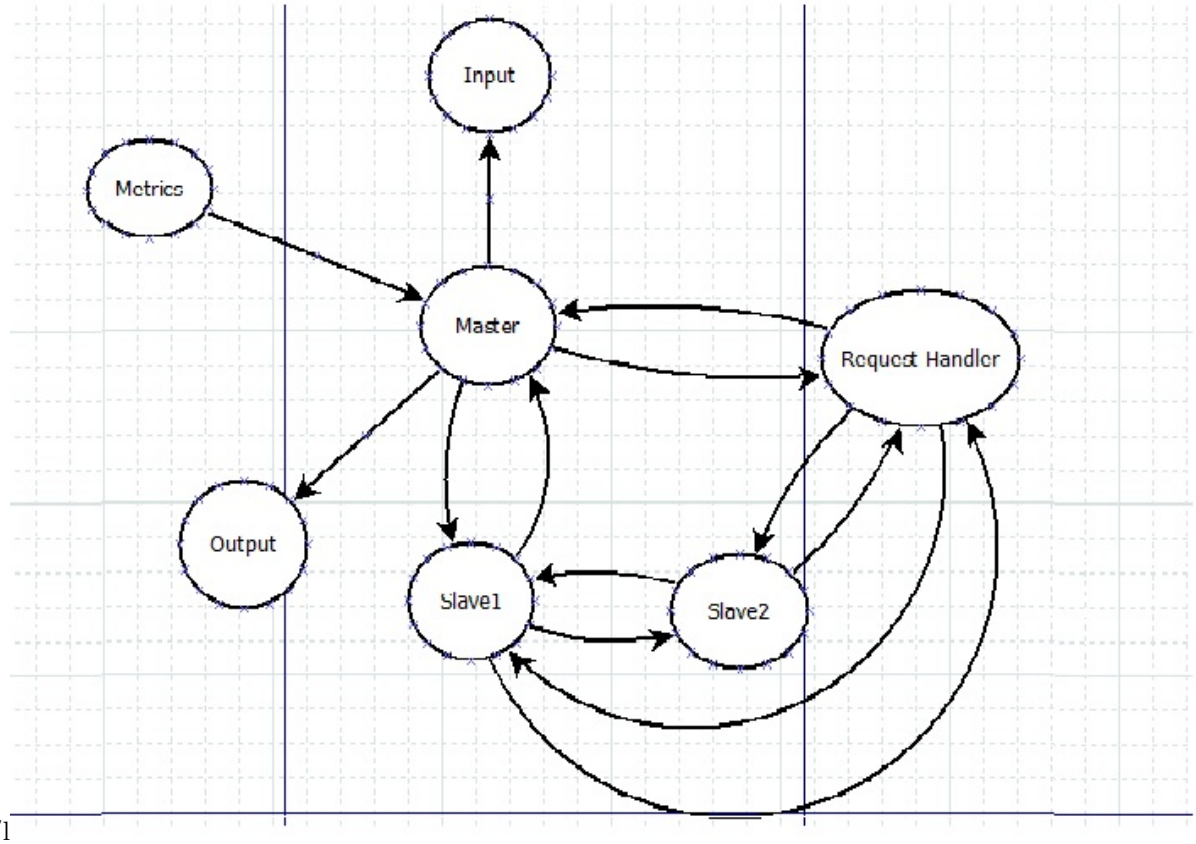
  INFINITY: This is a constant that will be represent a path which either does not exists or is unknown. It is equal to the greatest number that can be stored in a Double in Java.

  distanceMatrix[][]: This is the NxN matrix upon which the sequential Floyd-Warshall is performed on. Before any processing can occur, the data from edgeMatrix must be transferred over to this object, filling in gaps with the INFINITY constant. As the algorithm is performed, this matrix will be repeatedly updated to show the results of the processing. Ultimately, it will represent the results of the completed sequential Floyd-Warshall algorithm.

  numberofvertices: This is a fairly self-explanatory Integer. It exists mostly for the use of instantiating the distanceMatrix and printing out the results at the end of the algorithm.

11.2 HAMR Implementation

11.3 Spark Implementation



/;/l

Figure 1: A use-case diagram of the modular design.

12 Relative Concept

To give a picture of how different the results will be, we introduce the fundamental partition mechanism of HAMR, GraphX, and MapReduce.

- **HAMR and GraphX**: These two frameworks have similar partitions according to the key value (HAMR calls this part KVS and GraphX called it RDD). They search for the particular key in parallel. Different partitions will send messages for communication purposes through the distributed system.

- **MapReduce**: MapReduce is a programming model and an associated implementation for processing and generating large data sets with a parallel, distributed algorithm on a cluster. Below is the data flow diagram:

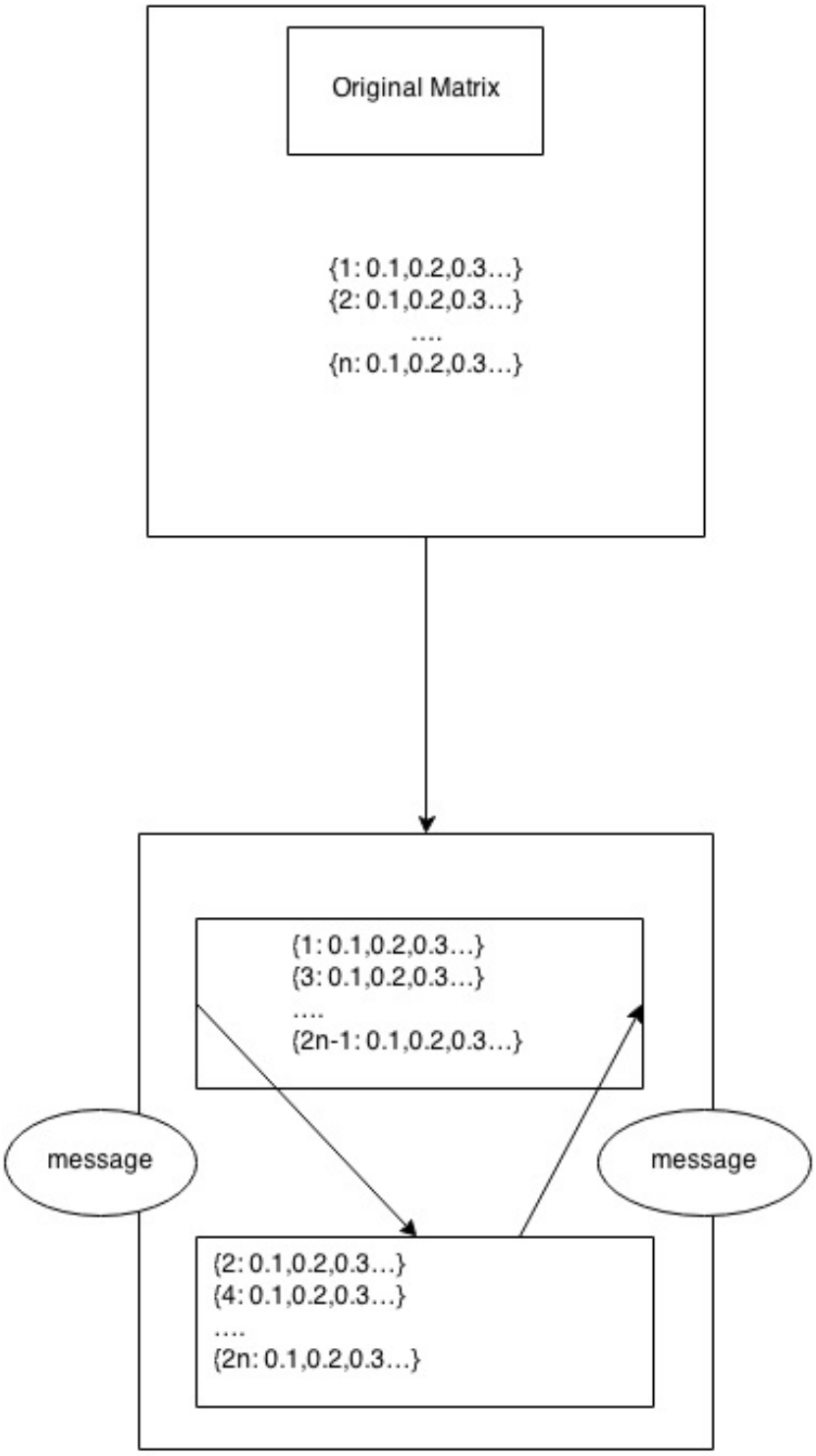


Figure 2: This simple diagram shows how the partition works.

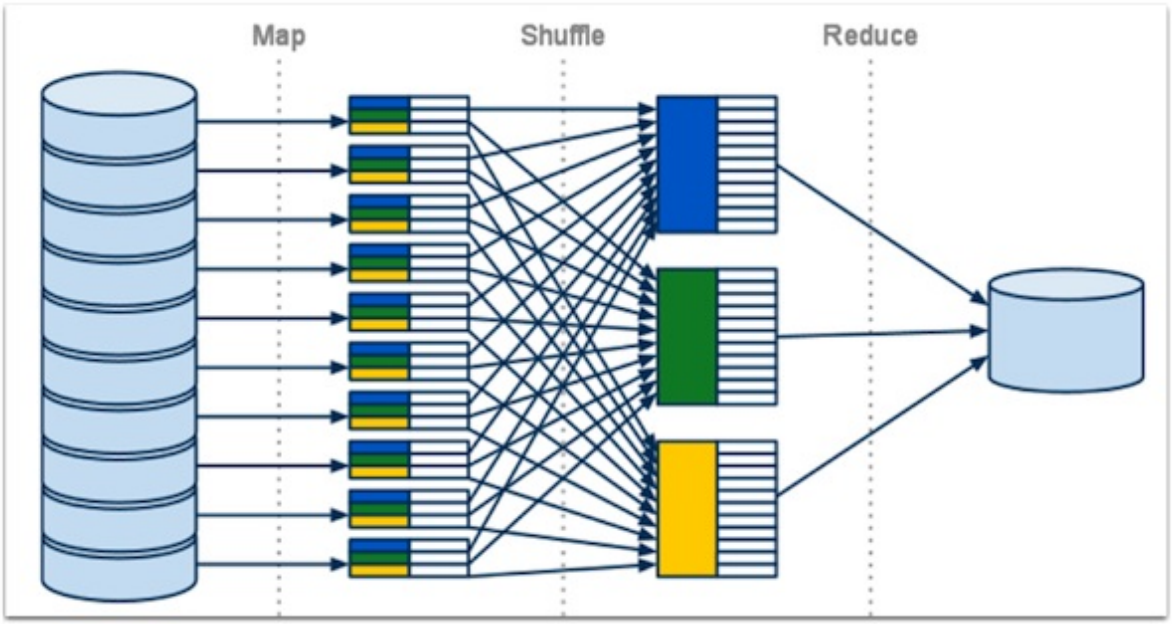


Figure 3: This simple diagram shows how the partition works.

13 Goals

- The primary goal of this project is to test the ease of use of ET International's proprietary HAMR software with other leading distributive processing frameworks.
- The Floyd-Warshall algorithm is being used for its relative ease in terms of understanding and implementation. As such, the actual implementation of this algorithm is a secondary concern to the testing of HAMR.

14 Glossary

- **HAMR**: HAMR is the only Big Data analytics engine capable of pulling data from multiple sources simultaneously, while processing in batch or real-time streaming modes.
- **ETI**: ETI's world-class team of experts will conduct a situation analysis of your organization. A report will evaluate how effectively your strategies align with your current Big Data systems.
- **ASPS**: The Floyd-Warshall algorithm is being used for its relative ease in terms of understanding and implementation. As such, the actual implementation of this algorithm is a secondary concern to the testing of HAMR.
- **Spark**: A Resilient Distributed Graph System.
- **Hadoop**:Apache Hadoop is an open-source software framework for storage and large- scale processing of data-sets on clusters of commodity hardware.
- **MapReduce**:MapReduce is a programming model and an associated implementation for processing and generating large data sets with a parallel, distributed algorithm on a cluster.
- **RDD**:A Resilient Distributed Dataset (RDD), is the basic abstraction in Spark. It represents an immutable, partitioned collection of elements that can be operated on in parallel.
- **KVS**:Key/Value Stores. Flow nodes that store and retrieve KV pairs. Can read and write.
- **Graph**: In mathematics, a graph G is represented as G = (V, E), where V is the set containing every vertice in the graph and E is the set contain every edge on the graph. And edge connects two nodes together.

  An undirected, weighted graph is a special kind of graph. A graph that is undirected means that all edges are bidirectional, and can be traversed in either direction. A graph that is weighted means that edges can have a variable weight. Each edge will have its own weight.