CS6240 Project Report

Pankaj Tripathi, Kartik Mahaley, Shakti Patro, Chen Bai

April 22, 2016

The report encompasses the details about the API, design choice, implementation, study of performance, installation guideline along with usage and future scope/enhancements.

Introduction:

The goal of this project is to implement a MapReduce framework with a streamlined API from first principles. A MapReduce API that can be used on the lines with the existing APIs provided by the Hadoop. A developer can simply use the API implemented for this project and use it develop his/her own MapReduce application with the minimal focus on the specifics of how the application would run on EC2 machines.

About API:

This project is a basic implementation of the map-reduce algorithm in Java for a pedagogical illustration of the programming model. Our API has adapted some of the interesting features of the Map Reduce API provided by Sylvain Hallé, Professor in the Department of Computer Science and Mathematics at the Université du Québec à Chicoutimi.
This project aims at providing a simple framework to create and test map-reduce jobs using a minimal setup (actually no setup at all), using straightforward implementations of all necessary concepts.
We have implemented some of the programs using the MapReduce we have developed to illustrate, how our API can be used for developing MapReduce applications.

Design:

The design that we have chosen to implement the task involves three phases furnished below. The three phases mentioned below follow the first principles for implementation.
Phase-1: Creating and running EC2 instances
Phase-2: Read data and develop Mapper
Phase-3: Shuffling and sorting
Phase-4: Develop Reducer
A detailed description of these phases and design is provided in the implementation section.

Implementation:

Phase-1: Creating and running EC2 instances

In this phase, we create EC2 instances. EC2 are described based on availability zone, Amazon Machine Image, instance type. A Java program creates of the security group and add instances to the group. We create an EC2 master node and slave nodes. The master node takes care of file distribution. The master node and the slave nodes get the JAR MapReduce application developed by any developer. We have a dedicated JAVA code which deploys the JAR file to each slave as well as the master. This code starts the EC2 instances, deploys the JAR file and then executes the file. The communication between the EC2 instances is achieved

using Sockets.Once instances are up and running their machine count, instance ID,public IP and public DNS name are written to an s3 bucket with the file name as "ipaddress.txt". Data from this file is then used by successive phases to establish a server socket architecture. We have implemeted loging feature of Apache to print logs for the operations.

Phase-2: Read data and develop Mapper

The master node reads the input folder content. After having the folder content it divides the files equally among the slave nodes. These files are read by each slave nodes which then provides these records to the Mapper. We have developed a Mapper class with map method. This method takes the records from the files as {KEY,VALUE} pair. The key here is line number and the value is the line string. The map method can be overridden by the developer to get these file data and use the same to write his/her own logic. The map method has two parameters one is the Tuple and OutCollector where Tuple is a class with fields KEY and VALUE. These fields represent the KEYs and VALUEs that mapper gets from the initial reading of the files. OutCollector is the data source used as the output of the map and reducer phases. An OutCollector can be used to store data tuples using the collect method.

Phase-3: Shuffling and sorting

To explain distributed sort lets us assume 4 EC2 machine which 1 master-3 nodes configuration.
n= number of nodes, N= Number of records at each node, w=N/n
Sort Local Data :
Each node is assigned a list of files and each record is converted to an object . Each list of file is disjoint hence nodes can sort their local data in parallel. Each node will select data from indices w, 2w+1, 3w+1 ... This list of data from each node is sent to master.
Find Global Pivots and Partitons:
Master gathers and sorts the local regular samples. n-1 pivots are selected from the sorted regular sample, at indices n + r; 2n + r; 3n + r; ... ; (n - 1) + r where r=n/2 . Each processor receives a copy of the pivots and forms n partitions from their sorted local blocks. A partition is contiguous internally and is disjoint from the other partitions.
Exchange Partition:
In parallel, each node i keeps the i th partition for itself and assigns the j th partition to the j th node For example, node 1 receives partition 1 from all of the nodes Therefore, each node keeps one partition, and reassigns n - 1 partitions.
Merge Partition:
Each node, in parallel, merges its n partitions into a single list that is disjoint from the merged lists of the other nodes. The concatenation of all the lists is the final sorted list.

Phase-4: Develop Reducer

The data from the prior phase is available for the reducer to compute or execute a logic written by the developer using our API. The Reducer class has reduced method which takes three parameters OutCollector, KEY, InCollector. Here the OutCollector will be used to write output tuples, The key associated with this instance of reducer, and An InCollector containing all the tuples generated in the map phase for the given key.

Package Details:

com.mr.net

This package contains the classes for master and slave node scoket communication.

com.mr.io

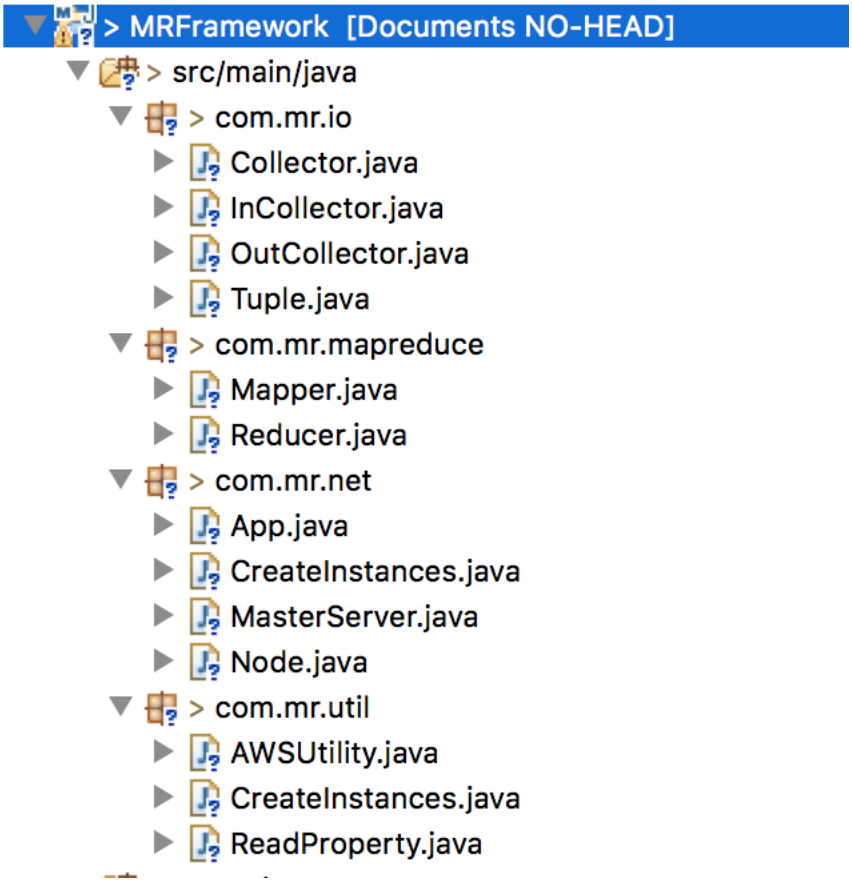
This package contains the InCollector, OutCollector and Tuple class. These classes are implemented for handling the input and output operations for Mapper and Reducer. Tuple class is used to store the data.

com.mr.mapreduce

This package contains Mapper and Reducer class. These classes have map and reduce method respectively which can be overriden by the developer in order to use our API.

com.mr.util

This package consists of miscellaneous utility methods.



API Features:

1. It runs out of the box, simply add the classes (or the jar) to your classpath.

2. Inheritance is fully supported when declaring the types for tuple keys and values. This means that a mapper working with tuples of type (K,V) will properly accept a tuple of type (K',V') if K' is a descendant of K and V' is a descendant of V. This does not work in Hadoop.

Sample Execution:

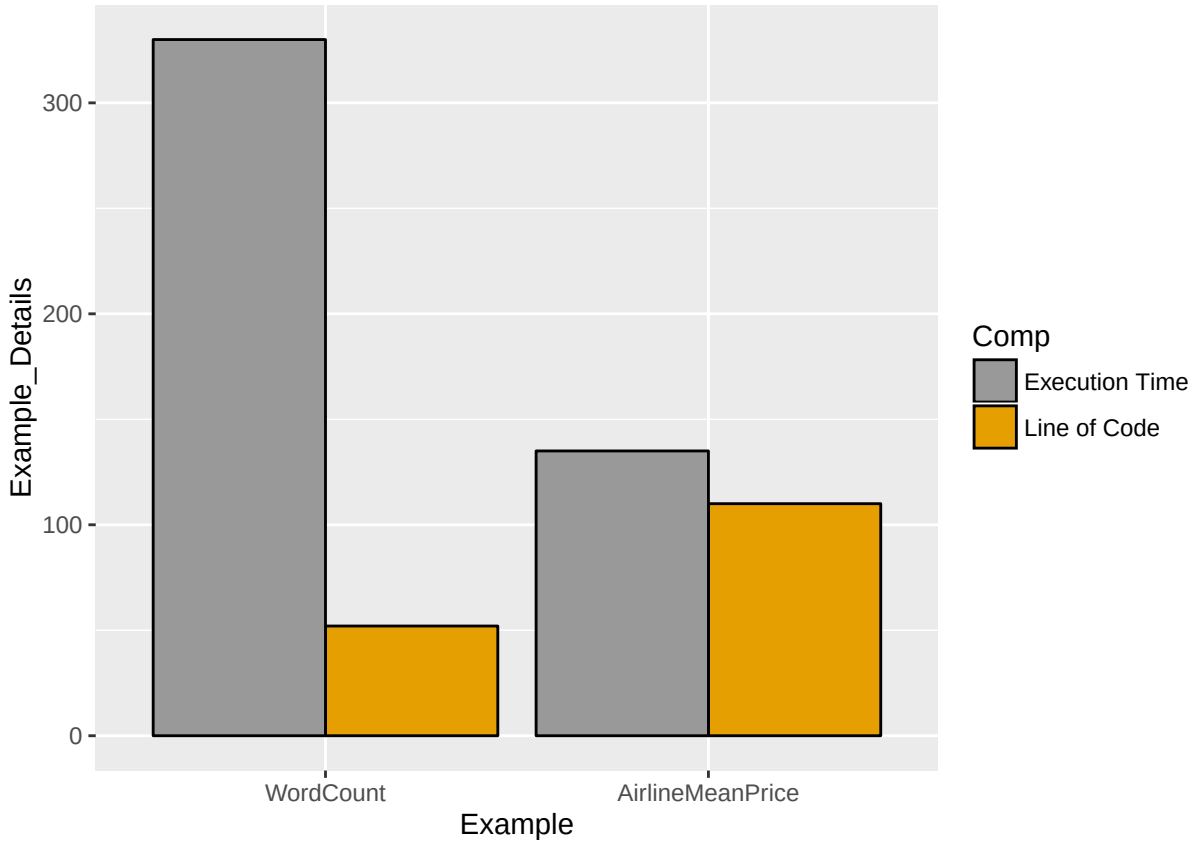
We executed some applications using our API on the EC2 instances the details for the same are provided below. The time provided includes time for starting the instances, uploading the JAR file and running the JAR. For Airline we executed for small files.

WordCount:
Master: 1
Slave: 3
Type: t2.nano
Total file size: 2MB
Time Seconds: 330

Airline Mean Price:
Master: 1
Slave: 3
Type: m4.large
Total file size: 349KB
Time Seconds: 115

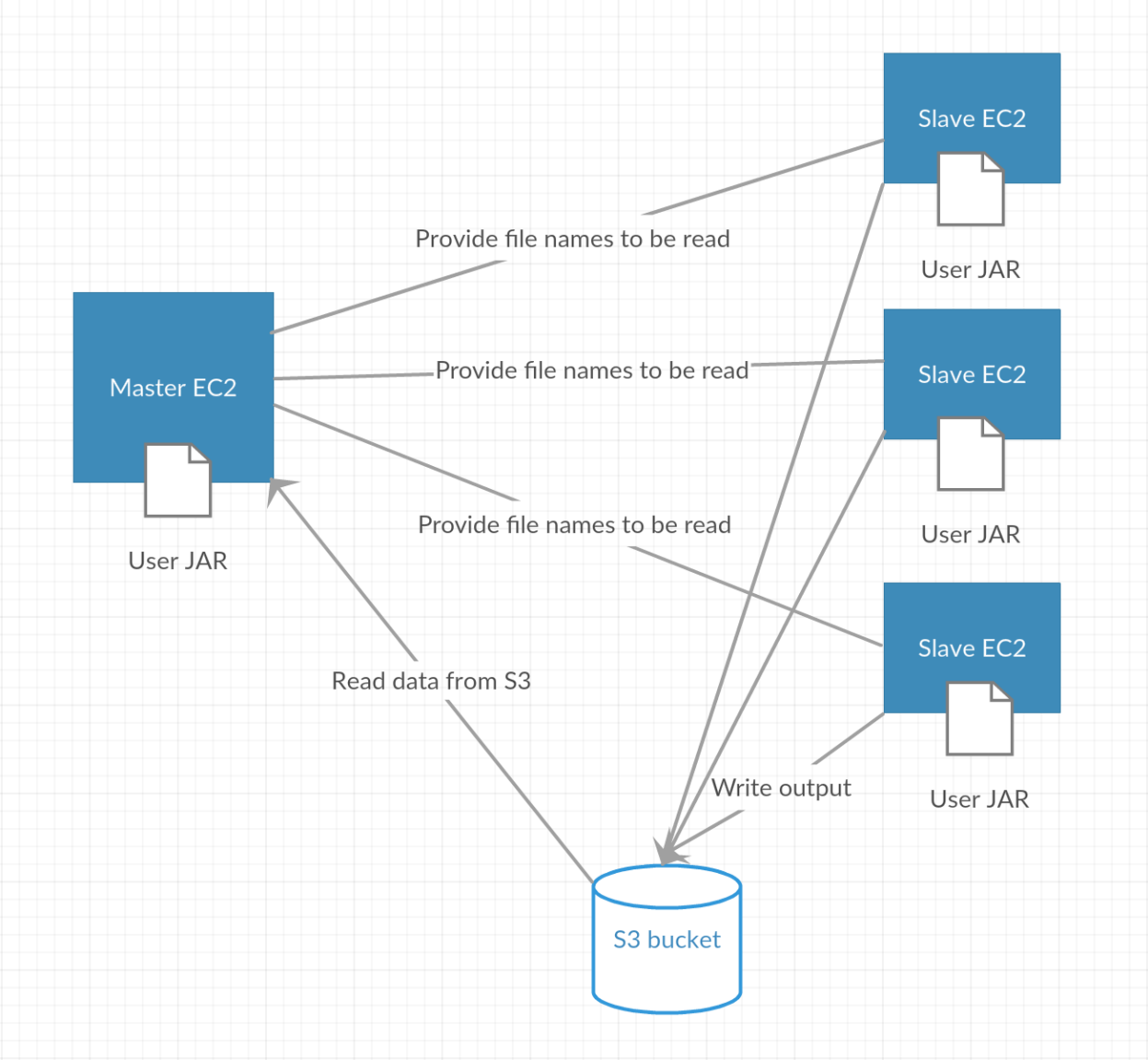
Graph:

Example	Line of Code	Execution Time
WordCount	52	330
Airline Mean Price	110	135

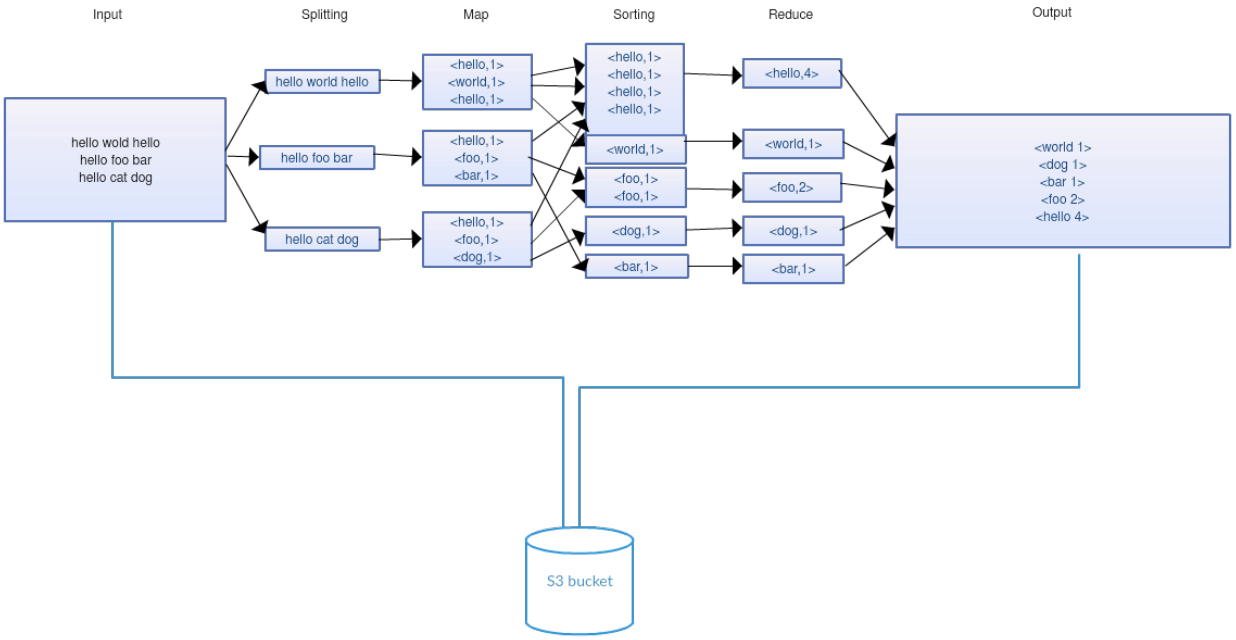


Flow Diagram:

Cluster Flow with user's jar:



Map Reduce flow:



Task Distribution:

Phase-1: Kartik Mahaley
Phase-2: Pankaj Tripathi
Phase-3: Shakti Patro
Phase-4: Shakti Patro
Packaging: Kartik Mahaley, Chen Bai, Pankaj Tripathi
Code Maintainence: Chen Bai
Report: Pankaj Tripathi

Usage:

To use the API simply add the JAR to class path or use the build tools like Maven and push the JAR to maven repository and us the given dependency.

```
<dependency>
<groupId>com.google.code</groupId>
<artifactId>mrframework</artifactId>
<version>1.0.0</version>
</dependency>
```

Challenges:

1. Communication between EC2 machines which caused errors due to network related issues like connection refused, connection reset.
2. Debugging and testing of the client server socket implementation which required starting and stopping EC2 machines.
3. Sending global pivots data from master to nodes during which master and nodes were in wait state.
4. Sorting and distributing files from master to nodes.
5. Collecting and passing the output between the Mapping, Sorting and Reducing phase.

Future Scope:

1. One of the features that we are thinking about adding to our API is fault tolerance. This will help us track the nodes which are alive or dead. If there is any node which is not responding then the another node should start.
2. The current API is feasible only for EC2 machines but we want it to have feature wherein the developer can use the API for any machine.
3. We want to enable API support for HDFS.
4. Handling compressed files for analysis. Currently the API can read data of any format but it should be able to read compressed file as well like GZIP file.

Reference:

1. On the Versatility of Parallel Sorting by Regular Sampling http://web.cs.dal.ca/~arc/teaching/CSci6702/2013/Assignment2/SampleSort.pdf
2. MrSim is a basic implementation of the map-reduce algorithm in Java. https://github.com/sylvainhalle/MrSim