Adaptive Scheduling in Spark

by

Rohan Mahajan

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Computer Science and Engineering

at the Massachusetts Institute of Technology

June 2016

Author . . . . . .
Department of Electrical Engineering and Computer Science
June 4, 2016

Certified by . . . . . .
Prof. Matei Zaharia
Thesis Supervisor

Accepted by . . . . . .
Dr. Christopher J. Terman
Chairman, Masters of Engineering Thesis Committee

Adaptive Scheduling in Spark

by

Rohan Mahajan

Submitted to the Department of Electrical Engineering and Computer Science
on June 4, 2016, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Computer Science and Engineering

Abstract

Because most data processing systems are distributed in nature, data must be transferred between these machines. Currently, Spark data processing systems predetermines the strategies for how this data is to be shuffled before the job has started, but in certain situations, perfomance may be improved by not performing the typical strategy. We add functionality to track metrics about the data during the job and adapt our strategy during the job. We use this strategy to improve regular shuffle performance, joins using Spark's RDD interface, and joins in Spark SQl.

Acknowledgments

First, I would like to thank my parents Umesh Mahajan and Manjulan for their enduring support and love throughout my time at MIT.

I would like to thank Professor Matei Zaharia for his guidance and support while advising me throughout this project.

At MIT, my work would never have been completed if not for the support of my friends. I would like to thank them for all the lessons that I have learned from them, the questions that they answered, and all of the memories that I have created. I would also like to give special thanks to James Thomas who answered numerous questions throughout the project.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Distributed Systems

With the rise of big data, new data processing systems have been designed to help process it. Instead of relying on using just onemore powerful computers, these systems use many computers and are thus distributed in nature due to economic costs,scalability, and fault tolerance. Because programming in these distributed environments is challenging, data processing systems try to abstract this from the user and provide a simple interface for them. One of the most popular systems is Map Reduce, invented by Google, which provides a simple map and reduce operation to the user. Another of these systems is Spark, which provides a slightly more expressive api then map reduce and also has different modes of fault tolerance than spark along with caching. One key and slow stage that both of these systems have, is that because they are distributed in nature, they have a stage where data must be transferred between machines, called the shuffle stage. This shuffle stage can be the bottleneck and be the reason for low performance of jobs.

1.2 Shuffle

1.2.1 Shuffle Introduction

In map reduce, data is loaded onto different computers and an computation is performed on it (the map phase) that results in a group of key value pairs. The final phase of map reduce,the reduce phase assumes that all key-value pairs with the same key are grouped onto the same machine. Thus, an intermediate phase that these systems handle themselves is the shuffle phase where data is transferred so that all key value pairs with the same keys result to be on the same machine.

The following example in Figure 1-1 details the inner workings of what happens in a shuffle for map reducer. Suppose we want to count the number of letters in a distributed file. The mappers will count the number of letters in their individual file. However, we need to aggregate this and thus all the counts for letter a will be sent to worker1, letter b will be sent to worker 2, letter c will be sent to worker c. These reducers will then promptly aggregate the counts that they receive from the mappers.

Because of the huge amounts of keys, these systems do not deal with on the granurality of keys. nstead, they deal with the concepts of partitions. These systems have different partitioning functions that map key-value pairs to different partitions. Some popular partition schemes include range and hash partitioning. As long as all the mappers agree to partition their data in the same way and send each partition that is the same to the same reducer, we are ensured that any two keys that are the same will be in the same partition.

1.2.2 Shuffle Analysis

Throughout the shuffle process, we want to balance the amount of data being sent to the reducers. These systems are constrained by the slowest worker, so generally we want to minimize the latencies of the slowest worker. By balancing the data being sent, we can first reduce the latency for network transfer. Second, because the data each node has to process is more balanced, we can reduce the execution time for the slowest node.

1.3 Overview of EEG Analysis

A seizure is a transient aberration in the brain's electrical activity. People with the central nervous system disorder epilepsy suffer from recurrent seizures, often happening suddenly and at unpredictable times. A seizure can vary from a lapse of attention to a whole-body convulsion. Frequent seizures are dangerous, as they can increase risk of sustaining physical injuries and can even result in death [7].

One method for detecting the onset of epileptic seizures is the analysis of scalp EEG data, a non-invasive measure of the brain's electrical activity. Continuous EEG (cEEG) data is typically recorded using 19 silver/silver chloride electrodes, affixed to the scalp [8]. Figure 1-2 shows a drawing of the placement of sensors on a patient's scalp.

Trained individuals, such as attending physicians, epilepsy/neurophysiology fellows, or registered EEG technicians (encephalographers), review and screen EEG recordings, which typically take place over a continuous 24-hour period [2]. Unlike traditional epilepsy monitoring units which focus on provoking and capturing seizures, the goal of cEEG studies is to efficiently identify future seizures and prevent them. This leads to an increase in the number of cEEG recordings for preventative measures. Intensive care unit centers are subsequently overwhelmed with the analysis of the growing dataset due to the small number of available trained individuals. Methods to screen long EEG recordings without sacrificing accuracy are necessary to be able to efficiently process this data.

Typically, EEGs displays show no more than 10 to 15 seconds of data per screen of raw voltage readings and requires an analyst to simultaneously inspect multiple channels. In contrast, a compressed spectral array [1] or spectrogram display may show 2 to 8 hours of data on a single color map [2]. This allows analysts to quickly screen long periods of EEG data, determining which segments, if any, require direct review of the raw data. Spectrogram review reduces cEEG review time by 78% [6], with minimal loss of sensitivity compared with conventional review. For these reasons, we focus on building a tool to

rapidly analyze spectrogram data.

Spectrograms are the most widely used compressed data format for EEG data [8]. A spectrogram consists of three-dimensional plots with time on the x-axis, frequency on the y-axis, and EEG power on the z-axis. Figure 1-3 shows an example of rendered spectrogram data. An analyst typically views four spectrograms concurrently, mapped to different regions of the brain. Each region is formed by using multiple EEG channels where an EEG channel is the difference between voltages measured at two electrodes. This captures the summed potential of millions of neurons [7]. Figure 1-2 shows the electrode placement on the patient's scalp, yielding four regions for analysis: left lateral power, `LL`, (Fp1-F7, F7-T3, T3-T5, T5-O1), left parasagittal power, `LP`, (Fp1-F3, F3-C3, C3-P3, P3-O1), right lateral power, `RL`, (Fp2-F8, F8-T4, T4-T6, T6-O2), right parasagittal power, `RP`, (Fp1-F4, F3-C4, C4-P4, P4-O2).

Data from a single patient can vary in size from tens to hundreds of gigabytes and the number of EEG tests performed each year is estimated to be between 10 and 25 million [3]. As this corpus of data collected at the ICU continues to grow, efficient mechanisms to store and visualize this data at scale are key for analysts to quickly view patient screenings. Pinky aims to provide this for analysts by giving them a simple yet powerful interface to view spectrogram data.

1.4 System Architecture

Pinky is comprised of three coupled layers which handle storage, computation and visualization. Figure 1-4 shows the overall architecture of the system.

1.4.1 Storage Layer

The storage layer, discussed in detail in Chapter **??**, is responsible for storing raw EEG patient data and the calculated spectrogram. This datastore must optimize both reads and

writes of array based data for multidimensional arrays on the order of tens to hundreds of gigabytes.

1.4.2 Compute Layer

The compute layer, discussed in detail in Chapter **??**, is an extensible module which handles the algorithms to calculate the spectrogram and other EEG related calculations. As we discuss in Section **??**, there are a number of extensions the project can take, thus it is important that an interested developer can easily add functionality to this layer. In addition, the compute layer contains two servers. One server interfaces with the optimized EEG algorithms and the storage layer to serve array based data. The second server is a lightweight server for the web resources of the visualization layer.

1.4.3 Visualization Layer

The visualization layer, discussed in detail in Chapter **??**, is a browser based module that renders the data to the client. The interface allows users to query based on a patient's id (medical record number, `mrn`) and view a spectrogram for a given time interval. An analyst may smoothly pan and zoom throughout the dataset.

1.4.4 Visgoth System

Since enabling interactivity is an important design criteria, we have designed and built an optimization module for browser based visualizations named Visgoth. The system uses profiling information from the client and server to suggest an adaptive scaling of the visualizations served in order to keep latency consistent, regardless of a client's hardware or network bandwidth. We discuss Visgoth in detail in Chapter **??**.

1.5 Usage

The project code base is available publicly on Github [4] at `https://github.com/joshblum/eeg-toolkit`, with documentation for installing the project for develop-

ment. In addition, we have created Docker [5] images that can easily be installed for production use. Armed with a dataset, any curious doctor is able to install the images and load the data for analysis. The docker images are available for public use on DockerHub: `https://hub.docker.com/r/joshblum/eeg-toolkit-webapp` and `https://hub.docker.com/r/joshblum/eeg-toolkit-toolkit`. The Github project contains specific installation instructions.

1.6 Contributions

Pinky makes the following contributions:

- Implements an abstraction for array based storage systems.
- Implements three different backends which adhere to the abstraction.
- Evaluates the different backends for varying input ranges and workloads.
- Implements optimized algorithms for analyzing EEG data.
- Provides an extensible framework for accessing array based data and visualizing it in the browser.
- Implements scalable in-browser visualizations using the client's GPU.
- Implements a new system, Visgoth, for reducing latency for browser based visualizations.

These contributions enable doctors and medical expert analysts to interactively analyze EEG data at scale.

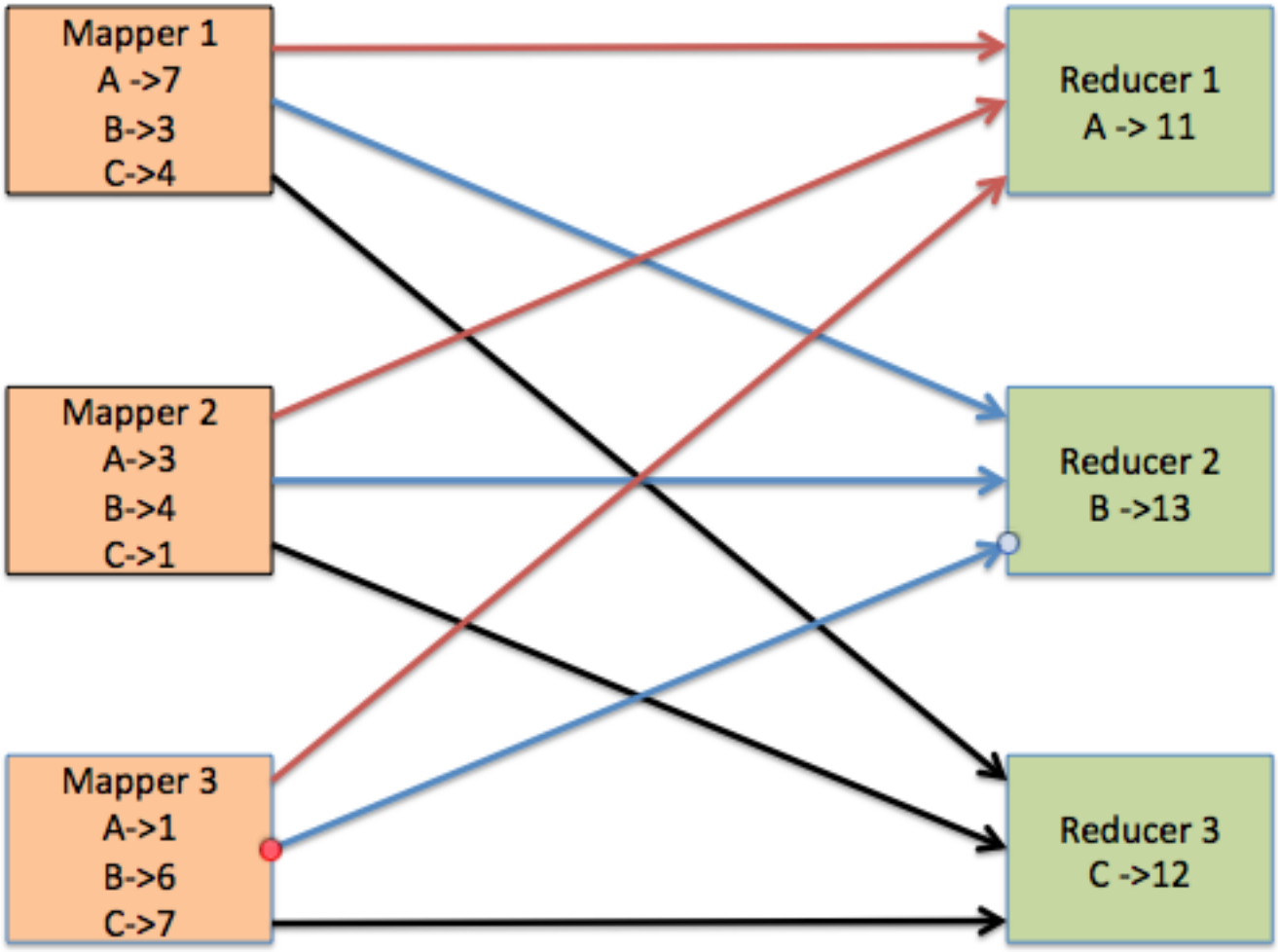

Figure 1-1: Shuffle for Letter Count in Map Reduce. The red arrows represent the transfer of the A key, the blue arrow represents the transfer of the B key, and the black arrows represents the transfer of the C key

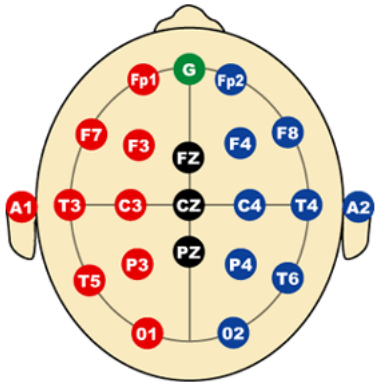

Figure 1-2: EEG electrode placement on a patient's scalp.

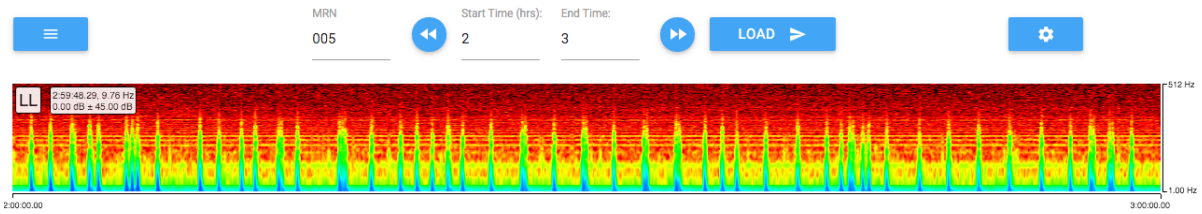

Figure 1-3: Spectrogram for one hour window of EEG data of the `LL` region of the brain.

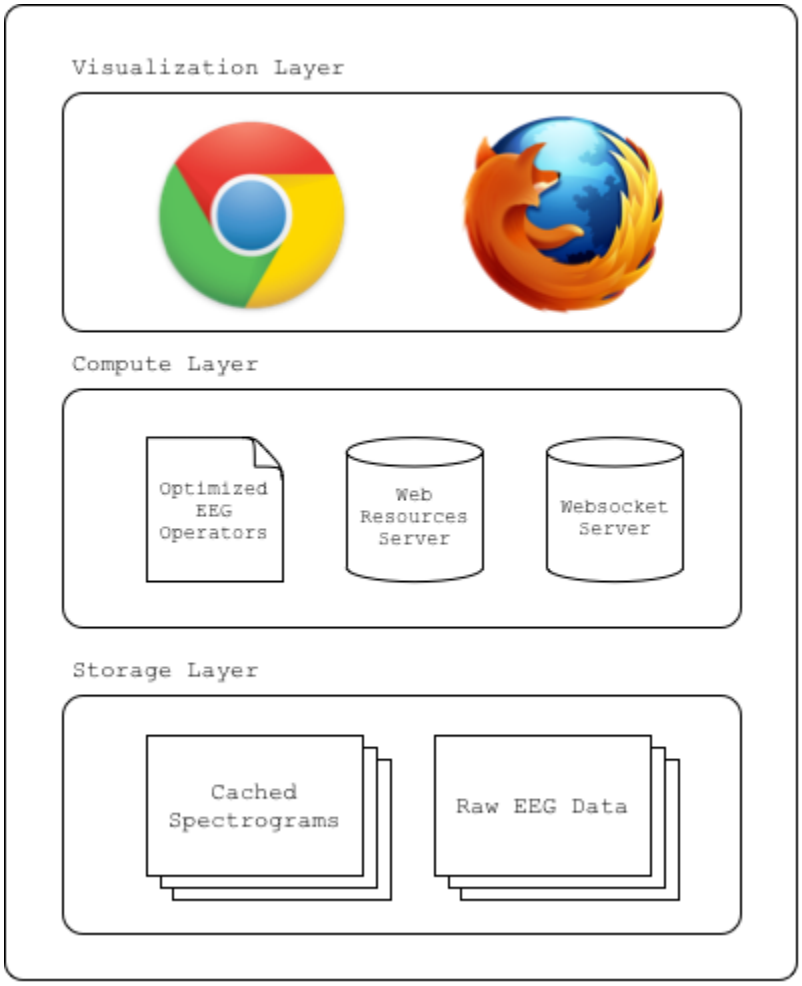

Figure 1-4: Pinky system architecture.

Bibliography

[1] A Bricolo, S Turazzi, Fo Faccioli, Fo Odorizzi, Go Sciarretta, and P Erculiani. Clinical application of compressed spectral array in long-term eeg monitoring of comatose patients. *Electroencephalography and clinical neurophysiology*, 45(2):211–225, 1978.

[2] C. Carlson. Can We Screen EEGs More Efficiently? Spectrographic Review of EEG Data. *Epilepsy Curr*, 15(1):24–25, 2015.

[3] L. Fleming Fallon. Gale Encyclopedia of Surgery: A Guide for Patients and Caregivers, 2004. http://www.encyclopedia.com/topic/electroencephalography.aspx.

[4] Github Inc. Github, 2008-2016. https://github.com.

[5] Dirk Merkel. Docker: Lightweight linux containers for consistent development and deployment. *Linux J.*, 2014(239), March 2014.

[6] Lidia MVR Moura, Mouhsin M Shafi, Marcus Ng, Sandipan Pati, Sydney S Cash, Andrew J Cole, Daniel Brian Hoch, Eric S Rosenthal, and M Brandon Westover. Spectrogram screening of adult eegs is sensitive and efficient. *Neurology*, 83(1):56–64, 2014.

[7] Ali H Shoeb and John V Guttag. Application of machine learning to epileptic seizure detection. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 975–982, 2010.

[8] Craig A Williamson, Sarah Wahlster, Mouhsin M Shafi, and M Brandon Westover. Sensitivity of compressed spectral arrays for detecting seizures in acutely ill adults. *Neurocritical care*, 20(1):32–39, 2014.