Large Scale Electroencephalography Processing With Hadoop

Matthew D. Burns, Yoav Freund

I. Introduction

Apache Hadoop [1] is an open-source implementation of the Google developed MapReduce [2] general programming model and Google File System [3]. These two core modules are known as Hadoop MapReduce and Hadoop Distributed File System (**HDFS**), which we refer to collectively as Hadoop. A few of the features of this system are:

- Runs on cheap hardware or cloud
- Robust/mature scheduler and load balancer
- Data-local task execution
- Highly redundant/fault-tolerant architecture
- Live tracking of cluster performance and job status

Instead of abstractly elaborating the details of how MapReduce and HDFS work together, we will demonstrate the basics of the system by example.

This paper serves to address the computational hurdles imposed by the analysis of electroencephalographic (**EEG**) data when the size of the data becomes very large (> 100 GB on disk). As the size of the data grows, network data transfer becomes very expensive and slow and it is more important to perform calculations as close as possible to the data. We will go over which parts of Hadoop worked for EEG analysis and which ones held us back. At the end, we suggest a better system in light of our experience.

To demonstrate how Hadoop can be applied to EEG analysis, we implement an EEG experiment using the Hadoop Streaming interface. Streaming works by passing the data to worker processes (called Map and Reduce tasks) as a text representation via stdin. Our executables are written in Python, recreating the Matlab analysis of [4] in its entirety. Our goal is to be able to parallel-process an arbitrary number of subjects in an EEG study, without incurring excessive network overhead.

II. Example Experiment/Problem

Event Related Potential (**ERP**) analysis [5] is a way to examine how specific cognitive processes affect brain electrical activity. An example of such an experiment is the Rapid Serial Visual Presentation (**RSVP**) experiment [6]. RSVP entails showing a subject a rapid burst of images while recording the subjects Electroencephalographic (**EEG**) output. In this case, the subject was shown a burst of 49 satellite images in 4.1 seconds. 60% of the bursts contained a single target image: an obvious plane image superimposed on the satellite background. The initial motivation for the experiment was to assist image analysts by rapidly triaging imagery with EEG assistance. In [4], the authors used the data from this experiment for another purpose: to evaluate two ERP estimators. They showed that ERPs obtained by averaging signal epochs can become degenerate when the spacing between stimuli in the experiment becomes very close. In that case, the averaged ERP estimate may become degenerate due to systematic overlap between the brain responses to the individual events. They showed that a multiple linear regression framework is capable of accounting for overlap between events and can produce a more accurate estimate of a single response, free of confounds introduced by averaging.

The experiment is fully described in [6]. 127-channel EEG data were collected during a RSVP task involving satellite image presentation. Seven participants recorded two sessions each, yielding 14 separate data sets for the entire study. The data were stored in a compact EEGLAB [7] format. The disk size of the study was approximately 25 GB: not especially large from a "big data" perspective. This made it a good candidate for development purposes, although the true performance edge from using Hadoop would come when processing a study with several hundred subjects.

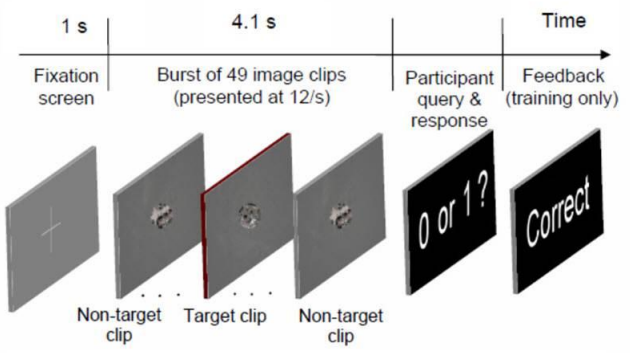

Fig. 1: Subjects in Rapid Serial Visual Presentation (RSVP) experiment are shown bursts of 49 satellite images. A single target image of a plane is randomly superimposed on the satellite background of one image for 60% of the bursts. After each burst, the participant is asked whether they saw a target image.

For each of the 14 datasets, the following steps (as in [4]) were applied with 5-fold cross validation (model is trained and tested 5 times):

M. D. Burns is with Dept. of Electrical and Computer Engineering, University of California, San Diego, San Diego, CA 92093 USA (phone: 858-822-7534; fax: 858-822-7556; email: mdburns@ucsd.edu).

(1) **Identify outlier** frames via probability
(2) **Reject events** near outlier frames
(3) **Generate ICA** component activations [8]
(4) **Generate predictor** matrix from recorded event timings
(5) **Calculate ERP estimate** with regression and averaging for all raw EEG channels and ICA activations
(6) **Calculate R^2** (coefficient of determination) [9] for regression and averaging in all raw EEG channels and ICA activations

R^2 is a common metric of how well a regression model accounts for variance in experimental data. The goal of the analysis was to demonstrate that regression has a statistically higher R^2 value, compared with averaging, when the responses to experimental stimuli are highly overlapping. Mathematical details of how to implement the analysis above are documented in [4]. Here, we focus on how these tasks can be accomplished in a distributed fashion using Hadoop and the broader implications to distributed EEG analysis.

III. Hadoop Implementation

The first problem we encounter is how to split up the data. Steps (1) – (4) require access to an entire dataset (14 datasets in this example). Our first approach was to compute (1) – (4) locally (~ 3min/dataset), then move the data to HDFS for further Hadoop processing on individual time series: channels and Independent Components (**IC**s). Since the bulk of the computation lies in (4) and (5), it is highly desirable to parallelize those steps.

Assuming we have completed the serial portion of the computation (~ 3 min/dataset), we want to render our data in a format that Hadoop can understand: the sequence-file input format. A sequence-file is a serialized list of key-value pairs (**KVP**). When the sequence-file is put to HDFS, it is spit into small blocks, replicated and distributed randomly across the data nodes. When Hadoop is invoked on the sequence-file, it processes each KVP as close as possible to whichever node hosts the data with Map and Reduce tasks.

In this framework, we want to encode each channel as a single KVP inside the sequence-file. It must be packaged with all information needed to perform the computation: the time series, the predictor matrix and the event structure of the experiment. As such, the sequence-file becomes highly redundant, replicating the same predictor matrix and event structure 127 times. This causes the size of the data to approximately double. On top of that, we must encode the binary Matlab data structures as text [10], causing another 50% increase.

Since the size of the data is the main reason for considering using Hadoop in the first place, it might seem like we have negated this approach entirely. On the contrary, we only have to write the data to HDFS once and we will presumably read it locally many times conserving network bandwidth and increasing the speed of computation. With 3x replication of data blocks, we will have magnified the disk size of the data nine-fold. Using current S3 pricing for data storage and network bandwidth, one could expect to pay about $120/month, prorated, to host a 150 GB study (expanded to 1.4 TB in HDFS). Network bandwidth, on the other hand, would incur the same charge by reading that 150 GB study from storage only 7 times. All of that is to say, expanding the data on disk nine-fold may not be as crazy as it sounds.

We encode the individual time series as KVPs in the sequence-file as follows:

- Key: the full path to the dataset, appended with a short code to identify whether the time series is an EEG channel or IC and which channel it is. */oasis/scratch/csd181/mdburns/data/RSVP/hadoop_input/exp44_continuous_with_ica.raw.1*
- Value: dictionary with the time series and relevant data for the calculation.

At the start of the program, the Map program receives KVPs, processes steps (5) & (6) once per cross validation fold and returns a dictionary with the results for that channel or IC. The Map task emits KVPs with the results of that channel as the value. The key from the sequence-file KVP is stripped of its type/number code, which is stored in the value dictionary. The modified sequence-file key is passed as the key to the output of the Map task. That is, each Map output from the same dataset will have the same key.

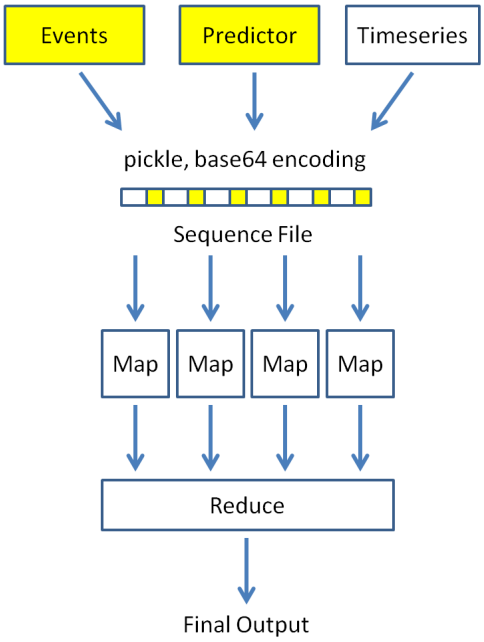

Fig. 2: Matlab based EEGLAB files are converted to sequence-file format, uploaded to HDFS and distributed to Map tasks for processing as individual time series. The results are collected, sorted and stored efficiently via the Reduce tasks.

In a process known as shuffle, Hadoop merges and sorts all KVPs from the Map tasks which have the same key. In the Streaming implementation, the reducer tasks receive a list of KVPs sorted according to dataset. The reducer simply accumulates the results for each experiment as a list. Additional sorting is applied to put the results in numerical order, and then the final results are sent out to HDFS. Fig. 2 briefly illustrates the entire process.

IV. Performance/Results

We run the analysis on a Hadoop cluster with 16 nodes. At 16 cores per node, this gives us a potential pool of 256 worker processes (Map tasks). For cluster stability purposes, we set the maximum number of Map tasks to 160. Our 14 subject analysis completed in 4.5 hours. By comparison, running the same analysis on a single node in Matlab (using parallel toolbox) would take approximately 56 hours.

From 14 datasets, we created 14 sequence-files, averaging 1.7 GB each (~ 25 GB) total. Since each sequence-file contains 256 KVPs (127 EEG channels, 127 ICs), we were able to max out the clusters CPU capabilities with just one sequence-file. This means our small 20 GB test study could scale to a cluster with 250 nodes (~ 4000 cores). Thus our processing power is effectively limited by how many nodes we can afford to run. The HDFS framework makes it very easy to scale a cluster.

As a whole, the program consumed the following resources:

- < 1GB network traffic
- 200 GB memory
- 617 CPU hours
- 26 GB read from HDFS

These metrics could be further turned to obtain a more streamlined process.

V. We Can Do Better

In the case of a larger study with hundreds or thousands of datasets, it would be undesirable to compute steps (1) – (4) on a single node. In that case, we would want store the Matlab formatted EEGLAB files directly to HDFS as single blocks. The individual time series will no longer be distributed over the cluster in this format. We would write two Hadoop programs: the first would process the single EEGLAB files, computing (1) – (4) in parallel over the cluster. The results of this program would be the sequence-files we have been using so far. When those files are written to HDFS, the individual time series will distribute across the cluster, as before. Thus, the granularity of the data distribution can be controlled in stages by separate MapReduce programs.

Since Hadoop Streaming only operates on text based inputs (via stdin, stdout), the EEGLAB files would still have to be encoded [10] before uploading to HDFS. For exploratory analysis, this is a fair tradeoff: Python is well suited to scientific computing on a large cluster. Without encoding everything as text, we would be forced to write our programs in Java, limiting our access to scientific Python packages like Numpy and Scipy. We believe most EEG experiments could benefit from the Hadoop Streaming approach.

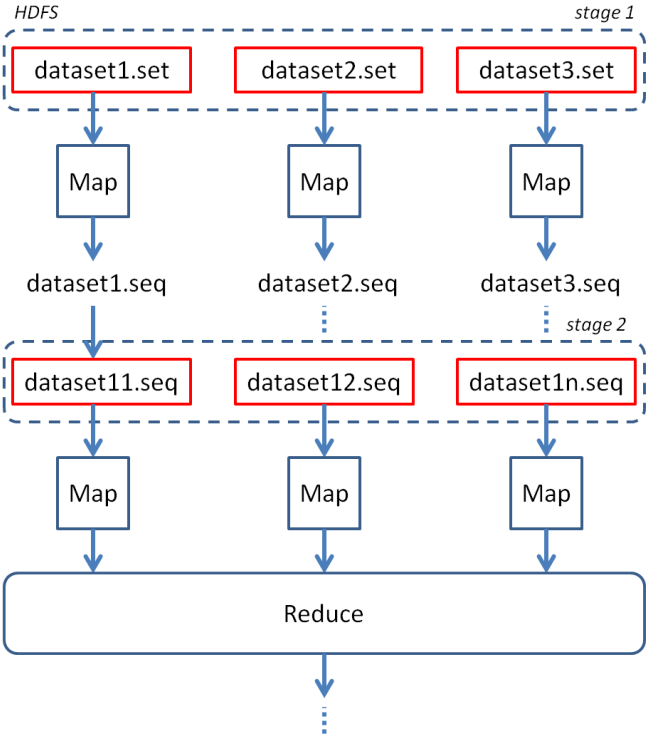

Fig. 3: By controlling the HDFS block size at different stages of the analysis, we can parallel process the data at different scales. In stage 1, we keep the datasets whole in HDFS for large scale processing. In stage two, we make the HDFS size very small to spread the individual time series across the system.

Fig. 3 shows how HDFS and Hadoop can be effective at processing data at different scales. At the larger scale, we can pre-compute as much of the problem as possible (e.g. factorization caching). The idea is to run stage 1 once and reuse those results many times in stage 2.

In our current implementation, step (5) is done in a single Map task. For this experiment, we were only minimizing the L2 norm of our error function in the regression phase. This was accomplished very efficiently as a quadratic cone program with the CVXOPT library [11]. If we wanted to try something more sophisticated (e.g. p-norm, regularization, cross-validate parameters, EM), we may desire an even finer grained distribution of the computations. This can also be accomplished in the MapReduce framework, but not with Hadoop Streaming. For such a problem, we introduce a different framework: Spark [12].

A drawback to using Hadoop is that it has no notion of cyclic data flow. A Hadoop program consists of reading a set of inputs from HDFS and writing the results. If one wants to implement an iterative algorithm with a scatter/update

approach, a separate Hadoop program must be launched for each iteration. This is highly inefficient; Hadoop takes several minutes just to set up each job. Spark supports all of the good features of Hadoop, with the added benefit of cyclic data capability. It is said to outperform Hadoop by an order of magnitude for iterative machine learning algorithms. This framework could support algorithms such as Expectation Maximization [13] and Alternating Direction Method of Multipliers [14] for very large datasets. We believe that a combination of Spark Python programs running on top of HDFS will offer the greatest flexibility and cost benefit for distributed EEG processing.

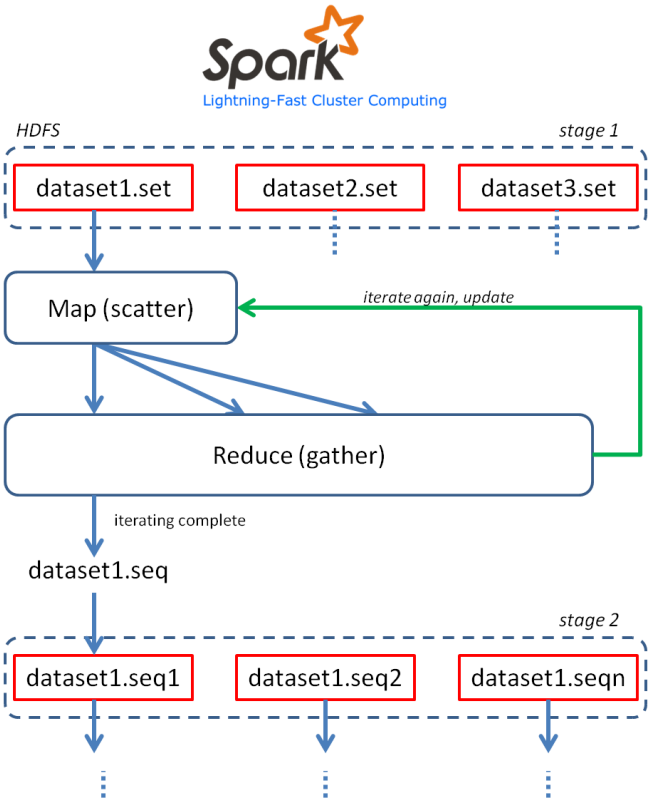

Fig. 4: By using Spark with HDFS, we can implement a system similar to the one shown in Fig. 3. The added benefit is that at each stage, we have the capability performing iterative algorithms which rely on scatter/gather techniques (e.g. Alternating Direction Method of Multipiers [14]).

VI. Conclusion

The barrier to entry for large scale EEG processing is lower than ever. The distributed frameworks Hadoop and Spark are capable of running on local Rocks clusters (concurrently) or on the cloud with services such as Amazon Elastic MapReduce and Elastic Compute Cloud. New technologies (perhaps something like RAMCloud [15]) are bound to eventually supersede these specific implementations, but the MapReduce trend for big data analytics is not likely to vanish any time soon.

VII. References

[1] White, T., *Hadoop: The definitive guide*. 2012: O'Reilly Media.

[2] Dean, J. and S. Ghemawat, *MapReduce: simplified data processing on large clusters.* Communications of the ACM, 2008. **51**(1): p. 107-113.

[3] Ghemawat, S., H. Gobioff, and S.-T. Leung. *The Google file system*. in *ACM SIGOPS Operating Systems Review*. 2003. ACM.

[4] Burns, M.D., et al., *Comparison of Averaging and Regression Techniques for Estimating Event Related Potentials*, 2013.

[5] Luck, S.J., *An introduction to the event-related potential technique*. 2005.

[6] Bigdely-Shamlo, N., et al., *Brain activity-based image classification from rapid serial visual presentation.* Neural Systems and Rehabilitation Engineering, IEEE Transactions on, 2008. **16**(5): p. 432-441.

[7] Delorme, A. and S. Makeig, *EEGLAB: an open source toolbox for analysis of single-trial EEG dynamics including independent component analysis.* Journal of neuroscience methods, 2004. **134**(1): p. 9-21.

[8] Makeig, S., et al., *Independent component analysis of electroencephalographic data.* Advances in neural information processing systems, 1996: p. 145-151.

[9] Nagelkerke, N.J., *A note on a general definition of the coefficient of determination.* Biometrika, 1991. **78**(3): p. 691-692.

[10] Josefsson, S., *The base16, base32, and base64 data encodings.* 2006.

[11] Dahl, J. and L. Vandenberghe. *CVXOPT: A python package for convex optimization*. in *Proc. Eur. Conf. Op. Res*. 2006.

[12] Zaharia, M., et al. *Spark: cluster computing with working sets*. in *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*. 2010. USENIX Association.

[13] Moon, T.K., *The expectation-maximization algorithm.* Signal Processing Magazine, IEEE, 1996. **13**(6): p. 47-60.

[14] Boyd, S., et al., *Distributed optimization and statistical learning via the alternating direction method of multipliers.* Foundations and Trends® in Machine Learning, 2011. **3**(1): p. 1-122.

[15] Ousterhout, J., et al., *The case for ramcloud.* Communications of the ACM, 2011. **54**(7): p. 121-130.