Incremental Full Correlation Matrix Analysis for Real-Time fMRI Studies

Yida Wang*, Bryn Keller†, Mihai Capotă†, Michael J. Anderson†, Narayanan Sundaram†, Jonathan D. Cohen‡, Kai Li*, Nicholas B. Turk-Browne‡, Theodore L. Willke†
*Department of Computer Science, Princeton University,
†Parallel Computing Lab, Intel Corporation,
‡Princeton Neuroscience Institute, Princeton University

***Abstract*—Real-time functional magnetic resonance imaging (rtfMRI) is an emerging approach both for studying the function of the human brain, and for neural feedback-based training. To date, rtfMRI has relied exclusively on the real-time analyses that treat activity in different regions as independent of one another. However, critical aspects of brain function may depend on, and be revealed more sensitively by *correlations* among regions. An exhaustive analysis of such correlations is substantially more demanding computationally, so full correlation matrix analysis (FCMA) of the entire brain has not been carried out in real-time before. This paper presents the algorithms and an implementation of the first real-time system to perform whole brain FCMA in real-time. Our system includes incremental voxel selection, model training, and real-time classification. We have implemented this system on an fMRI scanner connected to a computer cluster over HTTP. Experiments show that our system is able to achieve real-time FCMA analysis of a stream of brain volumes with neurofeedback with less than 200 ms of lag with very few exceptions. The incremental FCMA algorithm running on our real-time fMRI system performs about 1.8x-6.2x faster than using an offline FCMA toolbox in the real-time context while getting comparable neurofeedback results.**

***Keywords*-Real-time fMRI; rtfMRI; fMRI; magnetic resonance imaging; streaming; machine learning; full correlation matrix analysis**

I. Introduction

Functional magnetic resonance imaging (fMRI) is the dominant technique for investigating human brain activity in neuroscience research. Almost all existing fMRI studies are conducted in an *offline* fashion — statistical analysis occurs only after all data have been acquired and sent to a file server or lab for processing, long after the research subject has been taken out of the scanner. This offline approach allows researchers to design an experiment, perform it in multiple subjects, and then analyze the reliability of the data by examining consistency across subjects. Although sufficient for many purposes, this approach has three major limitations: First, the pace of discovery is very slow, with typical fMRI studies lasting 6-12 months total, and often at least 2-3 months before even tentative results are known. Second, it is assumed that the same exact experiment should be run in each participant, missing opportunities to tailor the level of difficulty to an individual's cognitive abilities and to assess whether enough or the right kind of data has been collected in a given session. Third, it misses the opportunity to use information acquired during scanning as feedback to the subject, either to enhance participation and/or training (e.g., on attention [1]).

Recently, it has become possible to conduct online or real-time fMRI (rtfMRI) studies, in which data are preprocessed and analyzed as they are collected, keeping pace with the rate of data acquisition [2], [3], [4]. There are many forms of rtfMRI: "triggering" involves initiating experimental trials based on the amount or pattern of activity in a brain region, allowing for stronger inferences about the region and behavior [5]; "adaptive design" involves altering experimental parameters such as stimuli and tasks, in order optimize the experiment to recruit particular brain regions [6]; and "neurofeedback" involves returning to subjects some visualization of the amount or pattern of activity in a brain region, helping them better engage this neural representation [7], [8]. As an example, this latter form of rtfMRI with neurofeedback was recently used to train subjects to better sustain their attention, by externalizing their internal attentional states such that they could be better monitored and controlled [1].

Despite the advent of rtfMRI, there still exists a major gulf between online and offline measures. One critical constraint concerns the type of analyses that are possible. Most analyses of fMRI data and, to date, all analyses used in rtfMRI, treat the activity observed in each voxel independently of one another. Examining patterns of activity in this way has led to considerable progress in brain research. However, brain function relies not only on the isolated activity of different areas, but as if not more critically on *interactions* between different brain systems, which can be reflected in *correlations* of activity among these. Such correlations, sometimes referred to as functional connectivity analysis, have become an increasingly important focus of brain imaging research [9]. However, because of computational constraints, such correlational analyses have typically been restricted to predefined regions of interest (ROIs). This is problematic not only because it restricts the extent of analysis, but also biases it. Typically, correlational analyses have focused on seed regions identified by their isolated activity. However, this is blind to effects that reside entirely in patterns of correlation (i.e., without changes in mean activity). To address this limitation, we recently developed a method for full correlation matrix analysis (FCMA), allowing unbiased analysis of patterns of correlations in activity over the entire human brain. We have shown that this can reveal aspects of brain function not revealed by traditional, non-correlational methods [10]. However, because of the computational demands, to date it has not been possible to incorporate this method into studies using rtfMRI. In this paper, we describe a distributed system that addresses this limitation, and provide proof-of-concept for its use in rtfMRI.

In implementing this distributed system, we also addressed another problem that has constrained the use of rtfMRI — the lack of readily accessible, standardized tools for implementing it. Nowadays, different research teams have to set up their own rtfMRI experimental environments, typically using a standalone machine sitting beside the scanner to receive and analyze the incoming data stream. This results in considerable duplication of effort, and often inefficient solutions, or ones limited by local resources. To address this problem, we describe a new, general distributed system for rtfMRI that runs on a computer cluster that can be deployed in Software as a Service (SaaS) mode, which will allow neuroscientists from MRI centers around the world to conduct their own real-time neurofeedback experiments, leveraging and creating shared computing resources. We have built the system to require very minimal resources in the scanner room, and the back end, which resides on a local computer cluster or potentially a cloud service provider, has a simple REST HTTPS API that allows easy integration and fits easily into existing network administration rules. The REST server is a gateway to the distributed back end, which provides a flexible set of processes to handle classification, training, and incremental feature selection in various topologies, with different experimental configurations, and using a variety of algorithms.

We implement real-time FCMA as a neuroscientific application to our system (henceforth referred to as rtFCMA). Just as this approach has already produced novel insights when used for offline analysis [10], it may be as, and perhaps even more, valuable for real-time analysis. We adapted the fully optimized batch FCMA code by leveraging the characteristics of real-time data streams to only process the incoming data. In addition, rtFCMA is useful for showcasing our general framework, as it is an especially computationally intensive analysis that was not possible even in offline analysis until recently, and represents an upper limit on current computational demands for brain imaging analysis.

Figure 1 shows the general concept of our rtFCMA system. An fMRI machine continuously scans the brain of a human subject while they perform certain tasks to generate a data stream of 3D brain volumes. The data are sent to the rtFCMA data analysis that runs on a computer cluster. rtFCMA processes the brain volumes and produces feedback to send back to the fMRI scanner in real time to form a closed data transferring loop. Our system is the first ever rtfMRI system with low latency closed-loop feedback running on a computer cluster.

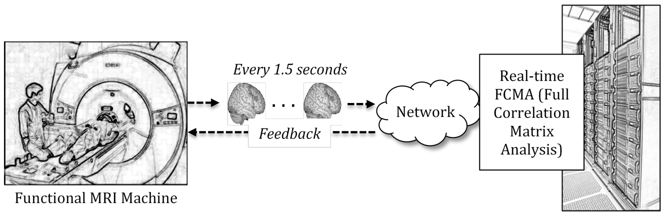

Fig. 1. Conceptual diagram for rtFCMA, an fMRI system with closed-loop neurofeedback based on real-time FCMA data analysis.

The rtFCMA system requires generating the neurofeedback with short latency. As an example, a typical fMRI scanner spends 1.5 s to produce a brain volume. To keep pace with the brain data stream from the scanner, rtFCMA needs to complete its processes for a given volume (particularly correlation computation and multivariate classification) before the end of the next volume. The latency from the completion of a brain volume to its delivery to rtFCMA can be as high as 780 ms in practice, which leaves rtFCMA a time budget of about 720 ms. We adapted the offline batch FCMA algorithm in various ways to meet this latency requirement. Experiments show that rtFCMA was able to conduct an FCMA study and provide neurofeedback with < 200 ms of lag with very few exceptions. The incremental FCMA algorithms running on our real-time fMRI system performed 1.8x-6.2x faster than the offline batch version in a real-time context, while maintaining the same level of accuracy.

II. Full Correlation Matrix Analysis

In this section, we briefly describe FCMA and its applicability to the rtfMRI studies.

A. Background

fMRI data consists of a series of 3D brain volumes collected in continuous time, typically one volume per 1.5 s. In the setting of our studies, each brain volume has about 25,000-35,000 points (called voxels) containing values depicting the instantaneous amplitude of blood-oxygen level dependent (BOLD) activity in the corresponding area. The subject performs cognitive tasks during the fMRI scanning, each of which corresponds to a continuous time course called an "epoch of interest". The epochs of interest can be labeled based on the types of cognitive task, for instance a label that classifies the sort of images being shown to the subject during that epoch.

Unlike traditional approaches to analyzing the neural patterns of the brain based on the amplitude of the voxel

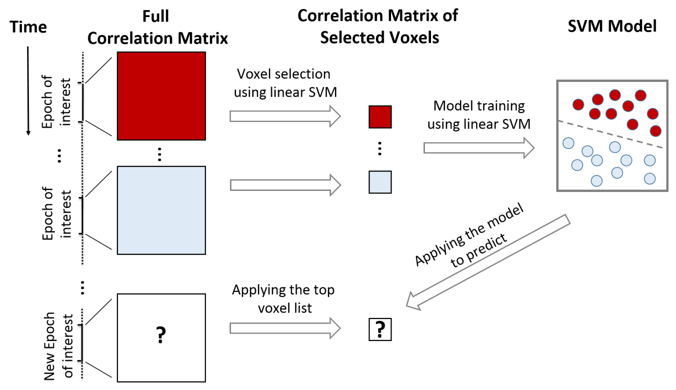
Time
Full Correlation Matrix
Correlation Matrix of Selected Voxels
SVM Model
Epoch of interest
Epoch of interest
New Epoch of interest
Voxel selection using linear SVM
Model training using linear SVM
Applying the model to predict
Applying the top voxel list
?
?

Fig. 2. The data processing flow of FCMA.

activity during different cognitive tasks, FCMA studies the fMRI data in terms of correlation between different brain regions. Figure 2 shows the general data processing flow of FCMA. The basic unit of operation for FCMA is a full correlation matrix, i.e., the temporal correlation of the activity of every voxel in the brain with every other voxel. A separate full correlation matrix is computed for each epoch of interest in the scanning session. Based on the task types of the epochs, the matrices are labeled as different categories (depicted in figure 2 as dark red and light blue), e.g., in epoch x, the subject is shown face images, in epoch y, the subject is shown scene images.

The goal of FCMA is to predict the state of the brain (e.g., the subject is viewing a series of faces or scenes) by analyzing full correlation matrices. In order to extract useful information from the huge and noisy full correlation matrices, FCMA first performs voxel selection (i.e., feature selection), which picks a subset of voxels whose correlation vectors are most category selective among all brain voxels for making predictions. In addition to improving the results by removing noise, voxel selection also makes results easier to interpret for neuroscientists and speeds up computation. Specifically, the voxels are selected based on the cross-validated prediction accuracy of their individual correlation vectors (i.e., for each individual voxel we build and test a linear SVM model).

Once the most relevant voxels have been selected, FCMA uses them to build correlation matrices of top-K selected voxels and trains a machine learning model to classify these correlation matrices of selected voxels in categories. In both voxel selection and correlation matrices training phases, FCMA typically deals with only hundreds of samples (corresponding to the number of epochs of interest) in tens of thousands of dimensions (corresponding to the number voxel pairs). Therefore, we build models using linear SVM to avoid overfitting.

In order to exhaustively study the pairwise correlations of all brain voxels, FCMA runs as a distributed program on a computer cluster. It uses different nodes to deal with different portions of brain voxels in parallel, which greatly reduces the memory and computation burden of a single node and accelerates the processing time. After careful optimization, it takes seconds or minutes to analyze a typical fMRI data in terms of correlation in batch[11].

B. Real-time FCMA

It is beneficial to incorporate FCMA into rtfMRI studies to understand subtle changes in brain interactions during different cognitive tasks as we discussed in Section I. A closed-loop rtFCMA system can use a pre-trained model to classify the incoming data in terms of correlation on the fly, therefore only the classification component of FCMA is needed in the pipeline, which must return the classification result as the neurofeedback in a short time. However, in order to achieve better classification accuracy, it is advantageous to leverage the data collected from the current scanning subject to incrementally update the model. As a result, the voxel selection and training components of FCMA also need to perform efficiently to provide the best model possible given the data that has arrived so far in the experiment. Our system can either start with a pre-trained model loaded from disk, or start from scratch and use the incoming data to build the model on the fly.

Our rtFCMA system is complex from two perspectives. First, it is a nested distributed system. In one layer, it distributes the voxel selection, correlation matrix training, and classification to different nodes; in the other layer, it also distributes the voxel selection work to multiple voxel selection workers. Second, it is a second-order machine learning system. Unlike most distributed machine learning systems, it does not directly take the incoming data to form samples for machine learning training. Instead of using amplitude of voxel activity directly, it uses the correlation between voxel activities, so it must compute the correlation based on incoming data. Additionally, using the correlation data as input for machine learning expands the data size by roughly three orders of magnitude. In order to meet the real-time requirement, high performance computing techniques are critical in this scenario.

III. Architecture for an rtFCMA System

A. Overview

We have built a configurable system, in that the various processes involved can be assigned to whatever nodes are desired at launch time, and also in that the communication and system organization are independent of the machine learning algorithms and feature selection algorithm used. In the experiments in this paper, we have configured the system to use SVM for classification, and FCMA for voxel selection, but the framework of our system allows any other algorithms that implemented the interfaces the system requires.

rtFCMA consists of several cooperating services. A lightweight daemon process on a machine in the fMRI scanner room reads brain volume files as they are generated by the scanner, and uploads them to a web server,

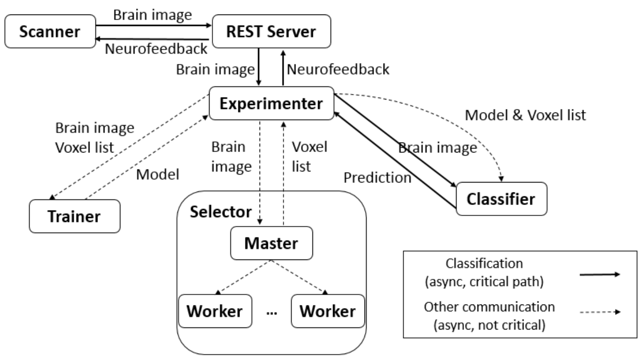

Fig. 3. Architecture diagram

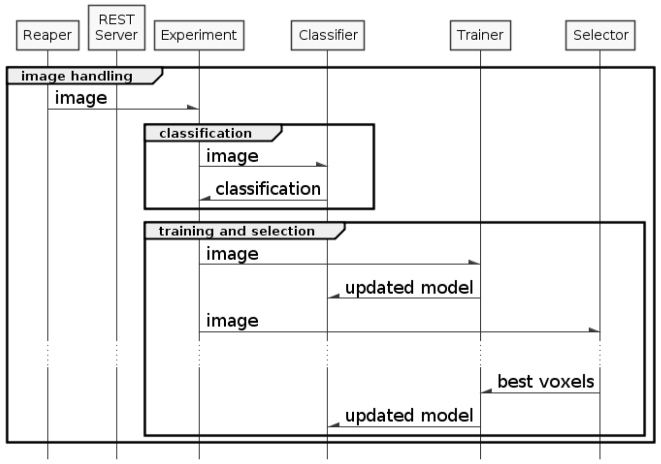

Fig. 4. Brain volume processing sequence. Note that all communication is asynchronous, and specifically that classification is not blocked by any other process.

which can be considered the front-end of the system. From there, the brain volume is dispatched to three separate processes, which may or may not be on different machines: the *classifier*, the *trainer*, and the *selector*. Each of these is described in more detail in Section III-B. Among the three, the classifier is in the critical path of the closed-loop system, which must classify the brain volumes using a machine learning model (in this case, linear SVM) within a few hundred milliseconds, so that those results can be used as neurofeedback to direct the adjacent cognitive task assigned to the subject (e.g. showing which image to the subject). In most deployments, the classifier service should be run on a dedicated node in order to prevent contention with other services that might threaten latency guarantees.

As an option, the brain volumes are also sent to the selector for incremental feature (voxel) selection as well as the trainer for updating the model using the results from the selector (described in Section IV). The actions of the selector and the trainer are not under the stringent time requirements of the classifier, but it is still highly desirable they should complete in a few seconds or less, so their outputs (models and voxel lists, respectively) can be put into use as quickly as possible.

B. Implementation

RtFCMA uses the actor model [12], as implemented in the Akka [13] toolkit, to distribute computation across a computer cluster. Figure 3 shows the organization of the system. The actors in rtFCMA are written in Scala and are lightweight and focused on communication and management of the system.

The services rtFCMA provides are accessed using a RESTful web API. The REST server is written in Scala, using the Spray framework [1], and does relatively little processing itself, instead it relies on the distributed actors in the system.

A process called *reaper* runs on the fMRI computer. Its job is to send brain volumes to the REST server (using HTTP POST requests) as they are generated by fMRI scanner. The intent is that this should be a minimal daemon process that does not interfere at all with the normal operation of the fMRI scanner. The reaper is written in Python, and is borrowed from the SciTran project [14] with minimal modification.

[1]http://spray.io

Requests received by the web server are dispatched to various actors for processing. Although the actors facilitate communication in the system, for good performance of an actor system, it is important to have both small message sizes (no more than a few kilobytes) and to do all time consuming work on a different thread. Thus as messages that require time to process arrive at one of the system's actors, the message will be validated and then queued for processing by a different thread dedicated for that purpose, so that the actor is free to respond to the next message. Furthermore, the heavy analytical computation in the system is delegated via Java Native Interface (JNI) wrappers to C++ routines that analyze the brain volumes. As a matter of design, the system is built to be extended to use many different algorithms, not only FCMA. Therefore the actors communicate with the algorithm-specific code via algorithm-agnostic interfaces. In the remainder of this section we describe the actors.

Experimenter: The Experimenter is the hub of the experiment. It is the coordinator that sees the brain volumes through their life-cycle. It creates actors to implement the needed functions, assigning them to remote nodes based on configuration files. The REST server sends brain volumes to Experimenter for processing, and also forwards client requests for the classification results for each brain volume to it. Figure 4 shows the sequence of actions that happens as each brain volume in an experiment is processed by the actor system. Note that there can be multiple experiments active at the same time, each with its own Experimenter created on-demand by the REST server when the first brain volume for the experiment arrives.

Selector: The Selector's job is to determine which voxels

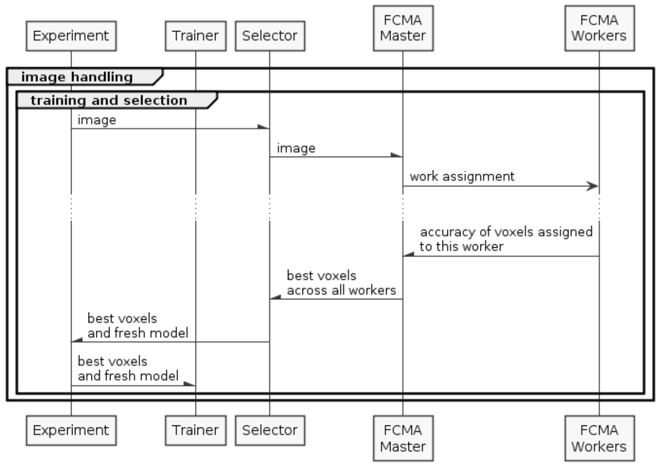
Experiment
Trainer
Selector
FCMA Master
FCMA Workers
image handling
training and selection
image
image
work assignment
accuracy of voxels assigned to this worker
best voxels across all workers
best voxels and fresh model
best voxels and fresh model

Fig. 5. FCMA distributed voxel selection sequence.

in the brain volume are category selective. It does this by delegating to the FCMAMaster actor, which in turn manages communication with a fleet of FCMAWorker actors. These are analogous to (and share C++ code with) the workers in the MPI job of the offline version of FCMA described in [11]. Even in the original offline C++ version of FCMA, the communication required between master and workers is simple and does not use any advanced features of MPI. With this in mind, we abandoned MPI and use Akka messages to distribute the work and collect results, which simplifies the deployment requirements for the system. Additionally, we believe this will simplify future work on fault tolerance, as Akka, with its heritage from Erlang, is better suited to building fault tolerant systems than MPI. The detailed sequence of how the selection is managed is shown in Figure 5.

Trainer: The Trainer takes in a brain volume and uses it to update its classification model. If the Trainer does in fact generate an updated model (some, such as FCMA, do not necessarily generate a new model after every brain volume), it sends the model to the Classifier.

Classifier: The Classifier receives brain volumes and passes them through a trained machine learning model to classify them. For this paper, it is configured to use SVM. Its classification results are sent back to the Experimenter for storage until a client, such as the subject feedback system, requests those results.

IV. rtFCMA Algorithms

Based on the system framework described in Section III, we are able to deploy FCMA as an fMRI data analysis application to use in real-time fMRI studies. However, the existing FCMA algorithms process the fMRI data in batch, which is not suitable for real-time analysis. In the real-time setting, when a new brain volume comes in, we need to leverage what we already have from previous computation and analysis and avoid re-computation as much as possible. In addition, it is helpful to incorporate the fresh incoming data to the classification model used for providing neurofeedback. To implement these ideas, we modified the optimized offline code to run in an incremental fashion as described in detail below.

A. Voxel Selection

Voxel selection represents the feature engineering step in FCMA. It is a voxel-wise screening to pick the most category selective voxels in terms of correlation. Specifically, for each voxel, the selector computes one correlation vector per epoch which consists of the correlation between the voxel and every other voxel in the brain; the correlation vector represents the row of the full correlation matrix corresponding to the voxel. To assess the prediction potential of voxels, the voxel selector builds for each voxel a machine learning model using the correlation vectors as training data. The voxel selector tests the accuracy of each voxel model using cross validation. It then sorts the voxels according to the accuracy of their models, which is an indicator of their category selectivity.

The algorithm can be described as a three-stage pipeline of correlation computation, correlation normalization and cross validation. The correlation computation builds the full correlation matrix, composed of correlation vectors. The normalization stage brings data from different subjects to the same scale for across-subject analysis. The cross validation stage uses subjects as folds and trains a linear SVM classifier to test in each fold for each voxel. That is, for m subjects, we build m linear SVM classifiers for each voxel using correlation vectors from $m-1$ subjects as training data and test using the mth subject.

Since it needs to exhaustively go through all voxels of all subjects, voxel selection is the most computationally intensive component in FCMA. We have already optimized voxel selection for batch processing by carefully designing the processing pipeline and organizing the data layout to use the cache and vector processing units of the processor efficiently [11]. The SVM linear kernel matrices are precomputed as well, to reduce the memory usage and avoid recomputation in the training iterations. The batch algorithm works in a distributed way by sending different portions of voxels to different worker nodes via MPI to process in parallel. A master node will collect results from the workers to form the voxel list sorted by cross validation accuracy numbers.

However, the optimization above assumes that the algorithm gets all data in at once, which is not true in the real-time context. In rtfMRI studies, data come in brain volume by brain volume. Voxel selection happens every time an epoch of interest completes. Figure 6 shows the scenario in a worker node when the data of a new epoch of interest is completely received. Notice that this worker node is assigned V voxels by the master node. At that time, there are already M epochs' data received and processed in the memory. When a new voxel selection takes place after receiving epoch $M+1$, we do not need to recompute the correlation and the SVM kernel matrices over all previous epochs of interest. Instead we only need

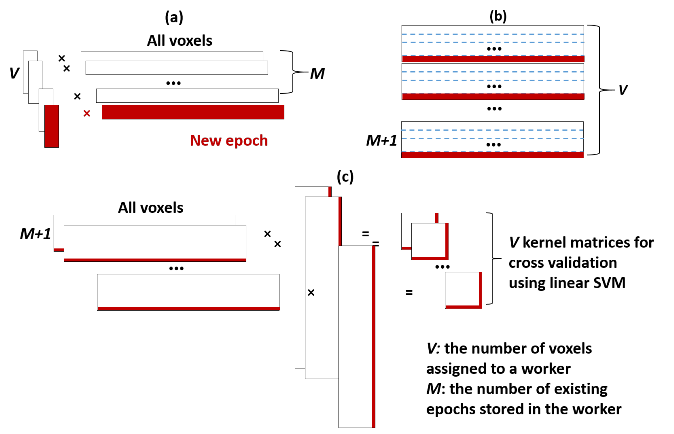
(a)
All voxels
V
M
New epoch
(b)
V
M+1
(c)
All voxels
M+1
V kernel matrices for cross validation using linear SVM
V: the number of voxels assigned to a worker
M: the number of existing epochs stored in the worker

Fig. 6. Incremental voxel selection in one selector worker actor. The worker is assigned to process v voxels of the brain, making sure all data resides in the memory in the entire process. (a) In the correlation computation stage, only the data of the newly coming epoch is computed; (b) In the normalization stage, only the data related to the newly computed correlation is normalized; (c) In the cross validation stage, when computing the kernel matrices, only the similarity values involving the newly normalized data are computed to expand the kernel matrices.

to compute the correlation over epoch $M + 1$ which is in dark red in figure 6a, normalize it using the existing data of the same subject as depicted in figure 6b, and compute the corresponding similarities with the existing correlation samples as shown in in figure 6c. This saves recomputation overhead and makes it possible to finish one round of voxel selection faster so that the latest voxel selection result can be leveraged in the following epochs as soon as possible to refresh the model used for classification.

B. Training with the Latest Data

Training a precise model for real-time classification is critical to rtfMRI studies. Traditionally, researchers use a fixed model trained from previously collected data [1] because a dynamically updated model cannot be handled in real-time by a single analysis machine. The fixed model does not take the latest brain data into consideration, so it may not reflect the neural status of the current subject, which therefore harms the classification performance.

In our rtFCMA algorithm, we implement a correlation-based training pipeline to incorporate the most recent data into the model. We first compute the correlation matrices of selected voxels consisting of correlations between the top-K selected voxels produced by the latest voxel selection over all existing epochs till now. Then we serialize the matrices into high dimensional samples, normalize them within subject, and train a linear SVM classifier over them using sequential minimal optimization (SMO) algorithm [15] from a randomly initialized state. This produces an SVM model reflecting the latest data and voxel list. Note that since the voxel list will change between voxel selections, we will have to reconstruct the correlation matrices of selected voxels accordingly, which leads to different training samples between different voxel selections and prevents us from doing an incremental training to update the model based on the current state. Although the incremental training may save a number iterations in the SMO algorithm compared to training from a random state, in our case we only have a small number of training samples (corresponding to the number of epochs the subjects go through, typically tens to hundreds), so the difference is not significant.

The training pipeline, together with the incremental voxel selection, work as independent processes to update the model asynchronously from classification.

C. Real-time Classification

The voxel selection and training described above are only auxiliary components that refine the model. The critical path of closed-loop rtfMRI studies consists of using the model to classify incoming data and sending the resulting prediction along with its confidence as neurofeedback back to the fMRI scanner. The classification must be done in real-time so that the neuroscientists can use the neurofeedback it produces for the subsequent steps in the experiment, e.g., adjusting the experimental task accordingly if the prediction the system makes matches what the subject is really doing. Typically, an rtfMRI study requires the neurofeedback of brain volume N to be produced within the time that brain volume $N + 1$ is generated so that it can direct the cognitive task the subject performs when generating brain volume $N + 2$.

In rtFCMA, unlike the previous rtfMRI studies which work on the brain's raw BOLD activity, the classification is done on the correlation values computed from incoming brain volumes over each epoch of interest. Because it is based on correlation, classification is only performed after several brain volumes within the same epoch of interest are collected. Once it has accumulated enough brain volumes of the same epoch, the classifier actor takes the top-K voxels from the latest voxel list to compute a correlation matrix of selected voxels, normalizes the values using up-to-date data from the same subject, and applies the result to the latest model as a test sample. In order to run multiple correlation-based classifications for one epoch of interest, the system applies a sliding window of length L to the epoch of interest. That is, each time a new brain volume is received within the epoch of interest, we drop the volume that was collected L time points ago and incorporate the new one. When L is equal to the length of an epoch, we only conduct one correlation-based classification per epoch. The sliding window does not cross an epoch, so when a new epoch starts, the sliding window accumulation also needs to restart.

Using the rtFCMA algorithms, our system is able to conduct an rtfMRI experiment in terms of full brain correlation-based analysis. The system takes the fresh brain data to update the model and the voxel list right after an epoch of interest on the fly, and always uses the latest model to classify the correlation computed based on the current epoch including the newly incoming brain data.

V. Evaluation

To evaluate the performance of our real-time fMRI system, we pursued answers to the following questions:

1) What is the overall performance of our real-time fMRI system? Specifically, can the classification produce neurofeedback within the time of generating the next brain volume? Can the incremental voxel selection and training update the model efficiently?
2) What is the performance gain of incremental voxel selection comparing to the optimized offline version?
3) Is our incremental voxel selection scalable as we add more compute nodes?
4) Can the simulated real-time experiments based on offline data achieve the comparable classification accuracy as the offline analysis?

A. Experimental setup

In our evaluation, we consider rtFCMA and the original FCMA toolbox, which we refer to as batchFCMA. We compare overall performance, as well as per-component performance.

All testing is done on *Metacortex*, a 50-node cluster located at the Princeton Neuroscience Institute (PNI). All nodes are interconnected by an Arista 10GE switch. Each node has two Intel Xeon E5-2670 CPUs running at 2.6 GHz, 256 GiB RAM, and eight 3 TB SATA hard disks. All our C++ code is multithreaded and uses 32 threads per process and one process per node.

We used two datasets to simulate rtfMRI studies. The *facescene* dataset was collected as the localizer runs in [16] and used as a proof of concept of FCMA in [10]. It consists of 244 brain volumes per subject from 18 subjects. The volumes are grouped in 12 epochs per subject, each corresponding to the subject in the fMRI scanner being shown either a series of face or scene images. The interval between the starting points of two epochs is 30 s. The *attention* dataset was collected and analyzed in [17]. It consists of 360 brain volumes per subject from 30 subjects. The volumes are grouped in 18 epochs per subject, each corresponding to the subject in the fMRI scanner being asked to attend the images on the left or right side of the screen. Like *facescene*, the interval between the starting points of two epochs in *attention* is also 30 s.

B. Full System Performance

We first demonstrate the performance of our system when all functions of the system, namely, voxel selector, trainer and classifier, are on. Two datasets mentioned above were used in this experiment, respectively. In order to fully simulate an rtfMRI experiment, we fed in our system brain volume by brain volume with a 1.5 s pause, which is the time interval of a brain volume being generated in the fMRI scanner; and introduced a 1-minute pause between subjects. Our system started from scratch, that is, there is no pre-trained model that can be used, and our system created the model and updated it using the incoming data stream while the real-time experiment was taking place. Like the batchFCMA, in voxel selection, our system did a leave-one-subject-out cross validation. The system invoked the voxel selector at the end of each epoch. In order to ensure the sample balance across cross validation folds, when receiving the Eth epoch of subject S, our system used the first E epochs of all S up-to-date collected subjects to do an S-fold cross validation. The selector master updated the model using the top voxels of the newly generated voxel list right after each voxel selection. In the simulated rtfMRI experiments, we used top-500 voxels to construct the model since the number is enough to depict a reasonable brain interaction pattern according to the accuracy numbers reported in [10], [17]. The classifier took the latest model along with the top-500 voxels in its corresponding voxel list to classify the correlation sample computed over a sliding window which included the newly incoming brain.

Figure 7 shows the classification performance of our system on two datasets. As we mentioned in Section I, although the time interval of generating a brain volume in our experiment is 1.5 s, the time budget left for classification is less than that considering the latency of other necessary steps in the real rtfMRI pipeline, e.g. reconstruction and preprocessing, which in practice took up to 780 ms. Therefore, we set a hard deadline of 720 ms to the classification. From the figure we can see that classification in our system generally finished faster than 200 ms, which is well bellow the deadline, indicating that our system is able to satisfy the latency requirement of closed-loop real-time fMRI studies. The only exceptions are the spike at the end of the *attention* dataset, and the ones clustered around epoch 225 of the *facescene* dataset, which we suspect were caused by network interference in the cluster. It should be noted that measured times do not include network communication overhead between the scanner and the REST server. Even if we consider cross-continental links, if we use a conservative estimate of 200 ms (for instance based on [18]) for each HTTP call, and assume the feedback system polls the REST server for results every 100 ms, the overhead should be within 500 ms. Since the classifier completes in less than 200 ms with very few exceptions, the system meets the 720 ms deadline. Additional care, such as use of keepalives or server side events (SSE) could well improve these numbers.

Figure 8 shows model update performance in our system. To finish one run of model updates, our system performed an incremental voxel selection and used the latest top-500 voxels to train a model based on the currently collected data. The model is updated after every epoch. From Figure 8 we are able to tell that model updates will complete before the end of the next epoch (30 s), marked with a red line, not visible in the *facescene* dataset which finishes substantially faster. That is, the classifier is able to use a fresh model that is updated by the data from the last epoch. The only two exceptions are the last epochs

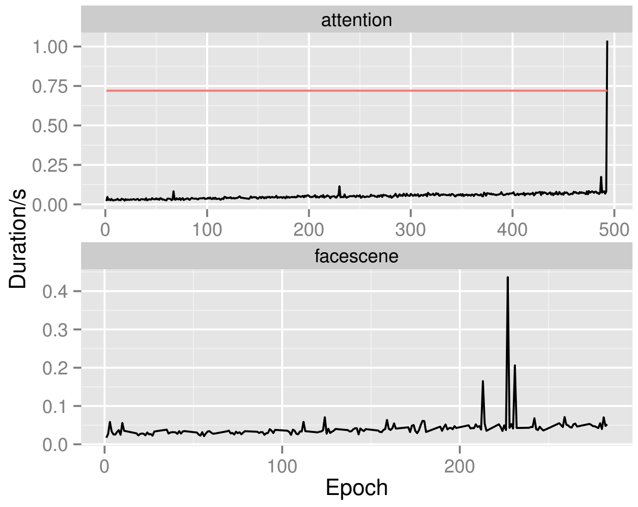

Fig. 7. Duration of classification.

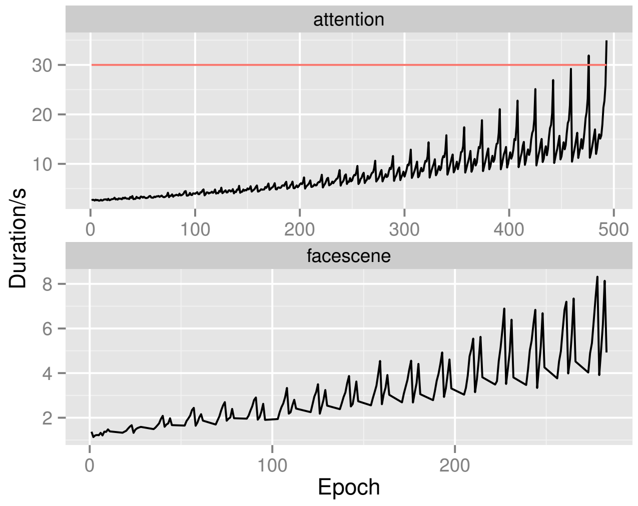

Fig. 8. Duration of voxel selection and model update.

of the last two subjects in the *attention* dataset, where the system has accumulated substantial historical data. According to the scalability feature of our system shown in Section V-D, we believe that we are able to meet the time requirement by adding more worker nodes into the system.

It is also worth noting that although generally the system takes longer and longer time to finish one round of voxel selection as the data scale grows, the running time dropped periodically in the middle of each subject. We attribute this to the implementation of the *sgemm* routine of the Intel MKL library we used in the SVM kernel matrix computation. Specifically, we invoked *sgemm* to compute the similarities between the newly normalized correlation samples and all the existing correlation samples depicted in Figure 6c. We consider that the actual implementation of *sgemm* changes as we increase of number of rows of one matrix (corresponding to the number of normalized correlation samples) in the matrix multiplication, which affects the overall performance.

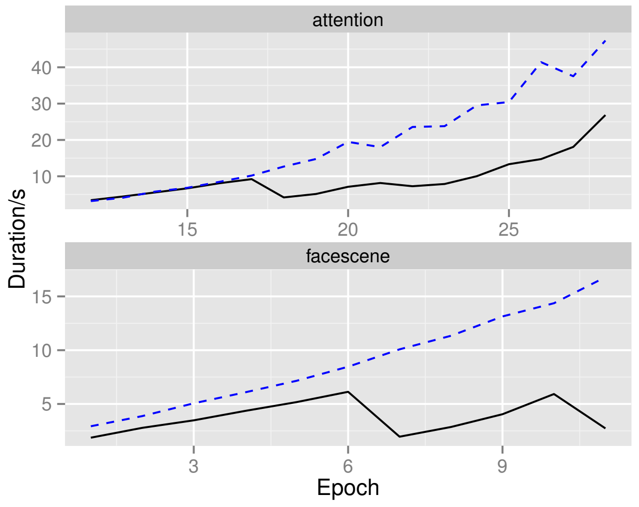

Fig. 9. Voxel selection time for rtFCMA (black) and batchFCMA (blue, dashed) during the last epoch of each dataset.

C. Performance Gain of Incremental Voxel Selection

We compared the performance gain we achieved using the incremental voxel selection algorithms. Our incremental voxel selection took advantage of the characteristics of data stream by accumulating and processing the data when they are coming in. On the other hand, the offline algorithm, although fully optimized, had to read in and process all the data in every process, which introduced a lot of overhead in the real-time context.

Figure 9 shows the performance comparison between the voxel selection of rtFCMA and batchFCMA in the last subject of both datasets. Notice that in the batch version, we excluded the time it took to read data in from disk and broadcast them to all worker nodes. The incremental voxel selection works 1.8x-6.2x faster than the batch version at the end of these two datasets. From the figure we observed a running time drop after some epochs for rtFCMA, which can be explained by the MKL implementation issue we discussed in Section V-B. The *facescene* dataset was affected more significantly because the portion it took to compute using MKL is relatively larger than the *attention* dataset.

D. Scalability of Incremental Voxel selection

We studied the speedup of the incremental voxel selection in our system by varying the number of worker nodes used in the voxel selector.

Figure 10 shows the speedup of the incremental voxel selection when processing the first 5 subjects in *facescene* dataset when using 12, 24 and 48 compute nodes. In average, we achieved 1.8x and 3.9x speedups using 48 nodes comparing to using 24 nodes and 12 nodes, respectively. This shows that as the scale of the real-time experiment increases, our system is able to process the data in real-time as long as more compute nodes are involved.

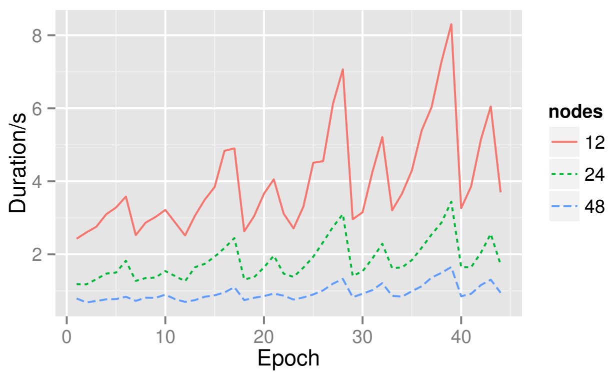

Fig. 10. Scalability of rtFCMA using 12, 24, and 48 compute nodes

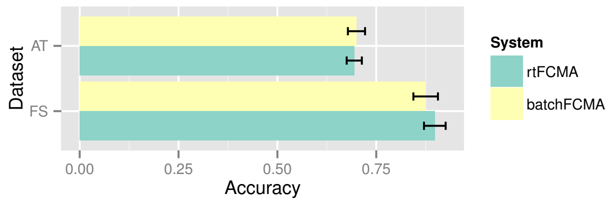

Fig. 11. Accuracy comparison between rtFCMA and batchFCMA. Mean of n-fold cross validation. Standard error of the mean error bars. n =number of subjects, 18 for *facescene* (FS), 30 for *attention* (AT).

E. *Effectiveness of rtFCMA System*

To verify the effectiveness of our system, we compared the accuracy results it generated on both datasets with the results batchFCMA generated using leave-one-subject-out cross validation. In rtFCMA pipeline, we sent the data brain volume by brain volume to the system to select voxels and train a model using $N-1$ subjects (where N is the number of subjects in a dataset) and use this model to test on the last subject. We did this for N times to allow all subjects to be tested. In batchFCMA, we replicated the pipeline used in [10]. The top-500 voxels were used in both cases.

Figure 11 shows the accuracy results of both datasets produced by rtFCMA and batchFCMA. We can see that our rtFCMA system got very similar results in both datasets to batchFCMA, which verifies the correctness of the rtFCMA system. The results are slightly different because in the correlation normalization steps, rtFCMA can only normalize the data based on what it received so far, while batchFCMA used all data within the same subject to do the normalization.

VI. Related work

In this section, we discuss some of the important related work on real-time processing of fMRI data for neuroscience, distributed stream processing, and parallel and distributed machine learning.

Though modern rtfMRI systems are capable of producing full brain scans every 1-2 seconds, analysis of these scans is typically performed offline. Incremental analysis of real-time functional magnetic resonance imaging is a very recent advancement, motivated in part by a surge in neurofeedback research [8]. In one recent study [1], a whole brain-classifier was trained on single-subject voxel activity patterns by batch processing the data (on a single machine) between experimental runs while the subject was in the scanner. The classifier was used to predict whether a subject was attending to a face or a scene within a composite image presented to them and to reinforce the desired attentional behavior by changing the mixture of the composite stimulus. rtFCMA extends the features considered by the classifier to full-brain correlation patterns ($O(n^2)$ features instead of $O(n)$), automatically selects the voxels that are most predictive, and adds the ability to incrementally train the classifier so that runs proceed faster with less down time. None of this would be possible without a parallel and distributed implementation of the algorithms.

Of course, many frameworks for stream-based parallel and distributed machine learning exist. These are largely memory-based systems that are efficient at iterative processing on streams. Apache Spark supports Spark Streaming as a means of joining stream data with historical data or performing window-based operations on the stream. It chops a data stream up into mini-batches and processes these. Recently, the Spark machine learning library, MLlib, was extended with streaming versions of linear regression and the k-means clustering algorithm [19]. Spark's primarily abstraction is the Resilient Distributed Dataset (RDD), an immutable collection that provides fault tolerance but is inefficient at dealing with small changes to large data structures (e.g., updates to a small number of vectors in the correlation matrix). Distributed stream processing engines (DSPEs), like Apache Storm [20], S4 [21], and Samza [22], have emerged that are designed from the ground up for stream processing data in a reactive manner, one update at time. The Apache incubator project SAMOA (Scalable Advanced Massive Online Analysis) provides a machine learning framework and library that utilizes DSPEs but abstracts away the execution engine.

Like some modern DSPEs, rtFCMA uses the highly concurrent and resilient Akka runtime [13] to distribute messages (Akka is largely based on and inspired by Erlang [23]), and like SAMOA we represent our algorithms with a directed graph of nodes that communicate using messages. But, to the best of our knowledge, rtFCMA is the first to implement any form of stream-based machine learning training on functional MRI data and the first streaming implementation of full correlation matrix-based classification for any application.

The incremental voxel selection algorithm that we describe is more generally a form of feature selection. Feature selection is a very common technique in machine learning, often used to increase model accuracy, improve generalization of the model, and to speed up model training.

For incremental training, the focus is on accuracy and generalization. Clustering techniques, component analysis, spectral transforms, and other methods are used broadly for fMRI data classification [24]. Though typically implemented as offline algorithms, many have incremental versions that can scale with parallel and distributed computing resources. However, data-driven feature selection using pairwise voxel correlations over the entire full brain is unique to FCMA [10] and more computationally challenging to implement incrementally than the classic techniques due to the large amount of memory and number of SVM models involved. As far as we know, our 50-machine cluster instantiation of the pipeline is the most parallelized feature selection model ever applied to fMRI data and the first use of incremental classifiers on this data.

This paper is concerned with the processing stages, algorithm design, and aspects of the parallel and distributed system architecture. It does not consider other important considerations for the construction of a practical real-time cloud system, such as the programming model, application isolation, dynamic resource allocation, and fault tolerance.

VII. Conclusion

In this paper, we propose a distributed system for the real-time analysis of fMRI data and provide a proof-of-concept of an important, computationally-intensive neuroimaging application. Our system implements the first incremental version of full correlation matrix analysis, allowing real-time analysis of patterns of correlations in brain activity and neurofeedback that is driven by this analysis. It is able to update a full-correlation model in a few seconds and classify in hundreds of milliseconds to keep pace with rtfMRI scanners, without sacrificing the classification accuracy of offline approaches. Our system can be deployed as a cloud service, which will allow neuroscientists around the world to conduct their own real-time experiments involving incremental full-brain feature selection, model training, and classification. As ours is the first known effort to build such a system, there remains a great detail of future work, including the construction of additional real-time pipelines for other types of analyses that are not currently possible. From a systems perspective, we will focus on making the system more robust by introducing fault tolerance and fine-grained resource allocation. In the end, we believe this distributed incremental learning system will help accelerate the pace of discovery in fMRI-based neuroscience research.

References

[1] M. T. deBettencourt, J. D. Cohen, R. F. Lee, K. A. Norman, and N. B. Turk-Browne, "Closed-loop Training of Attention with Real-time Brain Imaging," *Nature Neuroscience*, vol. 18, no. 3, pp. 470–475, 2015.

[2] N. Weiskopf, K. Mathiak, S. W. Bock, F. Scharnowski, R. Veit, W. Grodd, R. Goebel, and N. Birbaumer, "Principles of a brain-computer interface (BCI) based on real-time functional magnetic resonance imaging (fMRI)," *Biomedical Engineering, IEEE Transactions on*, vol. 51, no. 6, pp. 966–970, 2004.

[3] R. Christopher deCharms, "Applications of real-time fMRI," *Nature Reviews Neuroscience*, vol. 9, no. 9, pp. 720–729, 2008.

[4] S. M. LaConte, "Decoding fMRI brain states in real-time," *Neuroimage*, vol. 56, no. 2, pp. 440–454, 2011.

[5] J. J. Yoo, O. Hinds, N. Ofen, T. W. Thompson, S. Whitfield-Gabrieli, C. Triantafyllou, and J. D. Gabrieli, "When the brain is prepared to learn: enhancing human learning using real-time fMRI," *Neuroimage*, vol. 59, no. 1, pp. 846–852, 2012.

[6] D. D. Leeds, J. A. Pyles, and M. J. Tarr, "Exploration of complex visual feature spaces for object perception," *Frontiers in computational neuroscience*, vol. 8, 2014.

[7] K. Shibata, T. Watanabe, Y. Sasaki, and M. Kawato, "Perceptual learning incepted by decoded fMRI neurofeedback without stimulus presentation," *science*, vol. 334, no. 6061, pp. 1413–1415, 2011.

[8] J. Sulzer, S. Haller, F. Scharnowski, N. Weiskopf, N. Birbaumer, M. L. Blefari, A. Bruehl, L. Cohen, R. Gassert, R. Goebel *et al.*, "Real-time fMRI neurofeedback: progress and challenges," *NeuroImage*, vol. 76, pp. 386–399, 2013.

[9] N. B. Turk-Browne, "Functional interactions as big data in the human brain," *Science*, vol. 342, no. 6158, pp. 580–584, 2013.

[10] Y. Wang, J. D. Cohen, K. Li, and N. B. Turk-Browne, "Full correlation matrix analysis (FCMA): An unbiased method for task-related functional connectivity," *Journal of neuroscience methods*, vol. 251, pp. 108–119, 2015.

[11] Y. Wang, M. J. Anderson, J. D. Cohen, A. Heinecke, K. Li, N. Satish, N. Sundaram, N. B. Turk-Browne, and T. L. Willke, "Full Correlation Matrix Analsis of fMRI Data on Intel® Xeon Phi™ Coprocessors," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '15, Nov 2015.

[12] C. Hewitt, P. Bishop, and R. Steiger, "A Universal Modular ACTOR Formalism for Artificial Intelligence," in *Proceedings of the 3rd International Joint Conference on Artificial Intelligence*, ser. IJCAI'73. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1973, pp. 235–245.

[13] J. Bonér *et al.*, "Akka," http://akka.io.

[14] G. Schaefer *et al.*, "Scientific Transparency," https://github.com/scitran.

[15] J. Platt, "Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines," Microsoft Research, Tech. Rep. MSR-TR-98-14, Apr 1998.

[16] N. Al-Aidroos, C. P. Said, and N. B. Turk-Browne, "Top-down attention switches coupling between low-level and high-level areas of human visual cortex," *Proceedings of the National Academy of Sciences*, vol. 109, no. 36, pp. 14 675–14 680, 2012.

[17] J. Hutchinson, Y. Wang, and N. Turk-Browne, "Decoding the locus of attention from the full correlation matrix of the human brain," in *Society for Neuroscience*, ser. SfN '14, Nov 2014.

[18] S. Choy, B. Wong, G. Simon, and C. Rosenberg, "The brewing storm in cloud gaming: A measurement study on cloud to end-user latency," in *Network and Systems Support for Games (NetGames), 2012 11th Annual Workshop on*, Nov 2012, pp. 1–6.

[19] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen *et al.*, "MLlib: Machine Learning in Apache Spark," *arXiv preprint arXiv:1505.06807*, 2015.

[20] "Apache storm," http://storm.apache.org.

[21] L. Neumeyer, B. Robbins, A. Nair, and A. Kesari, "S4: Distributed stream computing platform," in *Data Mining Workshops (ICDMW), 2010 IEEE International Conference on*. IEEE, 2010, pp. 170–177.

[22] "Apache Samza," http://samza.apache.org.

[23] J. Armstrong, R. Virding, C. Wikström, and M. Williams, "Concurrent programming in erlang," 1993.

[24] C. Wu, "Feature selection for fmri classification," Project report, Carnegie Mellon Unversity, 2006.