High-Performance Computing Framework Based on Distributed Systems for Large-Scale Neurophysiological Data

Mohsen Hadianpour
School of Cognitive Sciences, Institute for Research in Fundamental Sciences, Tehran,
Ehsan Rezayat
School of Cognitive Sciences, Institute for Research in Fundamental Sciences, Tehran,
Mohammad-Reza Dehaqani (dehaqani@ut.ac.ir)
Cognitive Systems Laboratory, Control, and Intelligent Processing Center of Excellence (CIPCE), School of Electrical and Computer Engineering, College of Engineering, University of Tehran, Tehran,

Research Article

Keywords: Large-scale neural data, High-performance computing, Spike sorting, Distributed computation

Posted Date: January 6th, 2021

DOI: https://doi.org/10.21203/rs.3.rs-136986/v1

Abstract

Due to the significantly drastic progress and improvement in neurophysiological recording technologies, neuroscientists have faced various complexities dealing with unstructured large-scale neural data. In the neuroscience community, these complexities could create serious bottlenecks in storing, sharing, and processing neural datasets. In this article, we developed a distributed high-performance computing (HPC) framework called `Big neuronal data framework' (BNDF), to overcome these complexities. BNDF is based on open-source big data frameworks, Hadoop and Spark providing a flexible and scalable structure. We examined BNDF on three different large-scale electrophysiological recording datasets from nonhuman primate's brains. Our results exhibited faster runtimes with scalability due to the distributed nature of BNDF. We compared BNDF results to a widely used platform like MATLAB in an equitable computational resource. Compared with other similar methods, using BNDF provides more than five times faster performance in spike sorting as a usual neuroscience application.

Introduction

The goal of current technologies in neuroscience is to measure neural activity simultaneously from a large number of neurons (up to all neurons in the brain) for long periods of time. These technologies produce massive datasets, resulting bottlenecks and complexities in the neuroscience field. For example, one hour of neural recording, using a microelectrode array with 1000 electrodes (sites), with a typical sampling rate of 40 kHz, would approximately produce a 30 GB dataset. For longer recording sessions, the size of this dataset will increase dramatically. Two-photon imaging provides very big spatiotemporal datasets of small animal's brains such as zebrafish or mouse in the order of 1 TB[1,2]. Indeed, single-machine memory could not handle neural data at these scales. Also, simple processing on such a huge data with widely used platforms, e.g., MATLAB, is inconvenient and nontrivial[3]. Furthermore, sharing neural datasets across the community needs to have a standard data structure in order to satisfy both the data generators and the data consumers[4]. In the new future, the neuroscience community will experience a culture shift in sharing and organizing data while using open-source tools[3–6]. Moving toward high-performance computing (HPC) frameworks leads to leveraging computational power, memory, and storage capabilities offered by their resources. It could significantly influence the neurophysiological community to overcome existing big data bottlenecks[4].

HPC systems have had various applications in other aspects of neuroscience [5,7–9], but a few studies have stated the importance of using the HPC system for dealing with neuronal data[3,4,6,10–12]. HPC system provides an opportunity to perform simulation of cortical circuitry on a large scale (e.g., almost 31,000 neurons with 37 million synapses)[9] dataset, handling large-scale neuroimaging dataset[7,8] and processing massive two-photon imaging datasets[5]. These distributed frameworks were based on Spark cluster-computing framework. However, most of HPC based systems proposed in neuroscience have focused on the processing side and ignored problems like standardized formats and efficient I/O operations. They also did not propose any structure for neurophysiological datasets. In this article, we introduce a general

and flexible distributed framework called BNDF (Big Neuronal Data Framework) (Fig. 1). BNDF is implemented in Scala, based on Apache Spark cluster-computing framework. The advent of the big data paradigm mainly started with the introduction of large distributed computing *clusters'* and MapReduce' programming model[13]. Today the widely used MapReduce engine is developed in the open-source Apache Hadoop ecosystem[14]. Most of the statistical and machine learning modeling methods could be implemented using MapReduce, making it useful for neuronal data analysis. Hadoop distributed file system (HDFS) operates on disk. Since the MapReduce interacts with the HDFS, most iterative algorithms and computations could be slow which can be a bottleneck. The open-source distributed computing framework, Apache Spark, considers an abstraction called resilient distributed datasets (RDDs)[15]. RDDs could be cached and distributed in the memory across all nodes in the cluster, performing iterative operations much faster than Hadoop MapReduce[15].

BNDF is general and flexible enough that it could cover a variety of methods and experiments. It also addresses both the data generators and the data consumer's requirements. As for the data processing, BNDF could cover most of the widely used algorithms and models in the neuroscience community. We performed a spike sorting procedure on a big neural data as one of the widely used processing procedures in neuroscience. In an equitable computational resource state, we found five times smaller runtimes compare to commonly used platforms in the neuroscience community.

Results

Architecture and design

We developed the BNDF framework in an extendable way for processing electrophysiological neuronal datasets (Fig. 1). The BNDF concept generally is based on two main parts, "Data Ingestion" and "Data Processing". Data Ingestion describes procedures related to importing recorded neuronal data from various recording devices and transforming the data into the structured format provided by the BNDF. Currently, BNDF uses Mat files as raw input data; however, the flexibility of BNDF design lets the user work with other file formats. The data Loader module is created for handling required operation in the data ingestion procedure; which we will describe in detail in the later parts. In the case of Data Processing, Spark's elegant and flexible APIs provides various ways of processing data. Most distributed data processing operations are based on Spark libraries, Spark SQL, Spark Streaming, MLib, and GraphX. These libraries are implemented on top of the Spark core. Spark SQL provides SQL-like operations and optimizes queries. Spark Streaming, as its name represents, provides streaming operations. MLib is designed specifically for optimized various machine learning and statistical modeling methods. GraphX library is used for large-scale graph processing. With a combination of the functionality provided by spark libraries, numerous neuroscientific processing could be done efficiently in a distributed manner. For example, with a combination of the Sparks MLib and GraphX libraries, an efficient model could be developed for functional connectivity analysis in huge neuronal ensembles. Spark modules are accessible with different language APIs, Scala, Java, Python, and R. Each API has different features,

supports, and flexibilities. Since, Spark is written in Scala language, this API has the most support, efficiency, and flexible functionality. However, both Java and Scala shares JVM as a compiler and provide similar functionality. Other APIs are evolving, and Python is extended more in recent years. In the BNDF architecture, results can be retrieved directly, however, both Apache zeppelin and Thunder can provide an abstract layer for retrieving results from Spark.

Framework structure

In the BNDF, we proposed the idea of storing data in two separate standard structures. The first one is a nested JSON-structured meta-data stored in the MongoDB database. The second one is the flexible columnar data structure, Apache Parquet[16], stored on top of HDFS as a fast, horizontally-scalable, and storage-efficient massive data ingestion. These structures provided processing data efficiently without caching all of the data in memory. In this way, all processing operations could communicate directly with HDFS, avoid extra I/O time-wasting, and avoid unnecessary caching. Data Loader module provides capabilities for creating efficient structures for addressing both data generators and data consumer's needs. We consider all the operations related to getting the input data and create a standard structure (Fig. 2). The BNDF proposed a tree-like structure for input in which each node can be nested (parent) or not (child). We consider some constraints on the input file based on the output structure, meta-data, and raw-data. We consider two constraints for creating metadata; first, the matrices dimensions should be less than two (vector). Second, the length of the vectors should be equal to or less than 100. For raw-data we assume that the matrix dimension is (T * 1), which T is the maximum signal time index, and T is greater than 100. In the core of the Data Loader module, we developed the SchemaCreator package, which creates desired schemas for raw-data and meta-data (Fig. 2). The SchemaCreator generates two different schemas for raw data and metadata via two classes called metaDataSchemaCreator and channelDataSchemaCreator. The metaDataSchemaCreator class generates a dynamic multilevel nested JSON structure based on the raw file structure shown in Fig. 2. All generated meta-data should have some common fields, but additional fields are not limited and could vary from different recording devices or experiments.

The channelDataSchemaCreator class generates a schema for the raw channel data in a columnar structure. One important difference between these two structures is that JSON is nonhomogeneous (which fits well for metadata structure) and columnar structure is homogeneous data structures. The channelDataSchemaCreator generates raw channel data through the two other classes, rawDataSchemaCreator and eventDataSchemaCreator. MAT files generated by the experimenters could contain either channels, events information, or some similar conceptual structures. Either way, it contains various metadata information that needs to be extracted and stored. This procedure is controlled by the metaDataSchemaCreator class. Event files stores information about task and various stimuli data with time. The eventDataSchemaCreator generates a columnar structure based on input events MAT files. Technically, the rawDataSchemaCreator class operates similarly to the eventDataSchemaCreator class, but it considers only for massive neuronal data. The detailed algorithm of the SchemaCreation package and its so-called classes are available in Supplementary information. It is important to declare that the

data and meta-data storage in the BNDF could be reachable from all experimenters and data analysts and provides a unique set-up for sharing data across the neuroscience community.

Setup and installation

We deployed BNDF on a private cluster with 10 nodes. Each node had 20 cores CPU and 50 GB of RAM. In the cluster, we provided Hadoop and Spark in stand-alone mode with high availability enabled along with other BNDF's requirements described at cluster guide. Two nodes in the cluster are configured for Spark and Hadoop manager nodes, making processing handled by the other 8 nodes. We also configured BNDF alongside MATLAB on a single Linux machine with 80 cores CPU and 256 GB RAM. Spark version 3, Hadoop version 3.2.1, and MATLAB version R2020a are used for evaluating tests and benchmarks. Most of HPC systems proposed before focused on the processing-side and ignored problems like standardized formats, efficient load, and efficient writes. Here in BDNS, to overcome the mentioned problems, we developed two modules Data Loader and Spike Sorter.

Analysis and implementation

We used three datasets with different sizes, generated from the spiking activity of multiple channels. The first data set has 30 GB storage size, generated from spiking activity of 2 channels. The second data set has 90 GB storage size, generated from the spiking activity of 10 channels. The third and the last data set is the largest data set with 250 GB storage size which is generated from spiking activity of 18 channels. BNDF module, data loader, process these datasets, and store structured data in a storage-efficient format in HDFS. By storing data set as Parquet file with Snappy compression format, three data set storage reduces to 10, 30, and 50 GB. The achievement of using Parquet file with Snappy compression format is decreased in size of structured dataset compare to raw file data. Decreasing in the storage size is roughly in order of 3. However, this factor is increased non-monotonically with data size. Moreover, runtime of BNDF Data Loader module decreased by an increase in the number of nodes of the cluster (Fig. 3). As the distributed nature of BNDF, the scalability consequences faster data loading, saving, and various I/O operation.

Large-scale data application

We examined BNDF processing performance by implementing spike sorting module, mostly used preprocessing on neuronal datasets. We developed spike sorting module, called BNDF Sorter, in two sub-modules (Fig. 4a). These modules were built on top of sparks SQL, and MLib libraries, respectively (See Methods). The first sub-modules include thresholding (Thresh), windowing (Window), transforming (Trans) pipelines for detecting spikes, and cerates features for spike sorting. The second sub-module contains spike sorting using principal component analysis (PCA) for feature extraction, gaussian mixture model (GMM) for clustering, and bayesian information criteria (BIC) pipelines[17]. We applied Spike Sorter module on three datasets with the required initial value for each pipeline (See Methods). By collecting benchmark results, we observed that scalability has drastic improvement in both BNDF modules runtime and the aforementioned effect grows with data size (Fig. 4b). The runtime difference between 8 nodes

and 2 nodes for three data set is 7.7, 49.85, and 91.67 minutes respectively. By comparing distributed processing provided by the BNDF and MATLAB, our testing result clarifies that spike sorting procedure runtime could improve more than 5x. For example, for data set with 10 GB size, MATLAB runtime took 87.2 minutes while BNDF runtime took 16.9 minutes (Fig. 4c). We observed that BNDF spike sorter runtime, even for data set 90 GB (44.26 minutes) is lower than data set 30 GB processed by MATLAB (87.2 minutes total and 49.38 minutes exclude load time) (Fig. 4c).

Discussion

In this article, we introduced BNDF, as a distributed and scalable framework. We proposed a standard data structure for neurophysiological datasets, in order to organize and share datasets more easily between data generators and data consumers in different labs or institutes. In the neuroscience community, BNDF is the first distributed and scalable framework for neurophysiological datasets. It is possible to handle big data generated by various experiments, recording devices, or other existing sources of neuronal data generation. New improvements in multiarray recording generate massive neuronal data in near future, then, storing data in a standard and storage-efficient structure and processing data needs scalable, distributed, and flexible foundation, provided by HPC. By introducing BNDF in this article, we focused on how such an HPC framework could have a positive impact on such big datasets.

We developed two primary modules in BNDF, Data Loader, and Spike Sorter. BNDF has advantages compared to other HPC based frameworks. Firstly, BNDF focus on efficient and flexible storage of data and meta-data along with efficient data processing, while others just focused on processing data[5]. Secondly, in BNDF, data can be read in a distributed manner and it is much faster than loading data locally and caching with Apache Spark[13–15].Third, BNDF is developed in a general way with the capability of extension in the future, such that, other introduced HPC systems could be built and configure on top of BNDF. For example, Thunder[5] could be used on top of BNDF, with a small modification. It generalizes Thunder in a compatible manner. Moreover, reading data from HDFS is not possible with Thunder[5]. All data would read locally and cached with spark before processing. In BNDF we avoid unnecessary caching and we implement distribute read from HDFS through our standard data structure created by the BNDF Data Loader module.

BNDF also provides efficient large-scale data processing. Since spike sorting is a challenging and time-consuming process, we developed Spike Sorter to perform spike sorting on large-scale recording data. We determined that BNDF's Spike Sorter module operates faster with an increasing number of nodes. BNDF provides the possibility for processing algorithms like spike sorting on the massive neuronal data set, where it was not possible with a single machine platform like MATLAB. Also, due to the limitation of the available resource and cluster environments, we could only increase scalability by a factor of 2 up to 8 nodes, however, the impact of scalability is much higher when increasing nodes by a factor of 10. BNDF is at its early stages and it is flexible in a way that we could extend, or add new modules (features) to it easily. BNDF has Spike Sorter module for processing data now, however it has a more sophisticated

structure compare to other HPC-based framework[5]. Currently, BNDF operates as spark-submit batch jobs and it is only available in Scala API, but in future works, we could extend BNDF API to higher-level languages like Python, and R for ease of use. As for BNDF functionality, we could enhance its capability to run in a streaming manner and real-time processing data, in this way Spike Sorter module could operate as Online Spike Sorter. A variety of post-processing and coding (decoding) algorithms could be implemented in BNDF easily. For example, a complex, graphical functional connectivity method in the neuronal population level.

Methods

Summary

The detail of our work, examples and the source codes are available at our Gitlab, and documentation for quick start of BNDF are available here.

Deployment

For deployment of BNDF, there are various solutions available. They are varied from the one node standalone cluster to the most scalable, multi-node cluster with high availability. Deployment could be done using private clusters (custom clusters) or using cloud services provided by the various companies. At the sharing level of the lab or universities, privately distributed clusters could be deployed. We developed a cluster for BNDF using docker containers. A detailed procedure for creating a BNDF cluster is available at BNDF cluster guide. BNDF could be deployed through public cloud service providers such as Amazon Webservices (AWS), Microsoft Azure, Google Cloud, and IBM Cloud. In addition to highly scalable structures and resources of the public clusters, data, and meta-data storage in BNDF could be shared between all experimenters and data analysts in the neuroscience community.

We deployed BNDF on a private cluster with 10 nodes from the IPM Grid Center. Each node had 20 cores CPU and 50 GB of RAM. In the cluster, we provided Hadoop and Spark in stand-alone mode with high availability enabled along with other BNDF's requirements described at BNDF cluster guide. Two nodes in the cluster are configured for Spark and Hadoop manager nodes, making processing handled by the other 8 nodes. We also configured BNDF alongside MATLAB on a single Linux machine with 80 cores CPU and 256 GB RAM. Spark version 3, Hadoop version 3.2.1, and MATLAB version R2020a are used for evaluating tests and benchmarks.

Data Format

In this study, we worked on an electrophysiological recording dataset that have been recorded from the inferior temporal cortex of nonhuman[18]. BNDF could be extended to support other recording techniques like EEG, MEG, fMRI, and imaging methods. Since most recording devices generate standard MAT binary

files, we consider MAT files as raw input files, although it could be extended to support other input formats. Based on constraints on MAT files defined in figure 2, data generators should follow this structure during raw file creation.

Data Processing

In BNDF, we developed spike sorting module, as an exemplar processing. BNDF Sorter module was built on top of sparks SQL, and MLib libraries.

Spike Sorting procedure

BNDF Spike Sorter module is mainly based on the approach presented by [19,20]. More specifically it is a distributed version of it but with some slight differences. BNDF Sorter is developed in a way such that processing is divided into ordered pipelines as two sub-modules (Fig. 4a), described in detail as follows.

Separating spikes from raw continuous signal X were done through spike detection by defining a threshold. The estimated value for the threshold doesn't to be small or large to avoid false positive or missing spikes respectively. We chose the threshold value in which considered as an optimal estimation[19,20] as

$$Threshold = c\ \frac{median(|X|)}{0.6745} \qquad Equation\ 1$$

Where c is a constant between 3 and 5 that we select c=3. We extracted the spike time $\{s_{i;}\ i = 1, ..., N\}$, where the raw signal passed from threshold in the spike detection. Here N is the total number of detected potential spikes. The spike waveforms were extracted from 30 samples before of spike time and 50 samples after that (sampling frequency 40 kHz). In this pipeline, we construct the spike matrix based on the windowed spike data evaluated in the previous pipeline. Therefore, we construct spark matrix S with dimension N × 81 as the PCA input in the next pipeline and, also apply the required transformation for transition data between SQL and MLib module. We used PCA for feature extraction and dimensionality reduction for extracting the features and generating input for the clustering method[19]. Our PCA reduces the dimension of spike matrix S, from N × 81 to N × 6. We used Gaussian mixture model (GMM), as an effective clustering method, for spike detection[19,21]. In the literature, GMM is known as the soft clustering method, while other widely used methods like K-means are considered a hard-clustering method. Generally, GMM could be defined as follow[22].

$$f(x) = \sum_{k=0}^{K} \lambda_k \varphi(x; \mu_k, \boldsymbol{\Sigma}_k) \qquad Equation\ 2$$

Where λ_k is mixture weight $(\sum_{k=0}^{K} \lambda_k = 1)$ and $\mu_k\ \boldsymbol{\Sigma}_k$ are average and covariance matrix of the normal maximum likelihood, using EM algorithm[22]. We measured

the goodness of fit GMM method by Bayesian information criterion (BIC) [22]. Here we fit from 2 up 6 clusters and chose the GMM model with the minimum BIC value.

Declarations

Acknowledge

We thank R. Toosi for help developing MATLAB code for spike sorting procedure and discussion, and A. Shahamati and M. Zarei for the proof reading of manuscript. We also thank members of the IPM Turin Cloud Services (turin.ipm.ir) for delivering the compute power to conduct the experiments. This work was supported by the School of cognitive science of IPM.

Author contributions

M. H., E. R. and M.R. A.D. conceived of the project. M. H. developed the analysis library and analyzed the data. E.R. and M.R. A.D. collected the neuronal data. M. H., E. R. and M.R. A.D. wrote the manuscript.

Competing interests

The authors declare no competing interests.

References

1. Ahrens, M. B., Orger, M. B., Robson, D. N., Li, J. M. & Keller, P. J. Whole-brain functional imaging at cellular resolution using light-sheet microscopy. *Nat. Methods*10, 413–420 (2013).
2. Panier, T. *et al.* Fast functional imaging of multiple brain regions in intact zebrafish larvae using selective plane illumination microscopy. *Front. Neural Circuits*7, 65 (2013).
3. Freeman, J. Open source tools for large-scale neuroscience. *Curr. Opin. Neurobiol.*32, 156–163 (2015).
4. Bouchard, K. E. *et al.* High-Performance Computing in Neuroscience for Data-Driven Discovery, Integration, and Dissemination. *Neuron*92, 628–631 (2016).
5. Freeman, J. *et al.* Mapping brain activity at scale with cluster computing. *Nat. Methods*11, 941–950 (2014).
6. Bouchard, K. E. *et al.* International neuroscience initiatives through the lens of high-performance computing. *Computer (Long. Beach. Calif).*51, 50–59 (2018).
7. Makkie, M. *et al.* A Distributed Computing Platform for fMRI Big Data Analytics. *IEEE Trans. big data*5, 109–119 (2019).
8. Boubela, R. N., Kalcher, K., Huf, W., Našel, C. & Moser, E. Big Data Approaches for the Analysis of Large-Scale fMRI Data Using Apache Spark and GPU Processing: A Demonstration on Resting-State fMRI Data from the Human Connectome Project. *Front. Neurosci.*9, 492 (2015).

9. Markram, H. *et al.* Reconstruction and Simulation of Neocortical Microcircuitry. *Cell*163, 456–492 (2015).
10. Landhuis, E. Neuroscience: Big brain, big data. *Nature*541, 559–561 (2017).
11. Chen, Y., Wang, Z., Yuan, G. & Huang, L. An overview of online based platforms for sharing and analyzing electrophysiology data from big data perspective. *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.*7, e1206 (2017).
12. Cunningham, J. P. Analyzing neural data at huge scale. *Nat. Methods*11, 911–912 (2014).
13. Dean, J. & Ghemawat, S. MapReduce: simplified data processing on large clusters. *Commun. ACM*51, 107–113 (2008).
14. Shvachko, K., Kuang, H., Radia, S. & Chansler, R. The hadoop distributed file system. in *2010 IEEE 26th symposium on mass storage systems and technologies (MSST)* 1–10 (2010).
15. Zaharia, M. *et al.* Spark: Cluster computing with working sets. *HotCloud*10, 95 (2010).
16. Melnik, S. *et al.* Dremel: interactive analysis of web-scale datasets. *Proc. VLDB Endow.*3, 330–339 (2010).
17. Toosi, R., Akhaee, M. A. & Dehaqani, M. A. An Adaptive Detection for Automatic Spike Sorting Based on Mixture of Skew-t distributions. *bioRxiv* (2020).
18. Rezayat, E. *et al.* Frontotemporal Coordination Predicts Working Memory Performance and its Local Neural Signatures. *bioRxiv* (2020).
19. Rey, H. G., Pedreira, C. & Quian Quiroga, R. Past, present and future of spike sorting techniques. *Brain Res. Bull.*119, 106–117 (2015).
20. Quiroga, R. Q., Nadasdy, Z. & Ben-Shaul, Y. Unsupervised spike detection and sorting with wavelets and superparamagnetic clustering. *Neural Comput.*16, 1661–1687 (2004).
21. Harris, K. D., Henze, D. A., Csicsvari, J., Hirase, H. & Buzsáki, G. Accuracy of tetrode spike separation as determined by simultaneous intracellular and extracellular measurements. *J. Neurophysiol.*84, 401–414 (2000).
22. Friedman, J., Hastie, T. & Tibshirani, R. *The elements of statistical learning*. 1, (Springer series in statistics New York, 2001).

Figures

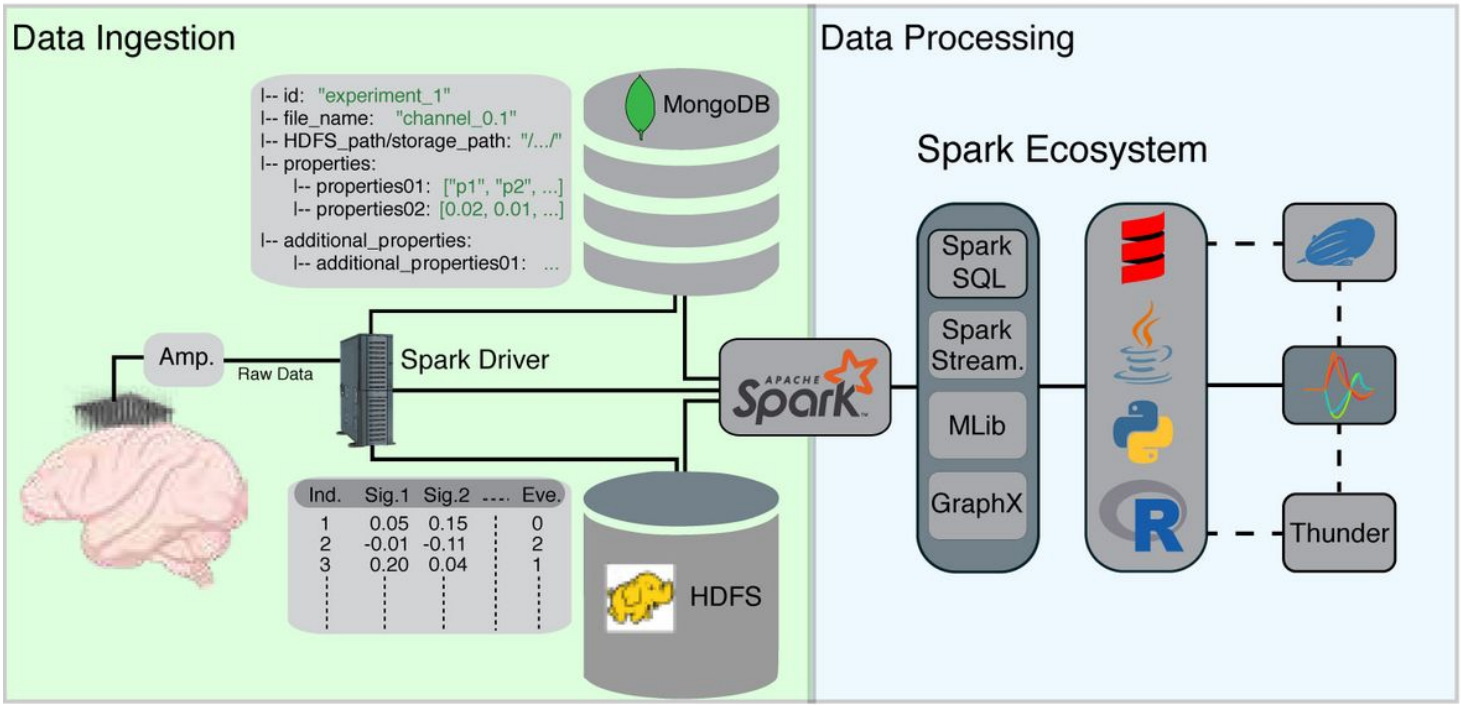

Figure 1

The architecture of the BNDF. The functionality provided by the BNDF is divided into two parts, Data Ingestion (green box) and Data Processing (light blue box). In Data Ingestion, multiarray recorded the raw data of neural activity from the brain. Spark driver gets raw data as input and follows as two tracks. In the top track, a meta-data is extracted from raw files as nested JSON-structured and stored in the MongoDB database. In the bottom track, massive raw data is stored as an efficient columnar structure, in the HDFS. In Data Processing, the structured data in HDFS, and MongoDB were retrieved by Spark for processing with four modules, Spark SQL, Spark Streaming, MLib, and GraphX. Spark modules are accessible with different Language APIs, Scala, Java, Python, and R and related icons were sorted from top to bottom based on features support and flexibilities. The results can be retrieved directly by modules provided by Spark. In the BNDF architecture, both Apache zeppelin and Thunder can provide an abstract layer for retrieving results from Spark.

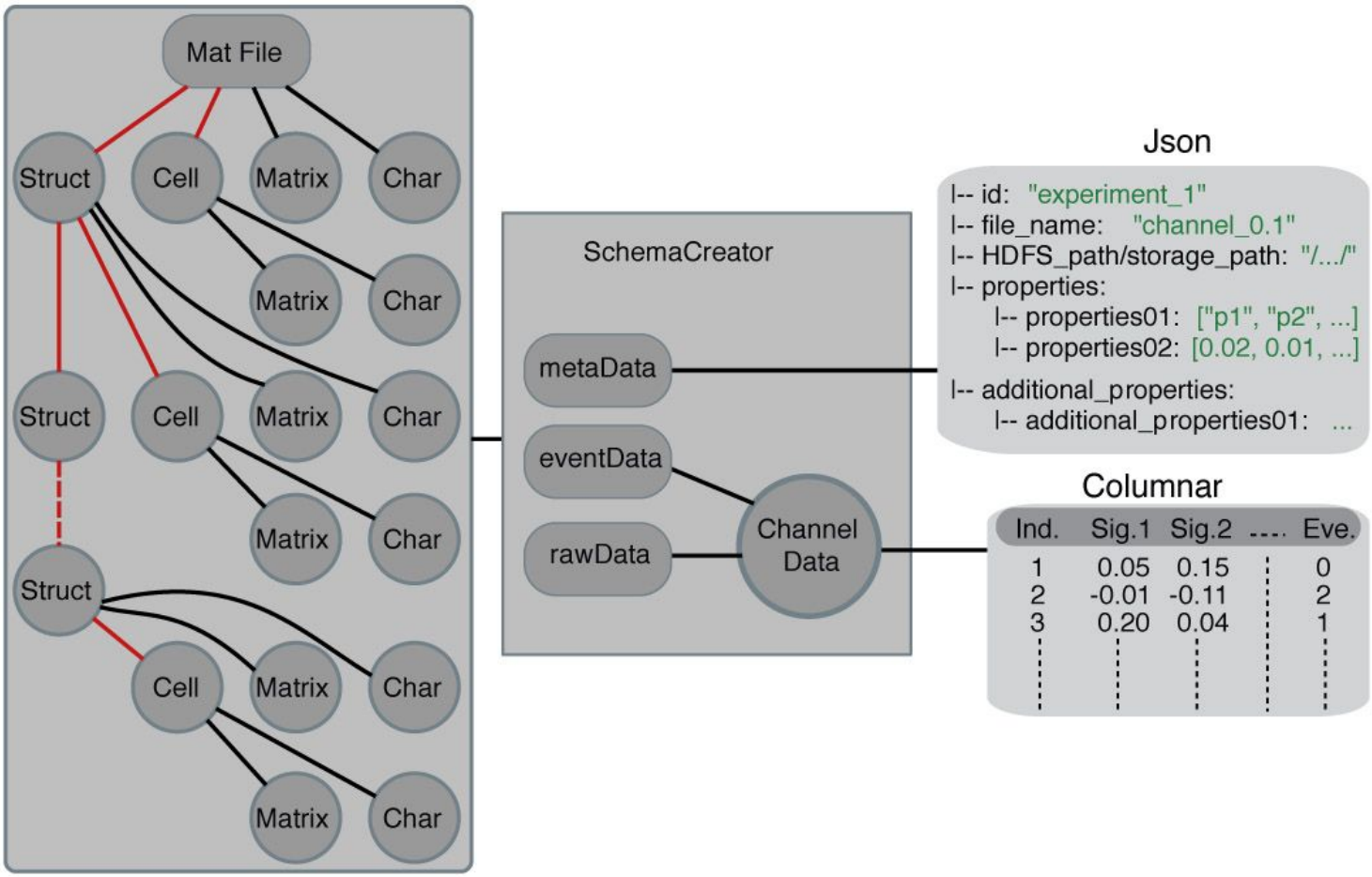

Figure 2

The functionality of BNDF data loader. Raw MAT file tree structure and constraints were showed on the left box. Black arrows correspond to child nodes and red arrows correspond to parent nodes. Raw MAT file consists of Struct, Cell, Matrix, and Char types in the root level of the file. Struct field is capable of n-level nesting. In the Middle, schema creator package constructs a schema for meta-data and raw-data using three subclasses, metaDataSchemaCreator (metadata box), eventDataSchemCreator (eventData box), and rawDataSchemCreator (rawData box) (methods for detail). (Right-hand side) Meta-data were formed as Nested JSON structure. Keys were shown black and values in green colors. Raw-data were formed as Parquet columnar structure.

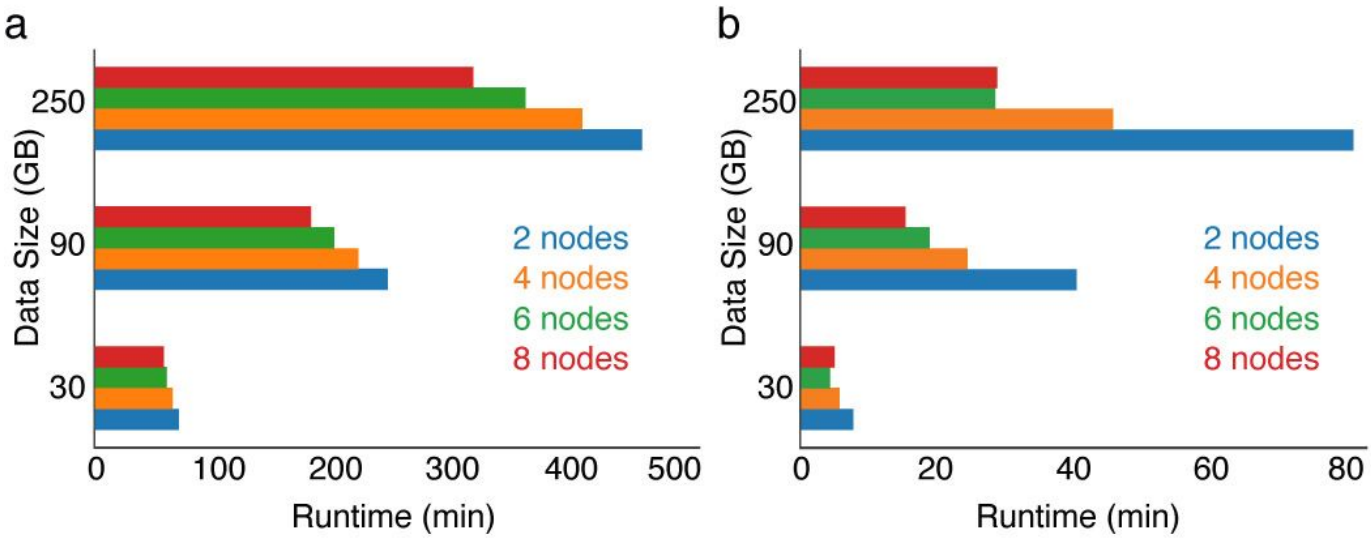

Figure 3

The Performance benchmark of BNDF. Benchmarks are evaluated with 2, 4, 6, and 8 nodes on three different dataset sizes in the cluster environment that each node had 20 cores of CPU and 200 GB RAM. (a) Runtime for all operations handled by the BNDF Data Loader module in the cluster environment. (b) Runtime for saving results in the BNDF setting.

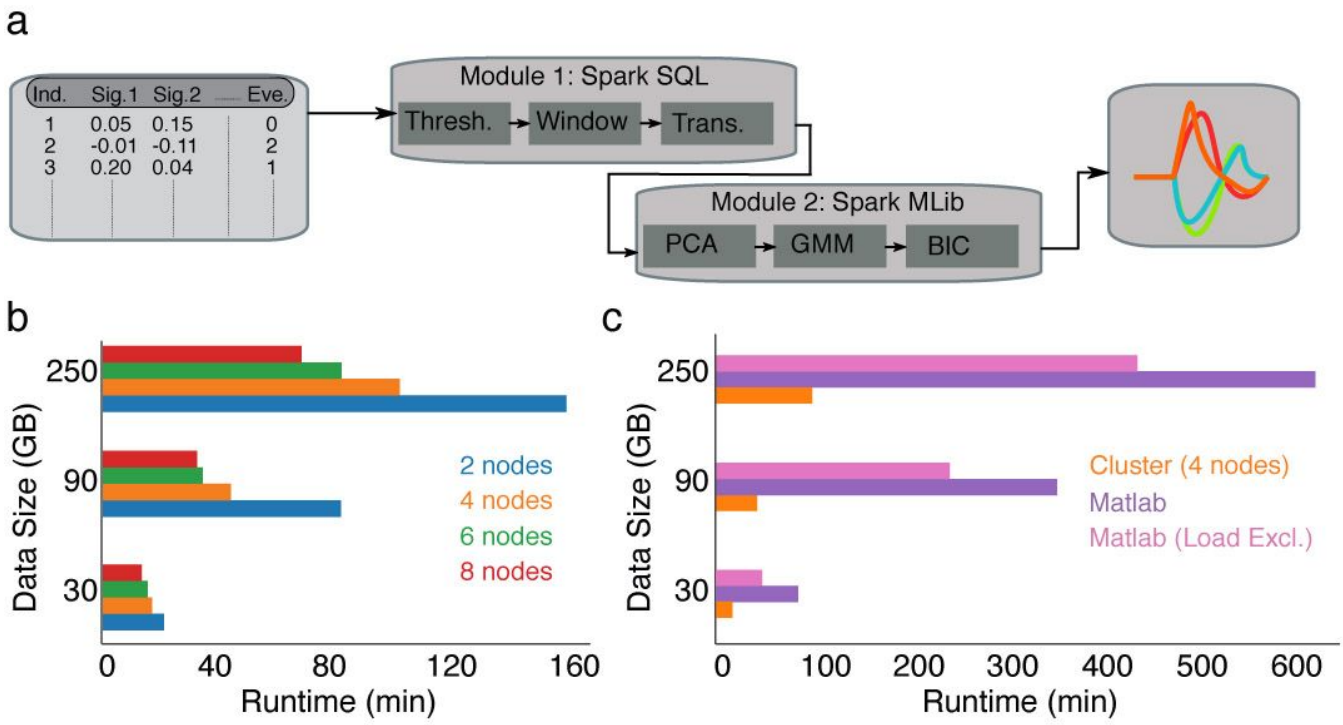

Figure 4

BNDF pipeline for sorting. (a) BNDF sorter module. From left to right, raw data in the columnar structure have entered Module 1 operating under the Spark SQL library and includes Thresholding, Windowing, and transforming pipelines. Module 2 operates under the Spark MLib library and covers PCA, GMM, and BIC (optimizing number of clusters) pipelines. The resulting sorted dataset is retrieved at the end of the module, (b) Performance of the BNDF Sorter module. Benchmarks are evaluated with 2, 4, 6, and 8 nodes on three different dataset sizes, in the cluster environment that each node had 20 cores of CPU and 200 GB RAM, (c) Comparing MATLAB sorter and BNDF sorter performance benchmark. Runtime for BNDF sorter (on the cluster with 4 nodes) and MATLAB installed on a single machine Linux workstation with 80 cores of CPU and 200 GB RAM. The total amount of resources for the BNDF cluster is equal to the single machine Linux workstation machine (both 80 cores of CPU and 200 GB RAM) which used for MATLAB. All pipeline operations described in a are written in MATLAB code with the same configurations for comparison. BNDF reads data as parquet files directly from HDFS, while for MATLAB, CSV files are generated. MATLAB version of spike detection reported for two scenarios, whole operation and only sorting procedure (loading data excluded). For two data set with 90, and 250 GB size MATLAB encounter out of memory (OOM) error. In the best-case scenario based on the runtime provided by BNDF in b, we could assume runtime for two datasets with 90, and 250 GB size are growth by an almost linear rate of 30 GB data set.

Supplementary Files

This is a list of supplementary files associated with this preprint. Click to download.

- Supplementary.pdf