Is Apache Spark Scalable to Seismic Data Analytics and Computations?

Yuzhong Yan, Lei Huang
Department of Computer Science
Prairie View A&M University
Prairie View, TX
Email: yyan@student.pvamu.edu, lhuang@pvamu.edu

Liqi Yi
Intel Corporation
2111 NE 25th Ave.
Hillsboro, OR
Email: liqi.yi@intel.com

***Abstract*—High Performance Computing (HPC) has been a dominated technology used in seismic data processing at the petroleum industry. However, with the increasing data size and varieties, traditional HPC focusing on computation meets new challenges. Researchers are looking for new computing platforms with a balance of both performance and productivity, as well as featured with big data analytics capability. Apache Spark is a new big data analytics platform that supports more than map/reduce parallel execution mode with good scalability and fault tolerance. In this paper, we try to answer the question that if Apache Spark is scalable to process seismic data with its in-memory computation and data locality features. We use a few typical seismic data processing algorithms to study the performance and productivity. Our contributions include customized seismic data distributions in Spark, extraction of commonly used templates for seismic data processing algorithms, and performance analysis of several typical seismic processing algorithms.**

***Index Terms*—Parallel Computing; Big Data Analytics; Seismic Data; Stencil Computing;**

I. Introduction

Petroleum is a traditional industry where massive seismic data sets are acquired for exploration using land-based or marine surveys. Huge amount of seismic data has already been generated and processed for several decades in the industry, although there was no the big data concept at that time. High Performance Computing (HPC) has been heavily used in the industry to process the pre-stack seismic data in order to create 3D seismic property volumes for interpretation.

The emerging challenges in petroleum domain are the burst increase of the volume size of acquired data and high-speed streaming data from sensors in wells that need to be analyzed on time. For instance, the volume size of high dimension such as 3D/4D seismic data and high density seismic data are growing exponentially. The seismic data processing becomes both computation- and data- intensive applications. The traditional HPC programming model is good at handling computation-intensive applications, however, with the continuously increasing sizes and varieties of petroleum data, HPC was not designed to handle the emerging big data problems. Moreover, HPC platforms has been an obstacle for most geophysicists to implement their algorithms on such platforms directly, who demand a productive and scalable platform to accelerate their innovations.

In many the data- and technology-driven industries, big data analytics platforms and cloud computing technologies have made great progress in recent years toward meeting the requirements of handling fast-growing data volumes and varieties. Hadoop [1] and Spark [2] are currently the most popular open source big data platforms that provide scalable solutions to store and process big data, which deliver dynamic, elastic and scalable data storage and analytics solutions to tackle the challenges in the big data era. These platforms allow data scientists to explore massive datasets and extract valuable information with scalable performance. Many technologies advances in statistics, machine learning, NoSQL database, and in-memory computing from both industry and academia continue to stimulate new innovations in the data analytics field.

Geophysicists need an ease-to-use and scalable platform that allows them incorporate the latest big data analytics technology with the geoscience domain knowledge to speed up their innovations in the exploration phase. Although there are some big data analytics platforms available in the market, they are not widely deployed in the petroleum industry since there is a big gap between these platforms and the special needs of the industry. For example, the seismic data formats are not supported by any of these platforms, and the machine learning algorithms need to be integrated with geology and geophysics knowledge to make the findings meaningful.

Are these big data analytics platforms suitable in the petroleum industry? Because of lack of domain knowledge, these platforms have been difficult to use in some traditional industry sectors such as petroleum, energy, security, and others. They need to be integrated and customized to meet the specific requirements of these traditional industry sectors. The paper targets to discuss the gap between the general functionality of the big data analytics platforms and the special requirements from the petroleum industry, and to experiment a prototype of Seismic Analytics Cloud platform (SAC for short) [3, 4]. The goal of SAC is to deliver a scalable and productive cloud Platform as a Service (PaaS) to seismic data analytics researchers and developers. SAC has two main characteristics: one is its scalability to process big seismic data, and the other is its ease-to-use feature for geophysicists. In this paper, we describe our implementation of SAC, experiment with a few typical algorithms in seismic data analytics and computations, and discuss the performance in details.

II. Related Work

The big data problem requires a reliable and scalable cluster computing or cloud computing support, which has been a longstanding challenge to scientists and software developers. Traditional High Performance Computing (HPC) researches have put significant efforts in parallel programming models including MPI [5], OpenMP [6], and PGAS languages [7, 8, 9], compiler parallelization and optimizations, runtime support, performance analysis, auto-tuning, debugging, scheduling and more. However, these efforts mostly focused on scientific computing, which are computation-intensive, while big data problems have both computation- and data-intensive challenges. Hence, these traditional HPC programming models are not suitable to big data problems anymore. Besides scalable performance, tackling big data problems requires a fault-tolerant framework with high-level programming models, highly scalable I/O or database, and batch, interactive and streaming tasks support for data analytics.

MapReduce [10] is one of the major innovations that created a high-level, fault-tolerant and scalable parallel programming framework to support big data processing. The Hadoop [1] package encloses Hadoop Distributed File System (HDFS), MapReduce parallel processing framework, job scheduling and resource management (YARN), and a list of data query, processing, analysis and management systems to create a big data processing ecosystem. Hadoop Ecosystem is fast growing to provide an innovative big data framework for big data storage, processing, query and analysis. Seismic Hadoop [11] combines Seismic Unix [12] with Cloudera's Distribution including Apache Hadoop to make it easy to execute common seismic data processing tasks on a Hadoop cluster. In [13], [14] and [15], some traditional signal processing and migration algorithms are already implemented on MapReduce platform. [16] built a large-scale multimedia data mining platform by using MapReduce framework, where the processing dataset is similar to seismic image. However, MapReduce only supports batch processing and relies on HDFS for data distribution and synchronization, which has significant overloads for iterative algorithms. Furthermore, there is no support for streaming and interactive processing in MapReduce, which becomes the biggest hole for supporting time-sensitive data processing applications.

To reduce the overhead of shuffling data into HDFS and to support more widely used iterative algorithms, there raised several in-memory computing frameworks. Apache Flink [17] is a fast and reliable large-scale data processing engine, which is originally coming from Stratosphere [18]. It provides in-memory data sets, query language, machine learning algorithms and interfaces handling streaming data. Besides providing batch processing function, Flink is good at incremental iterations by pipelining data in an execution engine, which makes it suitable for most machine learning algorithms. Spark [19] is a quick-rising star in big data processing systems, which combines the batch, interactive and streaming processing models into a single computing engine. It provides a highly scalable, memory-efficient, in-memory computing, real-time streaming-capable big data processing engine for high-volume, high-velocity and high-variety data. Moreover, it supports high-level language Scala that combines both object-oriented programming and functional programming into a single programming language. The innovative designed Resilient Distributed Dataset (RDD) [20] and its parallel operations provide a scalable and extensible internal data structure to enable in-memory computing and fault tolerance. There is a very active, and fast-growing research and industry community that builds their big data analytics projects on top of Spark.

However, all these frameworks are built for general propose cases and are focused on data parallelism with improved MapReduce model, and there is no communication mechanism between workers, which does not fit to some iterative seismic algorithms requiring frequent data communication among workers. Both traditional seismic data storage and processing algorithms need big changes to run on MapReduce platform.

With the growing exponentially of the seismic data volumes, how to store and manage the seismic data becomes a very challenge problem. The HPC applications need to distribute data to every worker node, which will consume more time on data transferring. The trend toward big data is leading to transitions in the computing paradigm, and in particular to the notion of moving computation to data, also called near-data-processing(NDP) [21]. In Data Parallel System such MapReduce [22], clusters are built with commodity hardware and each node takes the roles of both computation and storage, which makes it possible to bring computation to data. In [23], it presented an optimized implementation of RTM by experiments with different data partitioning, keeping data locality, and reducing data movement. In [24], it proposed a remote visualization solution by introducing GPU computing into cluster, which could overcome problems of dataset size by reducing data movements to local desktop. In [25], it evaluated the suitability of MapReduce framework to implement large-scale visualization techniques by combining data manipulation and data visualization system on cluster. Besides binary seismic data, there are huge amount semi-structured data and metadata generated at seismic data acquisition and processing. There are some obvious limitations of traditional RDBMS to handle this kind of big data; RDBMS is not flexible to handle different types of data, and there are also scalability and performance limitations on RDBMS. The NoSQL database[26] is designed to be more suitable in such a scenario. Cassandra [27] is a distributed NoSQL database designed to handle large amount of data that could achieve linear scalability and fault-tolerant ability without compromising performance. MongoDB [28] is a document-oriented NoSQL database that casts focus on flexible data model and highly scalability. Redis [29] is a data structure server that provides key-value cache and storage, and it works with in-memory dataset thus could achieve outstanding performance. Based on the characteristics of seismic data, Cassandra could be used to store intermediate data shared by all workers.

With the increasing volume size of seismic data, the algorithms applied on data also become more sophisticated to extract valuable information. Some advanced machine learning algorithms are already used in this area. [30] used artificial neural network (ANN) to predict sand fraction from learning multiple seismic attributes such as seismic impedance, amplitude and frequency. In [31], it set up a model by feeding five seismic attributes and the reservoir thickness to train Support Vector Machines (SVM) and then used it to predict the reservoir thickness. [32] used meta-attributes to train multi-layer neural networks and evaluated the effectiveness of the new generated seismic fault attribute. [33] used Back Propagation Neural Network (BPNN) for the automatic detection and identification of local and regional seismic P-Waves. Petuum [34] is a distributed machine learning framework that is focused on running generic machine learning algorithms on big data sets and simplifying the distributed implementation of programs. Based on characteristics of machine learning algorithms, Petuum provides iterative-convergent solutions that quickly minimize the loss function in a large-scale distributed cluster. Since these frameworks such as Petuum and Graphlab [35] are designed specially for machine learning algorithms, they could get better performance comparing with other general purpose MapReduce frameworks that emphasize on consistency, reliability and fault tolerance, but their programming models are not easy to use comparing with MapReduce.

In summary, although there are already many research projects trying to solve the big data problem, there is still no solution designed specific to seismic data. The MapReduce platform and its predecessors are too common to support complicated seismic algorithms, while some other platforms either emphasize on performance of computation or model optimization in narrow specific area. To make it easy for geophysicists and data scientists processing and analyzing big petroleum data, a new platform is needed by incorporating the advances of big data research into the industry. Such a platform should not only be capable of processing big data efficiently and running advanced analytics algorithms, but also should achieve fault tolerance, good scalability and usability. Seismic data management and distribution need to be implemented in order to allow geophysicists to utilize the platform. Moreover, the common-used seismic computation templates and workflow would be very useful to simplify their work.

III. Implementation

SAC is built on Spark to boost performance by utilizing in-memory computing model. In order to make it easy to use by geophysicists, we developed some high level seismic data processing templates to facilitate the user programming efforts.

Figure 1 shows the overall software stack used in SAC. In this diagram, the operating systems at the first level from bottom could be Unix-like or Windows system running on the virtual machines or bare metals. Above the OS layer, there is a layer that provides compiling and running environment, which includes JRE/JDK, Scala, Python and other native libraries such as OpenCV [36], FFTW [37] etc. In the third layer, there are some common components installed including HDFS, Mesos [38] and YARN [1] used for the data storage and resource scheduling. HDFS is a distributed file system delivered in Hadoop that provides fault-tolerance and high throughput access to big data sets. In SAC, HDFS is used for storing original binary seismic data. The metadata and intermediate data such as seismic attributes are stored in Cassandra database. Resource management in cluster is very important for application scheduling and load balance, and in SAC, Standalone, Mesos and YARN are all supported. In the fourth layer from bottom, it includes the actual computation components: signal and image processing libraries with Java/Scala interfaces; Breeze [39] is a set of libraries for machine learning and numerical computing written in Scala and Java. FFTW is a C subroutine library for computing the Discrete Fourier Transform (DFT) with one or more dimensions in both real and complex data format. There are already some Java FFTW wrappers make it could be used on JVM without giving up performance. SAC chose Spark as the computation platform due to its performance achieved with in-memory computation and its fault tolerance features. The main work of this paper is focus on the second and third layer from top. Seismic Analytics Cloud layer is used for providing SDK and running environment for client applications. SAC Framework plays the most important role in this cloud platform, and it is the bridge of user's applications and running environment on cluster. The template-Based framework provides common programming models for domain specific experts, and the workflow framework connects pieces of algorithms or models into job chains, and run them following the workflow sequence. Visualization is important for user to view results and to generate useful information intuitively. Seismic Applications on the top of stack are mainly developed by end users. There is an user friendly web interface provided by SAC, on which users could view datasets, programming and testing algorithms or running workflow in a convenient way by drag-and-drop of widgets.

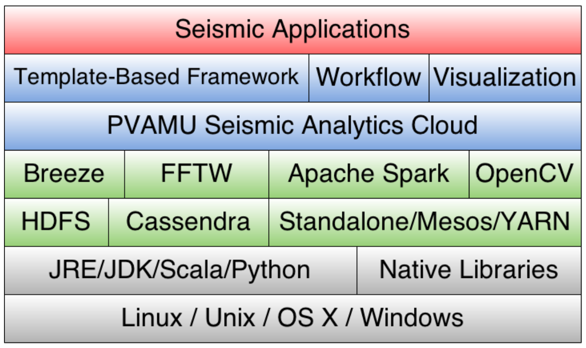


Fig. 1: The Software Stack of Seismic Analytics Cloud Platform

A. Seismic RDD

Resilient Distributed Datasets (RDDs) [20] is core concept proposed by Spark, which is a fault-tolerant collection of elements that can be operated in parallel. RDDs [40] support two types of operations: transformations, which create a new dataset from the existing one, and actions, which return a value to the driver program after running the defined computation on the dataset. Even in parallel programming on cluster, the program still consists of data structure and algorithms. Spark uses the RDD as a common data structure that distributed in multi-nodes, and provides some typical operations as algorithm frameworks in functional language style so that the user could plug in his own algorithms and apply them on RDD. Comparing with traditional parallel programming models such as MPI and OpenMP, the programming on Spark is much simpler. But for geophysicists or data scientists who have no much idea about RDD and functional language, there are still some tedious jobs to do. SAC tried to simplify work by introducing Seismic RDD and Template. Users only need to configure some parameters: the dataset need to process, input and output type of algorithms, then write a piece of codes, after that SAC will generate Seismic RDD, create the template, merge user's codes and run them automatically. Seismic RDD is built from SeismicInputFormat, and besides the basic operations provided by Spark, Seismic RDD also provides some other features: the fine-grain functions on pixel or trace inside each split, transferring RDD from default inline direction to other directions automatically basing on configuration, overiding some operators for easily used by high level applications. The most advantage of the RDD is caching most frequently used data in memory, thus improving performance of some iteration algorithms drastically.

B. Seismic Data Computation Templates

Essentially, seismic data is a 2D plane or 3D volume composed by traces. The data type of trace data is Float type in IEEE floating-point format or IBM floating-point format. Classical signal processing or image processing algorithms have been widely used for processing seismic data. The grain size of input/output data could be sample point, trace, 2D plane or 3D volume. The relationship between volume size of input and the other one of output is shown in Figure 2, in which solid circle (input data) or hollow circle (output) indicates one point, one trace, one plane or even one volume. The relationship could be 1 to 1 (Figure 2 a), N to 1 (Figure 2 b) or 1 to N (Figure 2 c). In some case such as median filter, there is overlap between each input split, which could be treated as a special case of 1 to 1, but the overlap edges need to be considered in data distribution. After study of the popular open source seismic data processing packages, signal processing and image processing packages such as SU, Madagascar, JTK [41], Breeze, OpenCV, ImageMagick etc., we define some typical templates in SAC: Pixel pattern, which uses the sub-volume or one pixel as input and output one pixel; Trace Pattern, which uses one trace or several traces as input and output one or more traces; Line pattern, which treats one line or more lines as input and one line or more lines as output; SubVolume pattern, which feeds user's application with a sub-volume and get output from it in sub-volume format. These templates could handle most cases with one seismic data set, but it could not handle other cases with two or more seismic data sets as input because map/flatMap functions can only be applied on one RDD. For the case with two RDDs, we can merge them into one RDD with zip function, and then apply map/flatMap functions on the combined RDD.

Beside these transformations, there are still some other summary operations or actions in Spark such as count, stats or collect etc. Those functions have no definite templates, but are very useful. So SAC provides a free-style template by passing RDD directly to user's application, on which users could call any transformations and actions as required. For some sophisticated models that are difficult to split into sub-tasks or have multiform of input or output, free-style template is also effective.

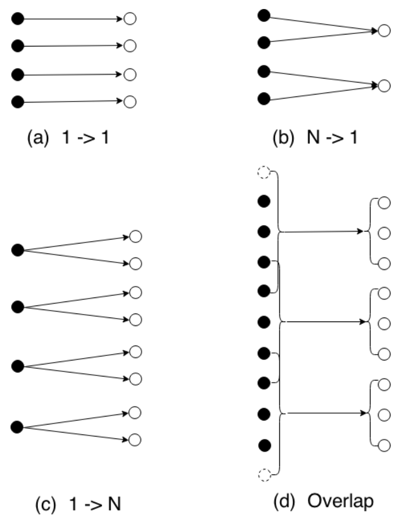


Fig. 2: Relationship of Input and Output in Seismic Data Processing

IV. Experiments

To evaluate the performance of SAC, we setup the experiment environment, develop the SAC framework and run some typical seismic applications on SAC. There are three main layers in SAC: the low-level runtime environment, SAC framework as the middleware and application development interfaces at up-level. The middleware layer of SAC was already discussed in the pervious section. The runtime environment is the base that SAC builds on, and the application development interfaces serve as the entry for users. We chose three typical seismic volume computation algorithms: Fourier Transform, Hilbert Transform, and Jacobi stencil computations as our experiments.

The cluster used for conducting experiments consists of 25 nodes, in which one is the management node and other 24 nodes are computation nodes. Each node in this cluster was equipped with Intel Xeon E5-2640 Sandy Bridge CPU (2.5GHz, 12 Cores or 24 Cores with Hyper-threading support), 64GB DDR3 memory and all nodes are inter-connected with 1GB ethernet. Each node has its own local disk, and also could access disk array through NFS. Following the architecture stated in previous section, we install CentOS 6.5 (Distributed by Redhat) and Oracle JDK 1.8.0_40 on each node. Hadoop 2.2.0, Spark 1.2.1 and other related libraries are also installed on each node. In the configuration of HDFS, the management node was configured as NameNode and other 24 computation nodes as DataNodes. It is similar in Spark: the management node is Master and other computation nodes are Workers. Cassandra was installed on all 24 computation nodes and the first four nodes of them were selected as seed nodes.

The public sample seismic dataset Penobscot [42] was selected as experiment data. The original format of Penobscot dataset is SEGY, and to make it easily processed with Spark, we transfer it into two files: one xml file that saves meta data, and another binary data file with dimension size of 600x481x1501 stores actual 3D data samples. The volume size of the original data file is about 1.7GB, which is not big enough comparing with datasets currently used in oil & gas industry, so we use it synthesize a new 100GB file for verifying algorithms and models on SAC. For some extensive time consuming algorithms, we still use the 1.7GB binary file as test data. Both of xml file and data file are stored on HDFS, so that every node could access them and utilize data locality. The intermediate results are stored in Cassandra basing on requirement and final results are persisted back to HDFS.

A. Fourier & Hilbert Transformation

In the signal and image processing area, Fourier transform (FT) is the most commonly used algorithm. The signal in time domain was decomposed into a series of frequencies through FT, and in the frequency domain, many problems such as filters are easier to perform comparing with in the time domain. Fast Fourier transform (FFT) [43] is an algorithm to compute the discrete Fourier transform (DFT) and its inverse by factorizing the DFT matrix into a product of sparse (mostly zero) factors. There are different implementations of FFT, such as FFTW, OpenCV, Kiss FFT, Breeze etc. FFTW[44] emphasizes performance by adapting to the hardware such as SIMD instructions in order to maximize performance, while Breeze aims to be generic, clean and powerful without sacrificing (much) efficiency. Breeze provides more concise interfaces of 1D/2D Fourier transforms and filtering functions, so we use FFT function in Breeze as the test case by applying it both in sequential codes and parallel codes running on Spark.

The Hilbert transform [45] is important in the field of signal processing where it is used to derive the analytic representation of a signal. A great number of geophysical applications consist in close relation of the Hilbert transform to analytic functions of complex variable [46]. Hilbert transform approach now forms the basis by which almost all amplitude, phase and frequency attributes are calculated by today's seismic interpretation software [47]. JTK already provided the Hilbert transform filter class, so we use it as the test case by apply Hilbert transform filter on each trace of input seismic data set.

B. Jacobi Stencil Computation

Stencil operations are the most common computations used in seismic data processing and reservoir simulation in oil & gas industry, and most of codes in this domain were written in MPI or OpenMP programming models running on large scale clusters or large-scale SMP. MPI codes typically involve complicated communications with significant programming efforts and could not handle fault tolerance very well. Although OpenMP makes it easy to parallelize sequential codes on SMP, with increasing size of volume of seismic data, SMP encounters the problem of caching large data volume and scalable performance issue.

We choose Scala as the programming language for experiments and develop four applications for Jacobi stencil computation: Sequential codes, Parallel codes using broadcasting variable, Parallel codes using Cassandra database and Parallel codes with boundary RDD, in which sequential codes run on single core and parallel codes in Spark run on the whole cluster. For the Sequential codes, we just split the big data file into small partitions and each partition includes several inlines, then use 3 nested loops to compute average value of 26 (3x3x3 sub-volume) neighbor samples. After computation, results of each partition will be saved into the temporary file to be used as input of next iteration. For the Parallel codes with broadcasting variable, the large input dataset is distributed to all active nodes in the whole cluster as RDD, then each node could get its own data section and apply map function (computation part) on it. Since the boundary data is need for Jacobi kernel and there is no communication interfaces between mappers, the boundary planes of each split are collected as a big variable and broadcasted to all nodes by driver node after one iteration, then each node could fetch new boundary Inlines in next iterative computation. After implementation of Parallel codes using broadcasting variable, we found the performance of collecting data is very bad, so we design new Parallel codes using Cassandra DB and boundary RDDs to exchange boundary data and to avoid collecting data in each iteration.

The data flow of parallel methods (broadcast variables and boundary RDD) are shown in Figure 3: each number denotes one plane of seismic data. The big seismic data file is stored on HDFS, which will be divided into small partitions and send to each worker node by driver node. Using broadcast variables shown in top half of diagram, the driver node needs to collect boundary planes from every node, which will take more time on network communication. The solution of using boundary RDDs is shown in below half diagram, in which boundary RDDs are filtered from result RDD, but they need repartition and resort for alignment and then merge with original RDD to form a new RDD for next iteration. In the programmer view of sharing data, sharing with Cassandra database is similar to broadcast variables of Spark, but the underline data communication is distinct.

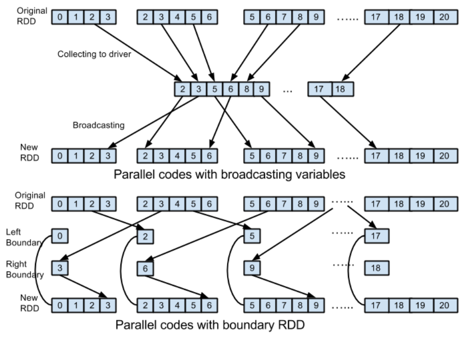


Fig. 3: Data Flow of Jacobi Parallel Codes

V. Performance Discussion

There are a lot of tools used for measuring performance of the application running on single node or on cluster. In order to make some deep analysis to find the performance bottleneck, we used several performance tools to collect the detailed performance metrics such as CPU usage, memory usage, disk I/O and network communication. Spark itself provides an web UI for monitoring of resource usage of the whole cluster, task running status and detail information of each stage in task, which emphasizes more on application profile and execution time. Ganglia [48] is a scalable distributed monitoring system for high-performance computing systems such as clusters and Grids, which is focus on utilization of resources on the cluster. Nigel's performance Monitor (nmon) could collect miscellaneous metrics information on each node, and NMONVisualizer could visualize the information for deep analysis. Since Spark framework runs on Java Virtual Machine (JVM), there are also several tools from Oracle Java Development Kit (JDK) that provide information within the JVM, such as Garbage Collection (GC) log, Java Flight Recorder, and heap dumps. We use wall clock to get total running time and execution time of each stage for sequential codes, and use Spark web monitor UI to get execution time of total job and running information of each stage for parallel codes running on Spark. Each node will launch nmon to collect runtime information at the same time of application starts, and then all of these data are merged together, feeding to NMONVisualizer for deep study.

A. *Performance Analysis of Hilbert Transformation Filter*

Figure 4 shows the speedup information of Hilbert transformation filter with parallel codes on Spark to sequential codes. There is a positive correlation between number of cores and performance in almost each split. The performance boost from 288 cores to 576 cores is not obvious, because 576 cores run in hyper-thread mode. The best speed up is located at 10 lines per split with 576 cores, and the performance decrease slightly after increasing split size to 30 lines or 100 lines. Figure 5 shows the resource utilization of Hilbert transform filter running with 10 lines per split and 576 cores, in which CPU usage quickly ramped up to 95% and kept steady till the end of the job. There are some small fluctuations due to reading data from network, and the big fluctuations related with system time came along with significant longer GC pauses. Figure 6 shows the same data running with 30 lines per split and 288 cores, in CPU usage of which there are 4 peak bands that mean 4 sub-tasks and there is one valley at the end where the entire job is finalizing and waiting for stragglers. Figure 7 shows resource usage on another node, on which 5 tasks have been run. In the case of 30 lines per split, each sub-task will take more time and not all nodes get same number of sub-tasks to run, so there are some nodes need to wait all tasks finished at end. The utilization of CPU is better in 10 lines per split, so with same resources, performance of 10 lines per split is better than the one of 30 lines per split.

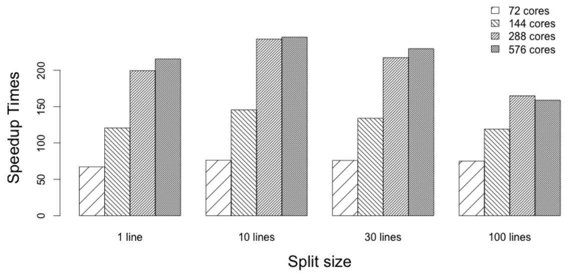
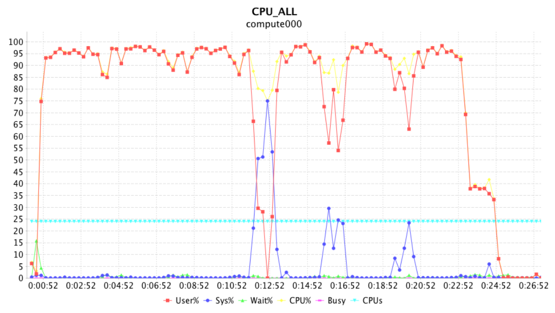
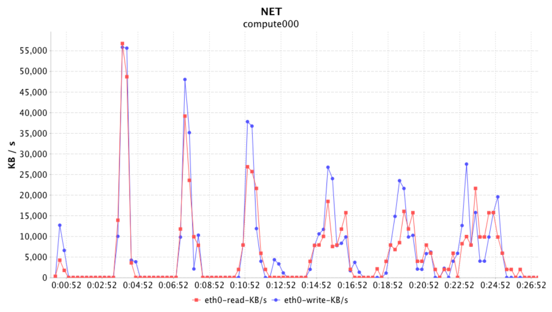


Fig. 4: Speedup of Hilbert Transform Filter Codes



(a) CPU Usage

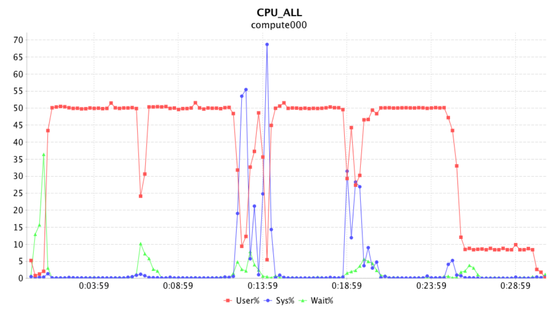


(b) Network Usage

Fig. 5: Resource Usage of Hilbert Transform Filter with 10 Lines per Split and 576 Cores

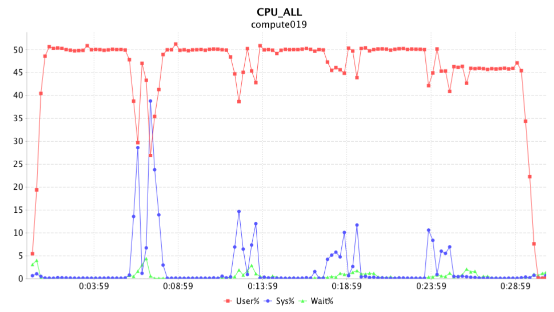
B. *Performance Analysis of FFT*

Figure 8 shows the speedup information of FFT parallel codes on Spark to sequential codes. There is a positive correla-



(a) CPU Usage

Fig. 6: Resource Usage of Hilbert Transform Filter with 30 Lines per Split and 288 Cores (1)



(a) CPU Usage

Fig. 7: Resource Usage of Hilbert Transform Filter with 30 Lines per Split and 288 Cores (2)

tion between number of cores and performance in almost each sub-volume size. The performance boost from 288 cores to 576 cores is not significant, which means we have bottleneck comes from other components of the system. Similar with Hilbert transformation filter, FFT codes in this case are also computation-intensive: every pixel with its 26 neighbor pixels in 3x3x3 pattern and every one with its 728 neighbors in 9x9x9 pattern are used for FFT. Figure 9 and 10 show resource usage of FFT codes with sub-volume size 3x3x3 that run with 576 cores. The overall percentage of CPU utilization is good except some spikes that mean GC of JVM. At the end of running period, there is a section of waiting time. From Figure 10, the disk is busy on DataNode of HDFS at that time; For the 100GB dataset with 36000 inlines, there are about 12000 sub-tasks need to write back results to HDFS, so it will take about 3 minutes to complete writing operations.

Figure 11 shows the resource utilization of FFT codes with sub-volume size 9x9x9 that running with 576 cores. Comparing with sub-volume size 3x3x3, the computational complexity of 9x9x9 increases quickly. The size of split for case 9x9x9 also increases, in which we define 15 lines with 4 overlaps on left side and right side, so each split could get FFT results of 7 lines after computation. The CPU utilization keeps about 95% from start to end, and the increase of split size reduces frequency of communication between each worker node and NameNode, so the overall speedup of this case is prominent.

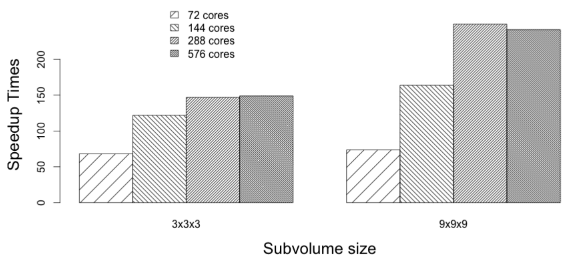
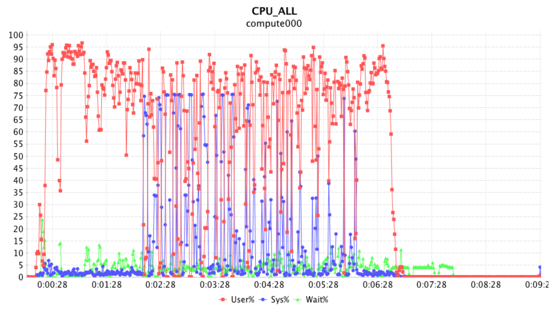
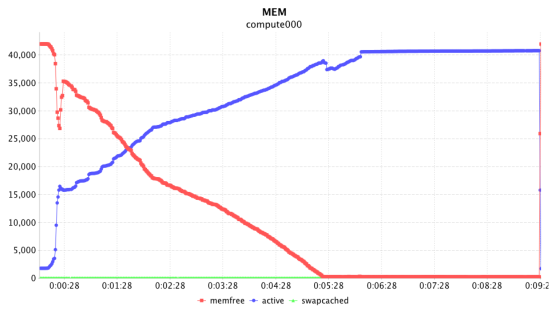


Fig. 8: Speedup of FFT Codes



(a) CPU Usage

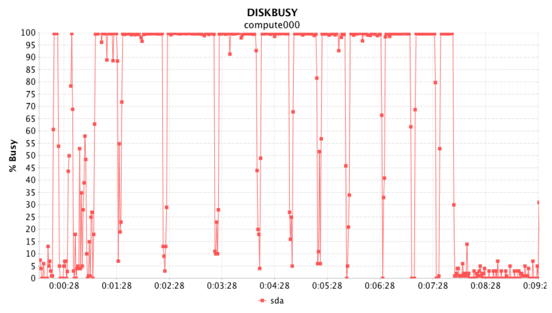


(b) Memory Usage

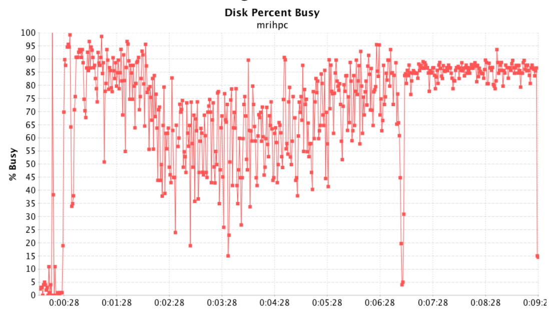
Fig. 9: Resource Usage of FFT with Sub-volume size 3x3x3 and 576 Cores (1)

C. *Performance Analysis of Jacobi Iteration*

Jacobi stencil computation application is more sophisticated than other ones in our experiments, since it needs to exchange boundary data after each iteration. The computation is not complicate, but in normal case, it will take several iterations to reach balance. In traditional Map/Reduce model, each worker task could not communicate with others, even they may be on the same node. To overcome this problem, some data sharing mechanism need to add for Jacobi stencil codes. Similar to scatter/gather functions in MPI, Spark itself provides broadcast variables and collect functions, but as shown in Figure 12, the performance is not good. So we introduce distributed database, Cassandra to save the intermediate boundary data, and all workers need to read boundary data from database before computation and save boundary data back to database

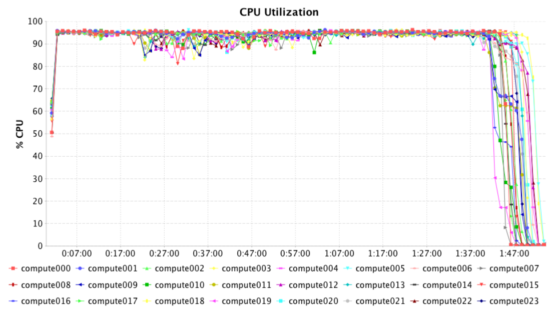


(a) Disk Usage of Worker Node

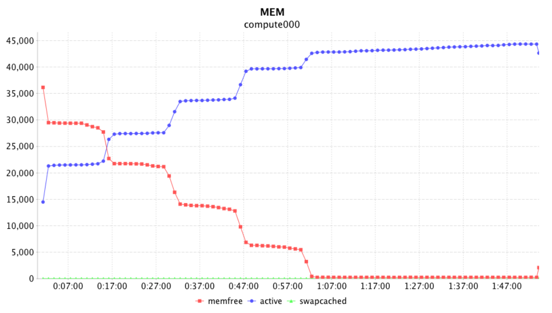


(b) Disk Usage of Master Node (NameNode)

Fig. 10: Resource Usage of FFT with Sub-volume size 3x3x3 and 576 Cores (2)



(a) CPU Usage



(b) Memory Usage

Fig. 11: Resource Usage of FFT with Sub-volume size 9x9x9 and 576 Cores

after finished in each iteration. To get best performance on Spark, the most important strategy is trying to avoid actions such as collection and persisting data, and maximize pipeline execution for all stages. So in the final implementation of Jacobi case, boundary RDD was introduced, which could get best speedup as shown in Figure 12.

Figure 13 shows the CPU and network utilization of Jacobi codes using broadcast variables, in which usage of CPU is very low, and more time is consumed on waiting and GC. In the 10 iterations, each one need to save boundary and read new one through network, so there are clear spikes in network read and write. With such bottlenecks in this case, the performance is certain to fall.

Figure 14 shows the CPU and network utilization of Jacobi codes using Cassandra database to save boundary data, in which the utilization of CPU is better than previous one, and there are not many GC pauses. However, to read boundary data from database and to write those back to database, there are still huge amount of network read/write that cause CPU to be under use.

Figure 15 shows the CPU and network utilization of Jacobi codes using boundary RDD to save edge data. The CPU utilization is better in each iteration, but since boundary RDDs need repartition and resort for alignment and then merge with result RDD to form a new RDD for next iteration using zip function in Spark, there are repeated network I/O for exchanging data between worker nodes. Comparing with sequential codes and previous two methods on Spark, the performance of boundary RDD is much better.

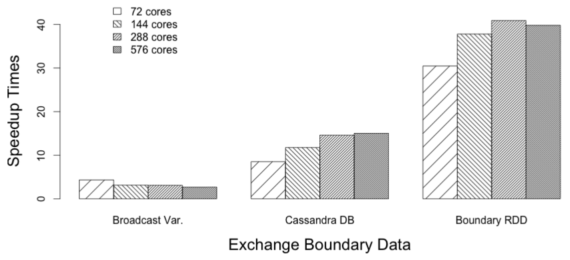
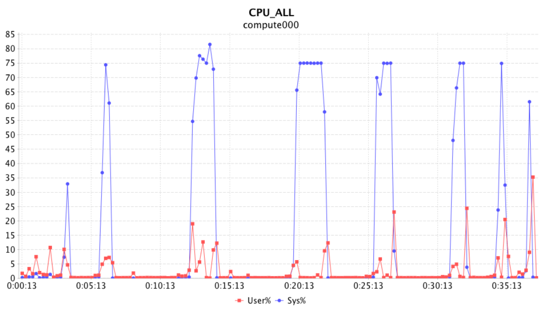


Fig. 12: Speedup of Jacobi Stencil Codes

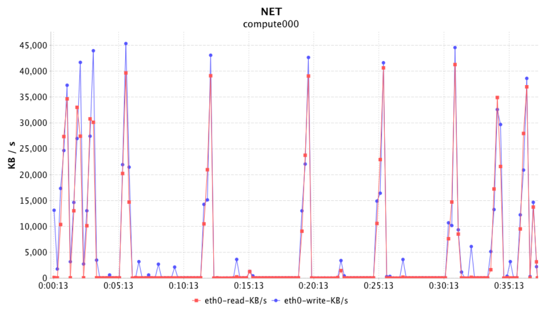
VI. Conclusion and Future Work

In this paper, we focus on experiments with a productive big data analytics cloud platform to overcome the challenges of processing big seismic data. SAC was developed and evaluated with several typical seismic applications, in which there is no prerequisite that users have parallel computing knowledge, and only the core algorithms need to be filled with the help of template provided by SAC. These templates provide a high-level user interface for geophysicists without sacrifice of performance. Some deep performance analysis about data partition, memory and network utilization are also given in this paper, which is a good experience for profiling Spark applications.

Although SAC has been evaluated to be a good candidate for processing seismic data, there are still some space to improve. Current templates can hand the typical seismic applications,

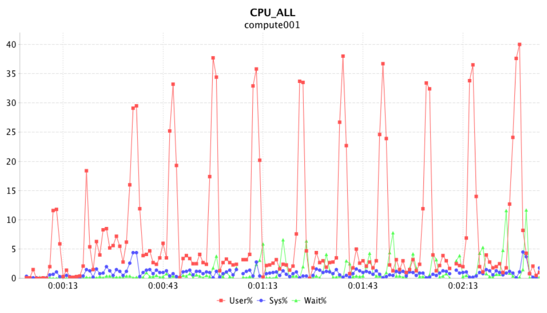


(a) CPU Usage

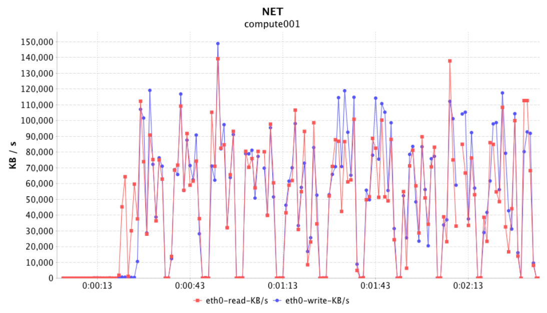


(b) Network Usage

Fig. 13: Resource Usage of Jacobi Codes Using Broadcast Variables for Sharing Data and Running with 576 Cores

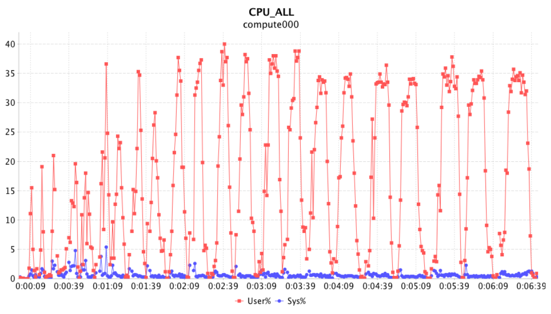


(a) CPU Usage

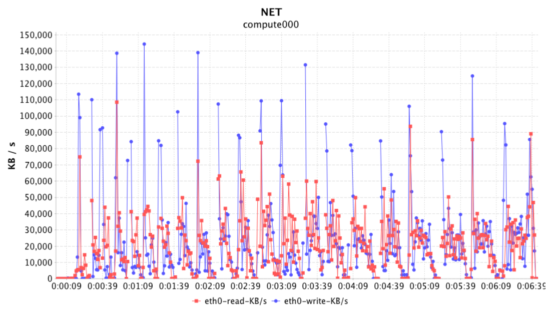


(b) Network Usage

Fig. 15: Resource Usage of Jacobi Codes Using Boundary RDD for Sharing Data and Running with 576 Cores



(a) CPU Usage



(b) Network Usage

Fig. 14: Resource Usage of Jacobi Codes Using Cassandra DB for Sharing Data and Running with 576 Cores

for some complicated cases, however, more templates need to be defined. It is still a challenge for defining a template if worker threads need to communicate with each other, which will be a focus of our next task. In the future, more high level machine learning algorithms will be added to SAC in order to create advanced seismic data analytics models. To make SAC easy to be used by high level users and improve communication efficiency, Workflow that could connect algorithms and Notebook for interactive seismic data processing are under development. Current visualization of seismic data in web interface is still in 2D mode, 3D view mode with remote rendering is already evaluated. For the performance optimization of parallel program in SAC, more deep research tasks are planned, such as adjusting GC parameters, GPU optimization and global memory access support, etc. In the view of applications developers, more data and computing models are needed to investigate, such as streaming data, and hybrid execution mode supporting legacy codes etc.

Acknowledgment

This research work is supported by the US NSF CRI program award #1205699 and by the Office of the Assistant Secretary of Defense for Research and Engineering (OASD(R&E)) under agreement number FA8750-15-2-0119. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the US NSF, or the Office of the Assistant Secretary of Defense for Research and Engineering (OASD(R&E)) or the U.S. Government.

References

[1] "Hadoop Introduction," http://hadoop.apache.org/, [Retrieved: July, 2015].

[2] "Spark Lightning-fast cluster computing," http://spark.incubator.apache.org/, [Retrieved: July, 2015].

[3] Y. Yan and L. Huang, "Large-scale Image Processing Research Cloud," in *CLOUD COMPUTING 2014, The Fifth International Conference on Cloud Computing, GRIDs, and Virtualization*, Venice, Italy, May, 25-29 2014.

[4] H. M. Y. L. Yan, Y. and L. Huang, "Building a productive domain-specific cloud for big data processing and analytics service," *Journal of Computer and Communications*, vol. 3, no. 5, pp. 107–117, 2015.

[5] M. P. I. Forum, "http://www.mpi-forum.org."

[6] "OpenMP: Simple, Portable, Scalable SMP Programming," http://www.openmp.org/, 2006.

[7] T. El-Ghazawi, W. Carlson, T. Sterling, and K. Yelick, *UPC: Distributed Shared Memory Programming*. John Wiley and Sons, May 2005.

[8] R. W. Numrich and J. Reid, "Co-array {Fortran} for parallel programming," *SIGPLAN Fortran Forum*, vol. 17, no. 2, pp. 1–31, 1998.

[9] P. Charles, C. Donawa, K. Ebcioglu, C. Grothoff, A. Kielstra, V. Saraswat, V. Sarkar, and C. V. Praun, "{X10}: An Object-Oriented Approach to Non-Uniform Cluster Computing," in *Proceedings of the 20th {ACM SIGPLAN} conference on Object-oriented programing, systems, languages, and applications*. ACM SIGPLAN, 2005, pp. 519–538. [Online]. Available: http://grothoff.org/christian/x10.pdf

[10] J. D. S. Ghemawat, "MapReduce: simplified data processing on large clusters," in *Communications of the ACM - 50th anniversary issue: 1958 - 2008*, vol. 51. ACM New York, Jan. 2008, pp. 107–113.

[11] "Seismic Data Science: Reflection Seismology and Hadoop," http://blog.cloudera.com/blog/2012/01/seismic-data-science-hadoop-use-case/, [Retrieved: July, 2015].

[12] "CWP/SU: Seismic Unix," http://www.cwp.mines.edu/cwpcodes/index.html, [Retrieved: May, 2015].

[13] A. Mohammadzaheri, H. Sadeghi, S. K. Hosseini, and M. Navazandeh, "Disray: A distributed ray tracing by map-reduce," *Comput. Geosci.*, vol. 52, pp. 453–458, Mar. 2013. [Online]. Available: http://dx.doi.org/10.1016/j.cageo.2012.10.009

[14] N. Rizvandi, A. Boloori, N. Kamyabpour, and A. Zomaya, "Mapreduce implementation of prestack kirchhoff time migration (pktm) on seismic data," in *Parallel and Distributed Computing, Applications and Technologies (PDCAT), 2011 12th International Conference on*, Oct 2011, pp. 86–91.

[15] T. Addair, D. Dodge, W. Walter, and S. Ruppert, "Large-scale seismic signal analysis with hadoop," *Comput. Geosci.*, vol. 66, no. C, pp. 145–154, May 2014. [Online]. Available: http://dx.doi.org/10.1016/j.cageo.2014.01.014

[16] H. Wang, Y. Shen, L. Wang, K. Zhufeng, W. Wang, and C. Cheng, "Large-scale multimedia data mining using mapreduce framework," in *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on*, Dec 2012, pp. 287–292.

[17] "Apache Flink: Fast and reliable large-scale data processing engine," http://flink.apache.org/index.html, [Retrieved: July, 2015].

[18] A. Alexandrov, R. Bergmann, S. Ewen, J.-C. Freytag, and F. Hueske, "The stratosphere platform for big data analytics," *The VLDB Journal*, vol. 23, pp. 939–964, 2014.

[19] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: cluster computing with working sets," in *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, ser. HotCloud'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 10–10. [Online]. Available: http://dl.acm.org/citation.cfm?id=1863103.1863113

[20] M. Z. Mosharaf Chowdhury and T. Das, "Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing," in *NSDI'12 Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*. San Jose, CA: USENIX Association Berkeley, Apr. 2012.

[21] R. Balasubramonian, J. Chang, T. Manning, J. H. Moreno, R. Murphy, R. Nair, and S. Swanson, "Near-data processing: Insights from a micro-46 workshop," *Micro, IEEE*, vol. 34, no. 4, pp. 36–42, July 2014.

[22] Z. Guo, G. Fox, and M. Zhou, "Investigation of data locality in mapreduce," in *Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on*, May 2012, pp. 419–426.

[23] M. Perrone, L.-K. Liu, L. Lu, K. Magerlein, C. Kim, I. Fedulova, and A. Semenikhin, "Reducing data movement costs: Scalable seismic imaging on blue gene," in *Parallel Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*, May 2012, pp. 320–329.

[24] n. C. N. Paradigm, NetApp, "Next-Generation Data Center Architecture for Advanced Compute and Visualization in Upstream Oil and Gas," http://www.pdgm.com/resource-library/articles-and-papers/2012/next-generation-data-center-architecture-for-advan/, [Retrieved: July, 2015].

[25] H. Vo, J. Bronson, B. Summa, J. Comba, J. Freire, B. Howe, V. Pascucci, and C. Silva, "Parallel visualization on large clusters using mapreduce," in *Large Data Analysis and Visualization (LDAV), 2011 IEEE Symposium on*, Oct 2011, pp. 81–88.

[26] "Apache Hadoop database, a distributed, scalable, big data store," http://hbase.apache.org/, [Retrieved: July, 2015].

[27] "Apache Cassandra database," http://cassandra.apache.org/, [Retrieved: July, 2015].

[28] "MongoDB: Agility, Scalability, Performance," http://www.mongodb.org/, [Retrieved: July, 2015].

[29] "Redis:advanced key-value cache and store," http://www.redis.io/, [Retrieved: July, 2015].

[30] S. Chaki, A. Routray, and W. Mohanty, "A novel preprocessing scheme to improve the prediction of sand fraction from seismic attributes using neural networks," *Selected Topics in Applied Earth Observations and Remote Sensing, IEEE Journal of*, vol. PP, no. 99, pp. 1–1, 2015.

[31] Y. Deng and H. Wang, "The support vector machines for predicting the reservoir thickness," in *Natural Computation (ICNC), 2012 Eighth International Conference on*, May 2012, pp. 118–120.

[32] M. Machado, M. Vellasco, P. M. Silva, and M. Gattass, "Using neural networks to evaluate the effectiveness of a new seismic fault attribute," in *Neural Networks, 2006. SBRN '06. Ninth Brazilian Symposium on*, Oct 2006, pp. 208–213.

[33] K. Kaur, M. Wadhwa, and E. Park, "Detection and identification of seismic p-waves using artificial neural networks," in *Neural Networks (IJCNN), The 2013 International Joint Conference on*, Aug 2013, pp. 1–6.

[34] W. Dai, J. Wei, X. Zheng, J. K. Kim, S. Lee, J. Yin, Q. Ho, and E. P. Xing, "Petuum: A framework for iterative-convergent distributed ml," *arXiv*, 2014.

[35] "GraphLab Create: A Python-based machine learning platform ," http://graphlab.org, [Retrieved: July, 2015].

[36] "Open Source Computer Vision," http://www.opencv.org/, [Retrieved: July, 2015].

[37] M. Frigo, Steven, and G. Johnson, "The design and implementation of FFTW3," in *Proceedings of the IEEE*, 2005, pp. 216–231.

[38] "Mesos: A distributed systems kernel," http://mesos.apache.org, [Retrieved: July, 2015].

[39] "ScalaNLP Scientific Computing, Machine Learning, and Natural Language Processing," http://www.scalanlp.org, [Retrieved: May, 2015].

[40] "Spark Programming Guide," https://spark.apache.org/docs/latest/programming-guide.html, [Retrieved: June, 2015].

[41] "Mines Java Toolkit (JTK)," http://inside.mines.edu/~dhale/jtk/index.html, [Retrieved: May, 2015].

[42] "Penobscot 3D Survey," https://opendtect.org/osr/pmwiki.php/Main/PENOBSCOT3DSABLEISLAND, [Retrieved: May, 2015].

[43] "Fast Fourier transform," https://en.wikipedia.org/wiki/Fast_Fourier_transform, [Retrieved: June, 2015].

[44] M. Frigo and S. G. Johnson, "The design and implementation of FFTW3," *Proceedings of the IEEE*, vol. 93, no. 2, pp. 216–231, 2005, special issue on "Program Generation, Optimization, and Platform Adaptation".

[45] "Hilbert transform," https://en.wikipedia.org/wiki/Hilbert_transform, [Retrieved: June, 2015].

[46] V. erven and J. Zahradnk, "Hilbert transform and its geophysical applications," *Acta Universitatis Carolinae. Mathematica et Physica*, vol. 16, pp. 67–81, 1975. [Online]. Available: http://hdl.handle.net/10338.dmlcz/142362

[47] "Hilbert Transform Remains a Valuable Tool," http://www.aapg.org/Publications/News/Explorer/Column/ArticleID/2908/Hilbert-Transform-Remains-a-Valuable-Tool, [Retrieved: June, 2015].

[48] "Ganglia Monitoring System," http://ganglia.sourceforge.net, [Retrieved: July, 2015].