A SCALABLE AND DISTRIBUTED SEISMIC DATA TOOLKIT ON BIG DATA ANALYTICS PLATFORM

A Thesis

by

CHAO CHEN

Submitted to the Office of Graduate Studies of
Prairie View A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

December 2016

Major Subject: Computer Science

A SCALABLE AND DISTRIBUTED SEISMIC DATA TOOLKIT ON BIG DATA ANALYTICS PLATFORM

A Thesis

by

CHAO CHEN

Submitted to the Office of Graduate Studies of
Prairie View A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Approved as to style and content by:

Lei Huang Chair of Committee	Li Lin Committee Member
Sherri S. Frizell Committee Member	Yonggao Yang Head of Department
Kendall T. Harris Dean, Roy G. Perry College of Engineering	Cajetan M. Akujuobi Dean, Graduate Studies

December 2016

Major Subject: Computer Science

ABSTRACT

A Scalable and Distributed Seismic Data Toolkit on Big Data Analytics Platform

December 2016

Chao Chen, B.S., southwest university of science and technology;

Chair of Advisory Committee: Dr. Lei Huang

Seismic volume is a basic data structure that has been widely used in geophysical computing, as well as in big data analytics applications of petroleum industry. It has been well studied in High Performance Computing (HPC) platforms. However, a little study has been done in big data analytics platforms. In this thesis, we present the work of designing and implementing an efficient Distributed Seismic Data Analytics Toolkit on top of Apache Spark big data analytics platform. The toolkit supports different ways of seismic volume data distributions, repartition, transposing, access, and data parallelism with a variety of parallel execution templates. This thesis also addresses the performance characteristics of these seismic volume data operations with different configurations, the services have been developed on top of this toolkit as well as the current status and future research plan of this big data analytics plaftorm. The scalability and parallelism are the main characteristics of this platform.

ACKNOWLEDGEMENTS

I would like to thank my advisor, professors, colleagues and friends who helped me a lot in the past two years. I cannot make it through and finish my thesis without your kind help.

First I want to thank my advisor Dr. Lei Huang, for giving me the chance to work in CloudComputing Lab and guiding my study and research in cloud computing, and thanks to Dr. Lin Li and Dr. Frizell for spending your time to refine my thesis which benefits me a lot in academic writing skills. Special thanks to Mr Ted Clee, for sharing his deep knowledge of seismic data analytics.

I also appreciate all my colleagues and classmates worked in the lab in the past two years, for sharing so many interesting things from different cultures, and, of course, for helping and encouraging each other in research work.

Finally I want to say thank you to my friend Yuzhong Yan, for always being the guy helps me out at so many tough time in my life. I have learned so much from you not only as an engineer but also as a kind and wise man.

Best wishes to all of you and your families.

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

TABLE Page

CHAPTER 1

INTRODUCTION

This thesis presents the design and development work of a scalable and distributed seismic data analytics toolkit, as well as the evaluation of the promised scalability and performance of it based on related experiments result.

Petroleum, the target market of this seismic data analytics toolkit, 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 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. However, the performance of traditional methodologies of oil & gas companies is limited by the resources on a single machine, as the complicated work is hard to implement in a scalable and distributable way on the traditional software architecture. To resolve this problem, we designed and implemented a scalable and distributed seismic data analytics software development toolkit on top of popular big data platform Apache Hadoop and Spark.

1.1 Motivations

1.1.1 Big Data and Scalability

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. Various types of data are adopted to build models and images of the Earth's structure and layers 5,000-35,000 feet below the surface, as well as

to address activities around the wells, such as oil flow rates and pressures. With approximately one million wells currently producing oil and/or gas in the United States alone, and many more gauges monitoring performance, this dataset is growing daily [18]. Moreover, not only the real world datasets, but also the dimensions and complexity of datasets themselves increase dramatically, such as 4D even 5D and high density seismic data. The traditional HPC solutions are able to improve the performance of many computation-intensive models, however, most processing models are also data-intensive which are still the bottleneck for the whole workflow.

In many 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 exploring the valuable information from fast-growing data volumes and varieties. Hadoop and Spark 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.

1.1.2 Complicated Workflow

Since the seismic data processing flow involves deep knowledges of geophysics, data science and computer science, it requests intensive collaborations among the scientists and developers from many different fields. This situation has long been another bottleneck for the whole system. For an instance, most scientists are using MATLAB code to build their models, which is usually hard to translate to MPI codes.

In most cases, the program needs to be reconstructed and parallelized by software engineers. For this situation, it is already a big challenge to both geoscientist and software engineers to understand each other's work, not to mention to maintenance or optimize the huge amount of legacy code in this industry.

1.2 Objectives

Geophysicists need an ease-to-use and scalable platform that allows them to advance the seismic data exploration process, design more intelligent algorithms to increase the drilling success rate. By Incorporating the latest big data analytics technology with the geoscience domain knowledge will speed up their innovations in the exploration/interpretation 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.

The main objective of the work is to develop a seismic data analytics software development kits (SDK) to enable geophysicists to easily leverage the latest big data analytics technology to improve the seismic data exploration.

CHAPTER 2

BACKGROUND

2.1 Reflection Seismology

Nowadays, reflection seismology (or seismic reflection) is widely used in the petroleum industry to estimate the properties of the Earth's subsurface from reflected seismic waves and explore geophysics using the principles of seismology. The complete processing flow of this method involves data acquisition, data processing, data interpretation and attributes analysis [7].

As shown in Figure 2.1, data acquisition is performed by using seismic sources such as dynamite or air gun generate to spread out seismic waves, which are reflected back from encountered materials underground and then recorded by the receiving sensors. To get data-scientists-usable 2D/3D seismic dataset, the collected seismic wavelet needs to be pre-processed by data processing methods including deconvolution, common-midpoint (CMP) stacking and migration. After data processing, the seismic events are geometrically re-located in either space or time to the location the event occurred in subsurface and create a complete image of subsurface [7].

The goal of seismic data interpretation and attributes analysis is to locate the potential petroleum reservoirs from processed seismic reflections map. It involves deep knowledge of seismic attributes, geophysics, and intensive collaborations between data scientists and geophysicists. This thesis focus on develop a scalable and distributed toolkit to facilitates seismic data attributes analytics.

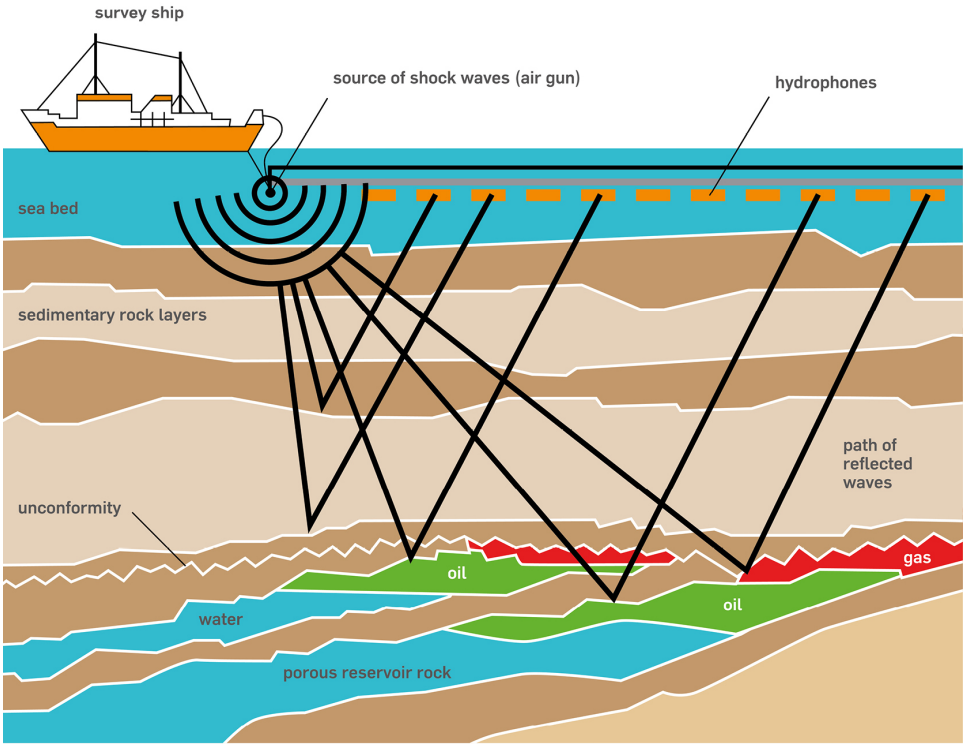

Figure 2.1. Reflection Seismology Survey [13]

2.2 Big Data Challenge

The most famous definition of big data, comes from Gartner analyst Doug Laney, specifies the 3Vs characteristics: volume, velocity and variety [20]. By which volume means the amount of data, velocity stands for the real-time speed of data in and out, and variety is the range of data types and sources. As mentioned in previous section, the burst increase of volume size, high-speed real-time streaming data from sensors, and various types of structured, unstructured and semi-structured data coming from different stages of seismic data processing all together matches the 3Vs definition. It determines seismic data is costly to store, access and manage in traditional methodology. Therefore, new technology should be adopted to address these

problems appropriately.

As the popularity of big data topic grows, there are more and more choices in big data market and open source communities. For instances, Apache Hive and HBase provide the scalable and distributed database solutions, Apache Storm is capable of handling real-time computation and Apache Kafka is a good choice if users are looking for a distributed streaming platform. These projects provide variety of big data solutions.

However, all of these big data platforms are designed for general purpose applications and focus on distributing the data, computation and IO overloads. Most MapReduce based framework do not have, or only have limited communication mechanisms between different maps, which is important to resolve the data and logical dependence problems in many complex applications. When it comes to the field of seismic data processing and analysis, the problem is more complicated. Since most scientists and researchers in petroleum industry do not have big data related knowledge or even computer science background, how to hide parallelism from them and let them easily deploy their works on new platform is a big challenge for all the researchers.

2.3 Hadoop File System

Since Google released its white paper series of big data processing technologies in 2004, the landscape of big data development has been changed profoundly. Many big data projects were inspired and developed based on MapReduce and Google File System framework. Hadoop and Spark is two most widely used open source big data solutions for many business and industry applications in recent years.

A year after the publication of MapReduce and Google File System framework, Doug Cutting and Mike Cafarella created an open source project Apache Hadoop,

which has been utilized in lots of industries to facilitate the works with big volume, variety and velocity of structured and unstructured input datasets [1]. Apache Hadoop consists of Hadoop Distributed File System (HDFS) and MapReduce [4].

Since distributed file system is fundamental to many main stream big data platforms, as it is able to store data across number of storage devices of a cluster, Hadoop Distributed Filesystem becomes one of the most popular Apache subprojects. Originally, HDFS was designed and built for the Apache search engine project Nutch as the file storage infrastructure [4]. Compare to traditional file system which holds sequential data on one device, HDFS provides far better scalability and support for parallel IO processing mode.

There are some most significant differences that distinguish HDFS from other distributed file systems. First, the high fault-tolerance makes HDFS is able to deployed on low-cost hardware. The high IO throughput access to distributed dataset of HDFS facilitates the applications that have large data sets. Also, HDFS provides streaming access to file system data which is unavailable in conventional POSIX file systems.

The architecture of a single NameNode and multiple DataNodes in a cluster simplifies the structure of the system. HDFS cluster runs in master-slave mode. It consists of a single NameNode, the master server which manages the filesystem namespace and the access to distributed datasets, and there are multiple DataNodes, which manage the data storage on the working nodes. HDFS manages the filesystem namespace which enable the general file-form data storage. Files are split into multiple blocks which are saved in multiple DataNodes. The NameNode manages the mapping of data distributions on the DataNodes, and provides general file operations such as open, close, and rename etc. It sends instructions to DataNodes, which perform the related operations on the data blocks and serve the requests of

data accessing. Figure 2.2 shows the NameNode-DataNodes architecture of Apache Hadoop File System.

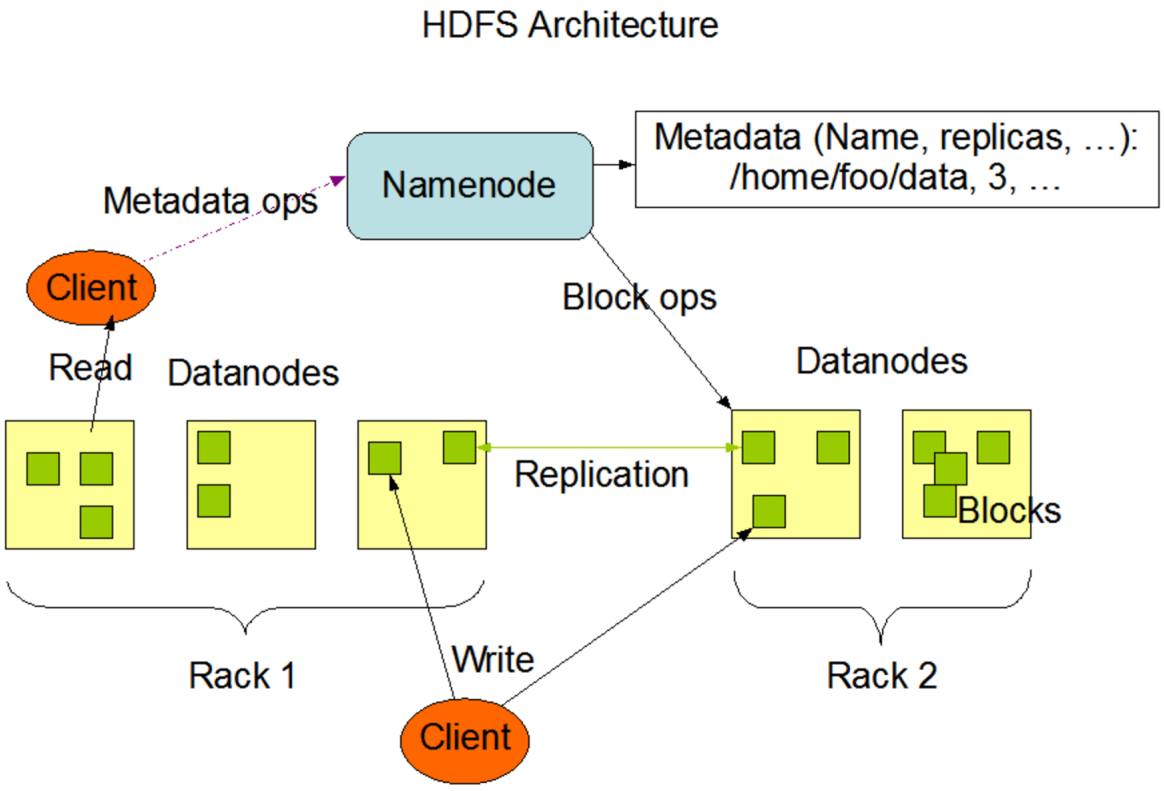

Figure 2.2. The Architecture of Hadoop File System [4]

2.4 Spark

The open source big data project Apache Spark provides programmers with an application programming interface centered on a data structure called the resilient distributed dataset (RDD), a fault-tolerant collection of elements that can be operated on in parallel [9]. Since Spark itself does not provide distributed file system, it is usually installed on top of Hadoop, by which Spark could utilize HDFS interface

to handle distributed data storage and access.

The most different part between Hadoop and Spark is the parallel processing interface. MapReduce writes the result back to the storage after each reduction, while Spark utilizes RDD to handle most its operations and result in memory. This leads to up to100 times performance improvement compare to Hadoop in certain circumstances [9]. Another advantage of Spark is it provides more advanced features such as real-time streaming processing interface and machine-learning library.

Resilient Distributed Dataset (RDD) is distributed data collection with high fault-tolerance and can be accessed and operated in parallel. A RDD can be generated in two way, transforming an existed RDD to a new RDD, or importing the data stored in file storage system. Although Spark supports data loading from both traditional and distributed file system, it is designed for performing on the latter one for the high performance and parallelism. Spark supports any data source providing the Hadoop input format, such as HDFS and HBase [9].

There are two types of RDD operations with distinct features: RDD transformation and action. The transformation operation builds a new RDD from an existing one, such as *map()* which processes the data elements by a user-defined function in parallel and output a new RDD as the result. Another type of operation is action, such as *count()* , which count all the elements in RDD and return the number of these data elements.

Spark handles all transformations in lazy-execution fashion, which posts the execution to later stage instead of processing it immediately. The system will record the applied transformations of RDDs and wait until there are actions applied on these datasets, then the posted transformations will be computed and the results of the actions will be output to driver program. This design make Spark can provide high efficient work flow for many complicated applications.

Spark will recompute and regenerate the RDD every time there is an action applied on this dataset by default. However, this system behavior can be replaced by making the RDD to persist in memory through *persist* (*cache*) method. It allows users to cache the data elements in memory for faster accessing. This feature distinguishes Spark from many other solutions such as MapReduce and is also the reason that Spark is one of the most efficient big data platform. Figure 2.3 shows the execution model of a Spark application.

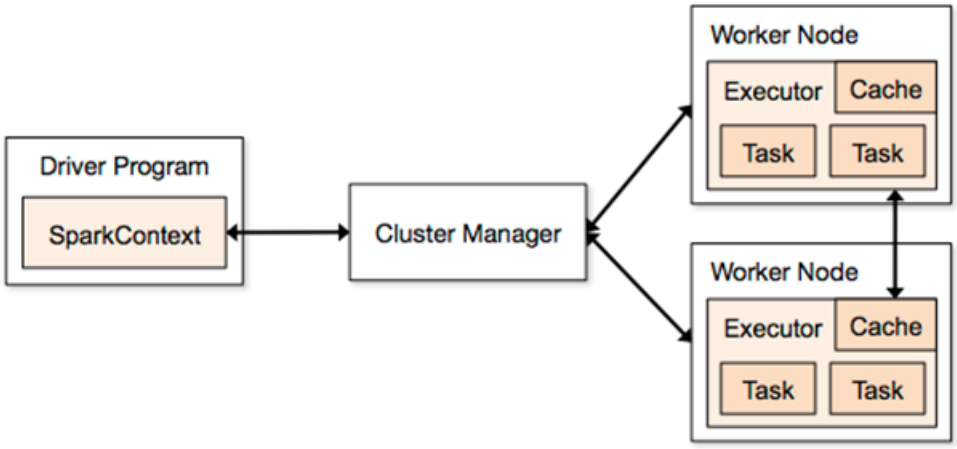

Figure 2.3. The Execution Model of Spark [9]

CHAPTER 3

RELATED WORK

Although various motivations exist in petroleum companies to adopt big data solutions to improve efficiency and reduce cost, only a few of them have deployed big data solutions. This situation may due to some technique barriers such as lack of technology knowledge, big data solution are not applicable in some steps of traditional workflow, and the cost and risk to convert legacy software to new platform solution etc. Moreover, there are lots of concerns of business-wise, such as the cost of infrastructure, data security issue (business or political restrictions on data accessing).

In [15], it concludes the applications of big data analytics in the petroleum industry are still in experimental stage. Only a few companies have applied the Big Data techniques on their workflows, and most of these innovations are developed by oil & gas service contracting companies such as Schlumberger and Halliburton, as well as some IT providers like IBM and Microsoft:

1. Chevron proof-of-concept adopted Apache Hadoop (IBM BigInsights) for seismic data processing;
2. Shell piloting Apache Hadoop in Amazon Virtual Private Cloud (Amazon VPC) for seismic sensor data;
3. Cloudera Seismic Hadoop integrated Seismic Unix with Apache Hadoop;
4. PointCross Seismic Data Server and Drilling Data Server utilizing Apache

Hadoop / NoSQL;

5. University of Stavanger data acquisition performance research used Apache Hadoop.

CHAPTER 4

DESIGN

The main goal of Seismic Data Analytics SDK is to implement a distributed software development toolkit to enable scalable storage, computation and analytics for big seismic volume datasets. This chapter will introduce the software architecture design and main functionalities of this toolkit.

4.1 Software Architecture Design

Seismic Data Analytics SDK is built upon Apache Hadoop and Spark. Figure 4.1 shows the software stack of a workable seismic data analytics platform. In this diagram, the gray part is the OS layer, the elements with green color stands for the infrastructure layer of this big data platform, and on top of that, SDK layer consists of the components with blue color. At the bottom of the infrastructure layer, there is Hadoop Distributed File System (HDFS) that stores the big seismic data files by utilizing the large number of local disks. The Cassandra as a NoSQL database is also used to store seismic data, intermediate results and meta data. YARN and Mesos are used for resources management. Apache Spark is the data distribution and parallel execution engine based on the innovative idea of Resilient Distributed DataSets (RDD) concept. MLLib is included in the Spark as the machine learning package to enable machine learning based data analytics algorithms. OpenCV is the widely used image processing package that is used to provide image processing capability. Breeze is the numerical processing package including linear algebra, signal processing, statistics, and other numerical computation and optimizations written in Scala. We have developed the seismic data RDD on top of Spark as the base

distributed seismic datasets to enable parallel operations and machine learning algorithms. Geophysicists and data scientists can use Seismic Data Analytics SDK to develop their own algorithms and leverage the capability of Apache Spark, as well as image processing, numerical computation, and deep learning packages.

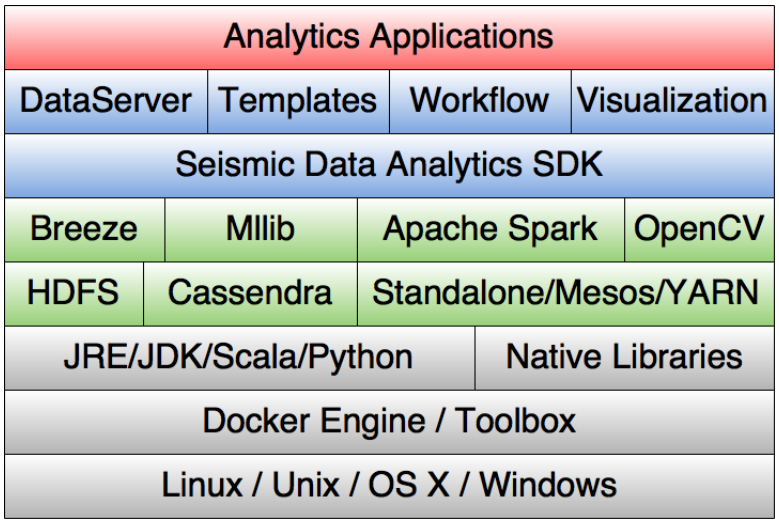

Figure 4.1. Software Stack of Seismic Data Analytics Platform

Figure 4.2 simplifies the development efforts for scalable and distributed computing and analytics of seismic datasets. It is built on top of the Apache Hadoop and Spark. The Hadoop provides a distributed file system(HDFS) and resource management system (YARN and Mesos), while Spark provides a high-level distributed data representation via Resilient Data Sets (RDD) and a data-parallelism execution engine. Seismic Data Analytics SDK provides configurable data distribution fashions for seismic volume data, as well as a configurable parallel execution interface to simplify the parallel programming efforts. Based on the functionality of SDK, we developed two useful utilities, parallel templates and data server, to facilitate SDK

for users to easily deploy their applications. Moreover, since Hadoop and Spark provide faults tolerance and task scheduling utilities, the toolkit inherits from them to provide fault tolerance and dynamic task scheduling for better reliability and task management.

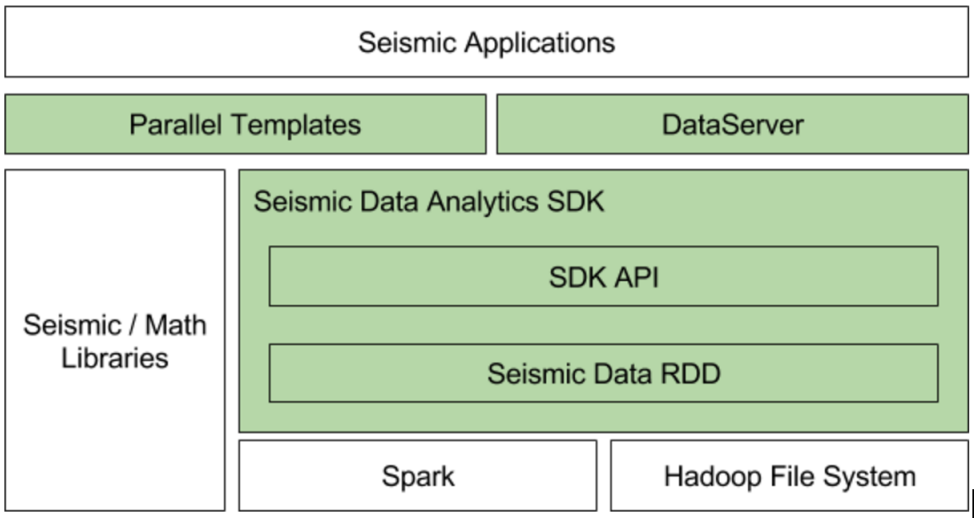

Figure 4.2. Framework of Seismic Data Analytics SDK

4.2 Interfaces and Functionalities

Figure 4.3 shows the main functionalities of Seismic Data Analytics SDK, including data loading/saving, configurable distribution, data accessing, 3D transpose and user-defined function mapping. It provides a single public class *SeismicVolume*, which integrates all the APIs of SDK. Developers are able to create *SeismicVolume* instances for specified seismic dataset, access data by configurable grain, perform 3D transposing and apply user function to the distributed data instance, and finally save the result to distributed file system through save API. The invalid input data format include 3D binary data and SEG-Y file [16] which is one of the most widely

used industrial standard format for seismic data.

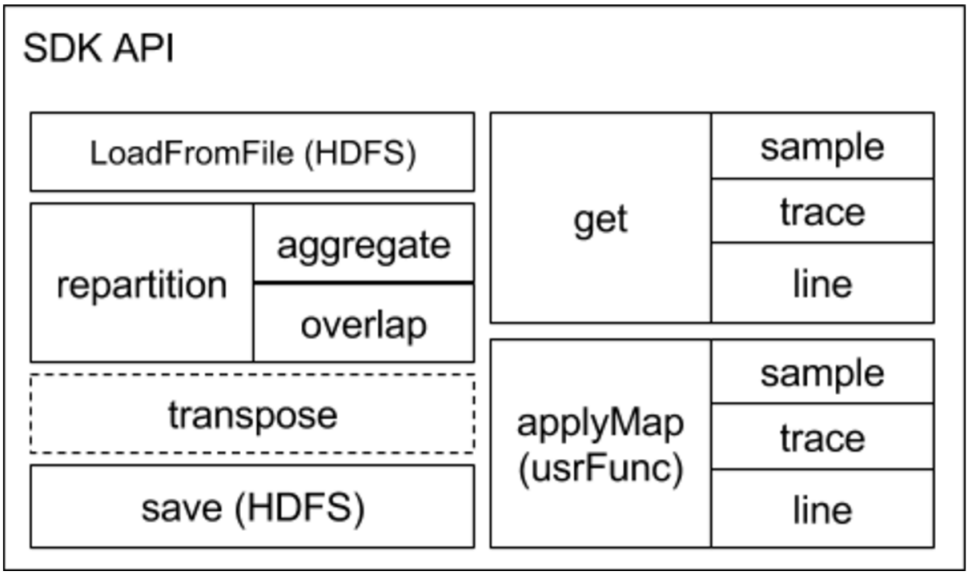

Figure 4.3. Main APIs of Seismic Data Analytics SDK

4.3 Programming Language

The host programming language of Seismic Analytics SDK is Scala, and the applications can be developed in Java. Scala (The acronym for Scalable Language), an object-oriented and functional language, provides great scalability in developing safe and high efficient multi-threaded programs [14]. The most important reason of using Scala as host language is it's also the native host language of Apache Spark. It means Scala is the most efficient programming language of this project. Moreover, Scala runs on the JVM which determines it can be freely integrated with Java and Java libraries and tools are also available. Since Scala compiler contains a subset of a Java compiler, Seismic Analytics SDK allows users who are not familiar with Scala can develop their applications in Java. This feature makes it possible for developers

to port the legacy Java applications to Seismic Analytics SDK without putting extra efforts on learning a new programming language.

CHAPTER 5

IMPLEMENTATION

5.1 Seismic Volume Data Loading, Distribution and Saving

As mentioned in chapter 2, seismic 3D volume data is a collection of estimated property values of the Earth's subsurface, obtained through seismic reflection survey and organized in 3D spacing form. It is widely used in energy companies for geophysics analysis, which could conduct more accurate subsurface exploration and exploit. As shown in Figure 5.1, the seismic volume data is defined through 3 different directions in 3D space: inline, crossline and timeline. The industry usually stored the data slice by slice along crossline direction and each slice is a inline section, which is also the default data organization format of Seismic Data Analytics SDK. In this case, each inline slice is a single split of the whole dataset.

The public class *SeismicVolume* provides an API *loadFromFile()* to load seismic data from HDFS and distribute them over Spark RDD according to users configurations. This API generates a *SeismicVolume* instance which contains a SeismicRDD with float/byte as internal binary data types. The SeismicRDD is a derived class from Spark RDD class with a variety of distributed fashions of seismic volume data. In addition, it also provides some optional parameters for advanced users who have already familiar with distributed system to specify the advanced data distribution fashions.

Figure 5.2 shows the flow of distributing an seismic volume file through the Hadoop filesystem and Spark RDD. It assumes the file has already been uploaded to Hadoop file system, which is able to support the distributed IO accessing for Spark to

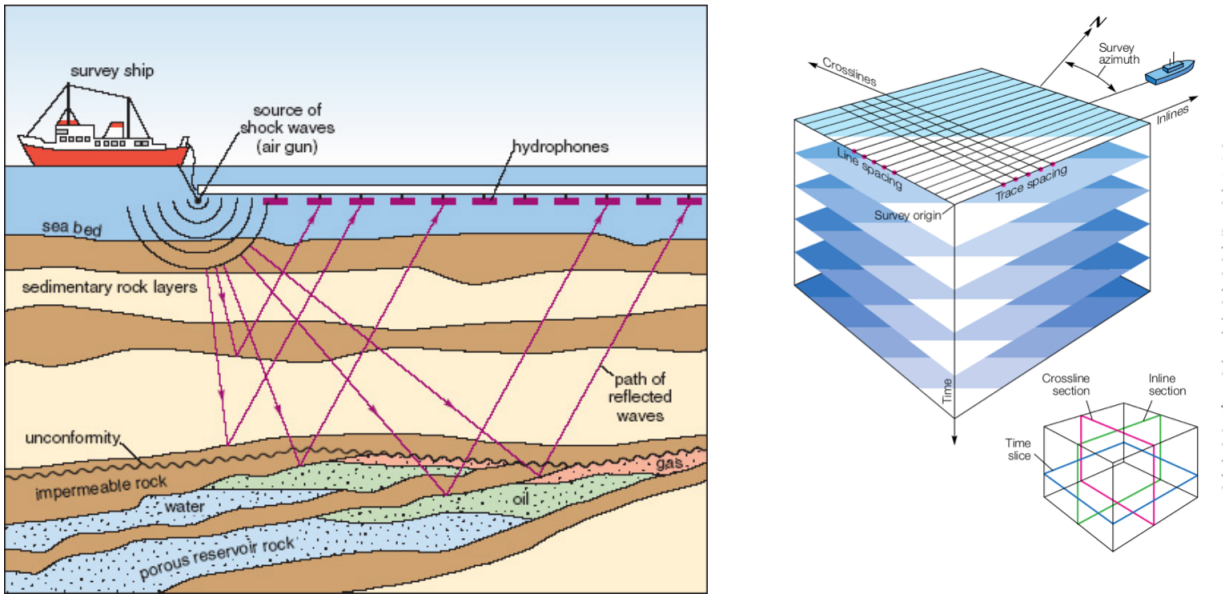

Figure 5.1. Reflection seismology Survey and Seismic Volume Data [13] [8]

load data in parallel. After users get the *SeismicVolume* instance of a specified file, all Spark RDD operations can be applied to the inside SeismicRDD object. Utilizing the RDD methods provided by Spark, developer could perform various of data operations and calculations on the dataset in parallel. By default, SDK distributes the volume in inline format slice by slice, which means each partition contains one single inline slice. The distribution direction could also be configured to crossline/timeline by given parameters in other APIs, this function will be present in the following Volume Data 3D Transposing section. Users could configure the slices count of each distribution to manage the data distribution grain, thus to efficiently tuning the performance of applications.

SeismicVolume class also provides *save()* API to allow users to store the data of SeismicRDD back to the HDFS. Although this operation is not recommended since it could introduce performance issues caused by data shuffling, it is still necessary when users need to backup the runtime data or apply the runtime data to traditional

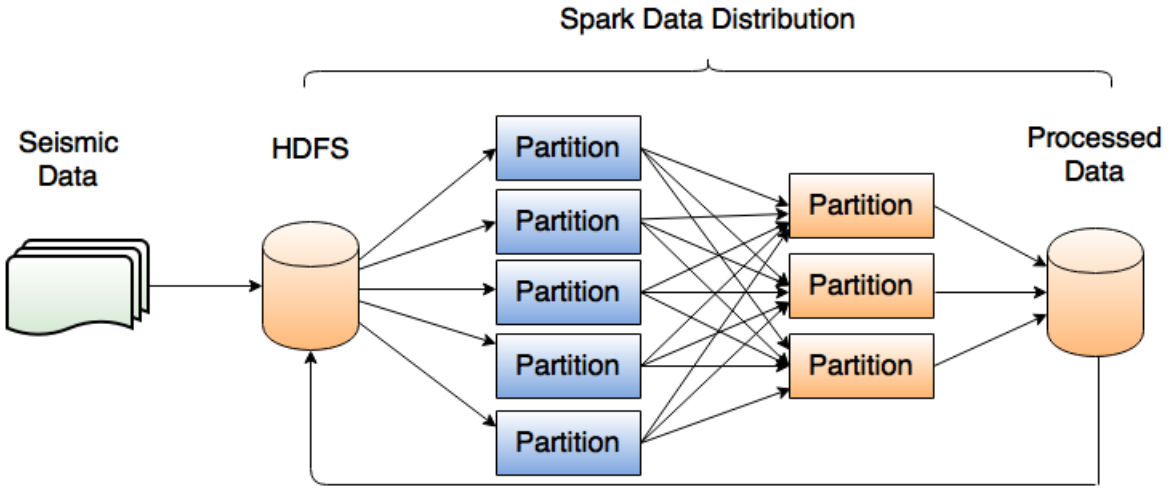

Figure 5.2. Seismic Volume Data Distribution Flow

sequential workflows.

5.2 Volume Data Accessing

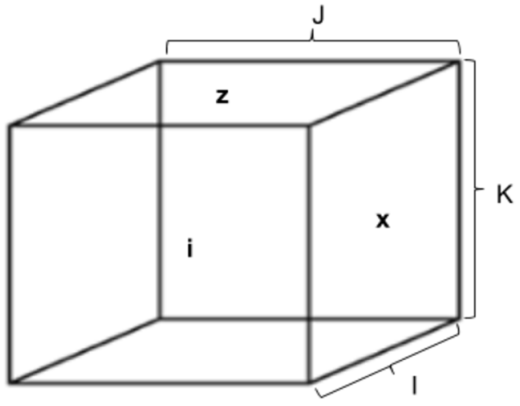

Figure 5.3. Seismic Volume Data Dimensions

Seismic Data Analytics SDK allows users to to access any slice/trace/sample data

in any direction of the volume through *SeismicVolume* APIs *getLine(direction:Int, idx:Int):Array[T]*, *getTrace(dir:Int, i:Int, j:Int):Array[T]* and *getSample(i:Int, j:Int, k:Int):T*.

For the convenience of addressing the data organization, we defined the three dimension of seismic volume as I (Dimension of inline slices), J (Dimension of crossline slices) and K (Dimension of timeline slices), as shown in Figure 5.3 and Figure 5.6. Users can specify any one of I, J and K directions to access data for visualization or computation purpose. Figure 5.4 shows the example of accessing seismic data by *getLine(direction:Int, idx:Int):Array[T]* API. The first parameter of *getLine*, which has three possible direction values(1 stands for I, 2 stands for J and 3 means K), specifies the direction users want to extract the data slice out of. The second parameter is the index number of the target slice in specified direction.

```
val sc = new SparkContext(new SparkConf());
val sv = SeismicVolume.loadFromFile[Float](sc, seismicXml);

sv.getLine(1, 0);
sv.getLine(2, 0);
sv.getLine(4, 0);
```

Figure 5.4. Example Code of Accessing Data Slice in SeismicVolume

Since Spark does not provide arbitrary data access, the APIs used IndexedRDD, a more efficient key-value management than native Spark API [5], to implement and speedup queries of slice/trace/sample data from any distributions to the master node.

5.3 Volume Data 3D Transposing

Since the volume data could only be stored in file following one specific direction(I, J or K), developer could not access slices of the other two directions directly. In traditional solutions, if users need data in other directions, organized as cross-line slice or time-line slice, the data fetching program must perform lots of seek operations between different file offsets to collect the data of a single cross-line or time-line slice. This tedious procedure will dramatically slow down the whole software performance. To resolve this problem and to achieve reasonable performance, SDK handles the transposing of the 3D volume data inside the *SeismicVolume* APIs and caches all three directions format SeismicRDD in *SeismicVolume*.

To explain the implementation clearly, we denote the seismic volume data as shown in Figure 5.3, in which i means I slice, x means J slice and z stands for K slice. The data is stored in iSlices format by default. To resolve the transposing problem in each distribution evenly, we split the volume to I of iSlices and distribute them over SeismicRDD. Each iSlice is a 2D matrix. As shown in Figure 5.6, each iSlice matrix consists of J of iTraces which have the length of K. An iSlice matrix could be iterated iTrace by iTrace. Since in 3D spacing, each iTrace is also the trace of xSlice, for example, the iTraces(0) is the xTrace of the 0th xSlice, the iTraces(1) is the xTrace of 1st xSlice, etc.

As mentioned in previous section, the default distribution of *SeismicVolume* splits the data slice by slice. To transpose the volume data organization to different direction, first we need to split the slice data partitions to number of smaller trace partitions and index them by trace index number. This process can be achieved by using *flatmap()* operation of Spark RDD. Figure 5.5 demonstrates how it splits the data partitions to more fine grain partitions. We apply a map function to *flatmap()*

API to index all iTraces of the volume. The new index is combined by index of iTrace and index of iSlice. After indexing the trace map, we got a volume RDD with new (iTraceIndex)(iSliceIndex) index as the key and trace data as the value.

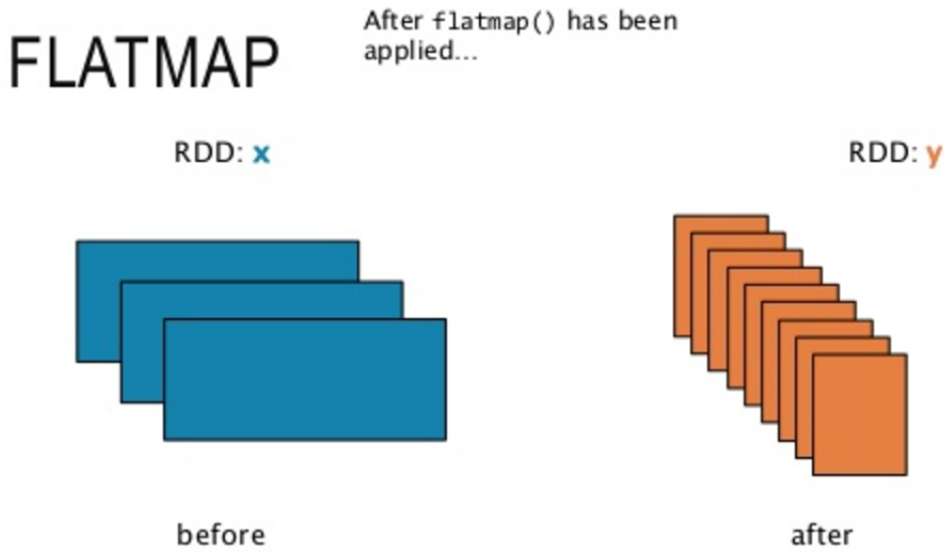

Figure 5.5. Flatmap Operation of Spark RDD

As shown in Figure 5.6, to get a xSlice, the next step is to group all the traces with the same iTraceIndex by utilizing the *groupByKey* operations of Spark RDD. As shown in Figure 5.7, this API groups all the data splits sharing the same key feature to a new data partition. After grouping, the xSlices data collection is generated in the new SeismicRDD distribution map. To organize them as a xSlices volume, all we need to do is sorting them by iTraceIndex. So far, the data in requested direction has already been stored in the *SeismicVolume* instance, therefore users could access and manipulate seismic data in any direction by specifying the direction parameter of related *SeismicVolume* APIs.

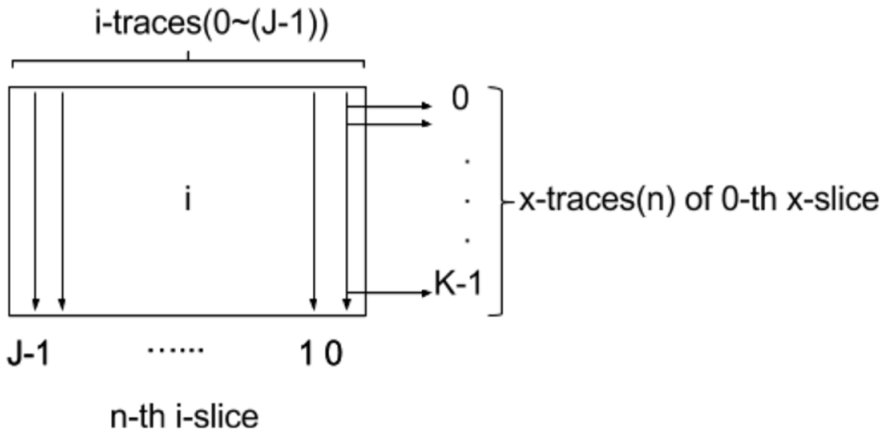

Figure 5.6. The Indexing for Resolving 3D Transposing Problem

5.4 Repartition

Users need a way to configure the data partition size and partitions number, since they need to efficiently tune the granularity of the distributed process to achieve better performance. By default, as shown in Figure 5.8, SDK distributes the volume in one specific direction slice by slice. In this case, each slice is a single split of the whole dataset. However, it will cause performance problems for some applications if we only support one distribution. For an instance, the transposing solution as mentioned in previous section needs to do lots of data exchanges between different data splits for the *group* operation to shuffle the dataset to expected arrangement. Since data shuffle in Spark RDD relies on lots of physical storage access and network transmissions in each worker node, it will become the bottleneck of the transposing performance if the number of partitions is too big. Obviously, to speedup the transposing operation, users need to reduce the number of partitions thus to reduce the data communications between worker nodes.

Developers can change the distribution layout to the aggregated and overlapped

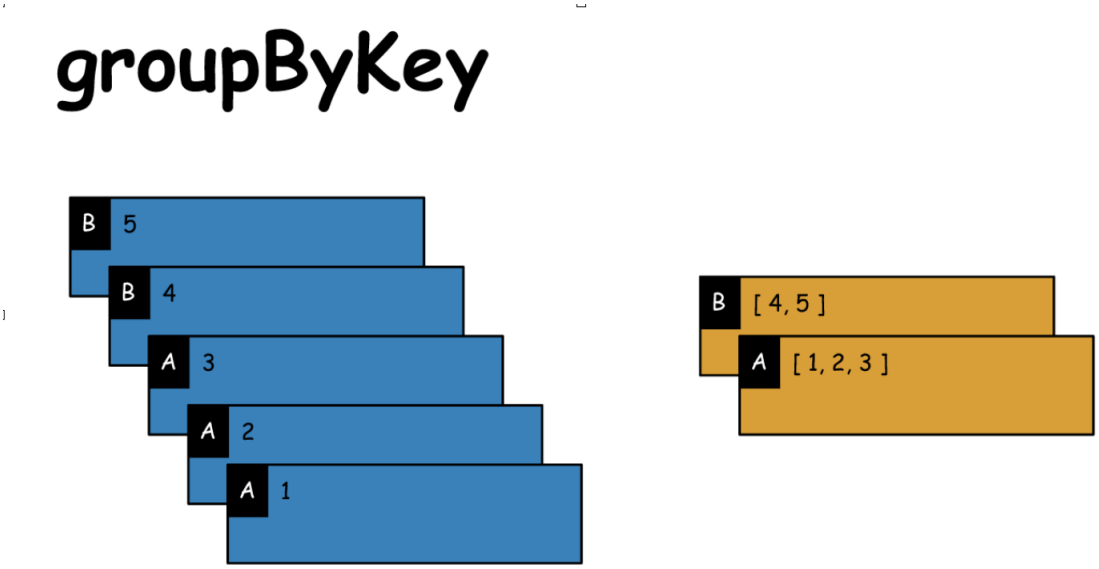

Figure 5.7. GroupByKey Operation of Spark RDD

fashions as shown in Figure 5.9 and Figure 5.11 by utilizing the *SeismicVolume* API :

repartition(planesPerMap:Int,overlapPlanes:Int):SeismicVolume[T].

5.4.1 Aggregation

As shown in Figure 5.9, aggregated data distribution could be achieved by utilizing the RDD group operations we mentioned in Volume Data 3D transposing section. The solution of this problem is to re-indexing all the slices in parallel through RDD map function by arranging an unique key to multiple data splits, then use RDD *groupByKey()* API to repartition the dataset. An aggregated dataset could have multiple slices in a single data split.

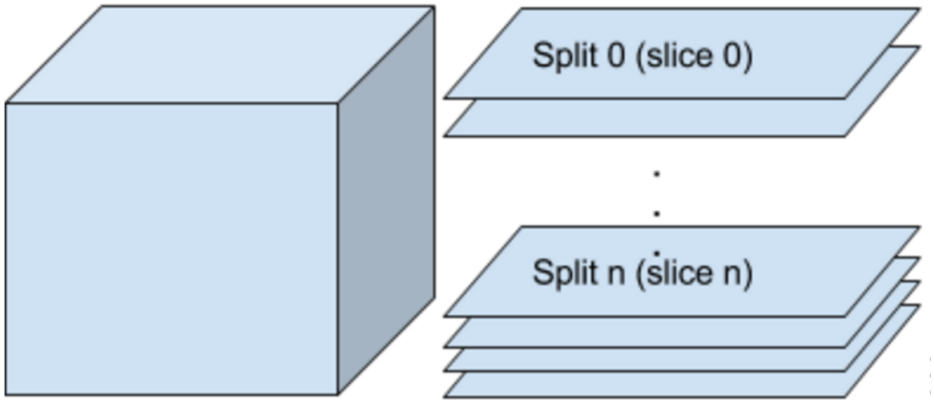

Figure 5.8. Default Distribution Fashion of Volume, planesPerMap=1, overlap=0

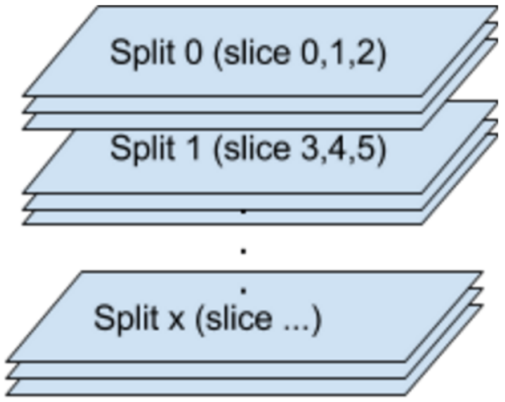

Figure 5.9. Aggregated Distribution of Volume, planesPerMap=3, overlap=0

5.4.2 Overlapping

The *repartition()* method not only lets users change the size of distributed splits, more powerfully, it allows developer to set the overlapped data areas between splits and to access the overlapped parts in each split. Practically, some applications may have strong data dependences in their logic which is very hard to parallelize the solutions. For example, the stencil computation needs lots of data communications

between neighbor units in each step, which is impossible to run it in parallel with other MapReduce frameworks.

To resolve this problem, *repartition()* API generates the head and tail boundaries RDD on top of the aggregated SeismicRDD and indexing them according to the related partition numbers. Finally, the boundaries are appended to each related data split through RDD *zip()* API to generate a new *SeismicVolume* instance which contains overlapped SeismicRDD. Figure 5.10 shows the process of boundary RDD solution.

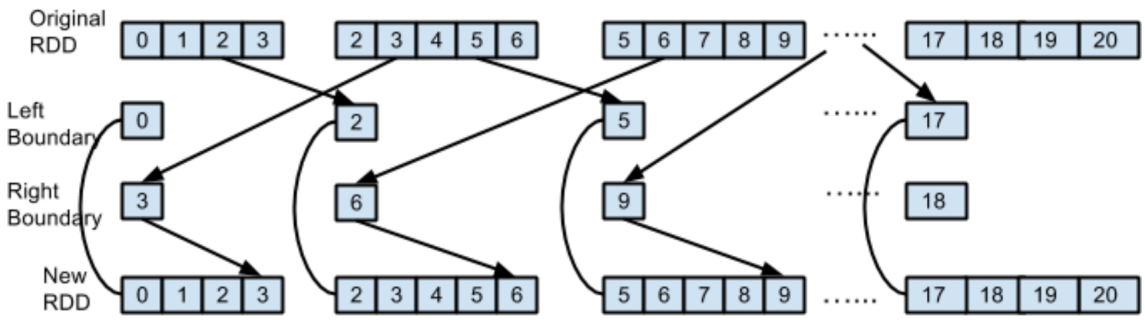

Figure 5.10. The Boundary RDD Solution of Overlapping Problem

Therefore, this method is not only capable of tuning the performance of distributed tasks, but also simplifies the stencil-style computation requiring neighbor communication, which is not easy to parallelize in MapReduce programming mode. We have developed a more complicated 3D stencil use case with 3D spacing overlapping which will be presented in detail in following Experiments chapter. It could be used for resolving seismic 3D attributes computation problem.

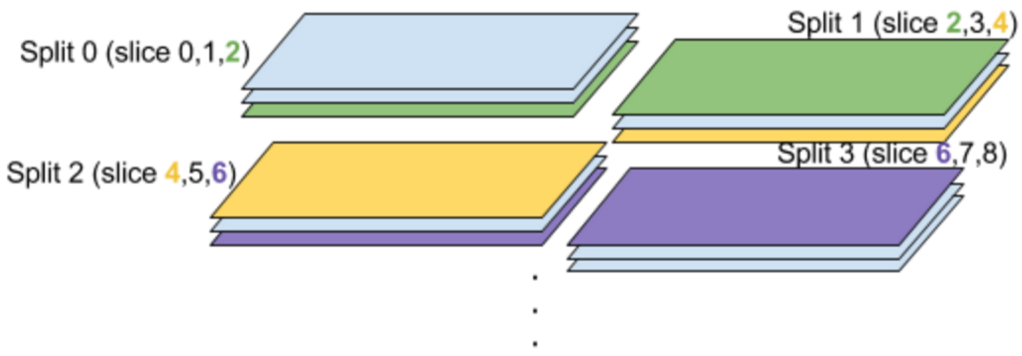

Figure 5.11. Overlapped distribution of Volume, planesPerMap=1, overlap=1

5.5 User Defined Function Mapping

As mentioned in Introduction chapter, one of the project objective is to provide an easy-to-use solution for geophysicists or data scientists to apply their programs on big data platform without concerning the parallelism and code reconstruction. To achieve this, *SeismicVolume* provides *applyMap()* API to allow developer to apply user-defined functions on any direction of the SeismicRDD in parallel.

The prototype of this API is *applyMap(direction:Int, f:(T-U))*. The first parameter *direction* indicates the direction that users would like to apply the function on. The other parameter *f:(T-U)* is a standard spark RDD key-value pairs operation callback function, which feeds the function distributed volume data with key-value forms in parallel. The data length in each key-value function depends on the specified distribution parameters of the target *SeismicVolume*. Users could apply any operation or computations on the given input data, and an output key-value pairs is required for the return value. This callback function will be executed along with the related data distribution in parallel. After execution, it will generate a new *SeismicVolume* object containing the processed and distributed dataset output by

user-defined function. Figure 5.12 shows the example code for users to apply their function.

```
def fftFunc[T:ClassTag](e:(Long,Array[T])):(Long,Array[T]) = {
    val k = e._1;
    val v = e._2;

    var w = kVar;
    var h = jVar;
    if (dir == 2) {
        w = kVar;
        h = iVar;
    }
    else if (dir == 4) {
        w = jVar;
        h = iVar;
    }

    val planeSize = w * h;
    val num = v.length/(planeSize);
    var res = new Array[T](0);
    for(i <- 0 until num) {
        val line = new DenseMatrix(h, w, v.slice(planeSize*i,planeSize*i+planeSize));
        val lined= line.mapValues(a=>a.asInstanceOf[Double]);
        val fft = fourierTr(lined);
        val tmp = fft.mapValues(a=>a.abs.asInstanceOf[T]).data;
        res = res ++ tmp;
    }
    (k,res)
}

println("start to apply FFT function in dir " + dir + ": " + Calendar.getInstance().getTime());
val newsv = sv.applyMap(dir, 0, fftFunc);
println("start to save result as : " + path + " " + args(3) + " " + Calendar.getInstance().getTime())
newsv.save(path, tp);
```

Figure 5.12. Sample Code for Applying User-defined function in Parallel

5.6 Parallel Templates

Data distribution plays an important role in parallel programs to achieve scalable performance. However, most scientists and researchers in petroleum industry do not have enough big data background knowledge to support them to convert their work to Spark applications. To overcome the usability barriers, we developed several parallel templates based on Seismic Data Analytics SDK APIs to make it be easily used by

domain algorithm designer other than computer scientists.

These templates defines the data distributions and parallel computation so that users can simply select the right templates for their algorithms without handling the data distribution and parallelism details. Three templates currently include: Trace, Line and Sub-volume. Each template can handle one or more volumes, and will output one or more volumes. Trace template is simple, in which the input is a 2D array (dimension 1 for number of volumes and dimension 2 for 1D trace data), and output is also a 2D array. Line template defines a 3D array as input and a 3D array as output respectively (dimension 1 for number of volumes and dimension 2 for 2D slice data, line is the petroleum terminology). Sub-volume template is a powerful solution to handle data 3D data distribution with overlaps, in which both input and output are 4D array (dimension 1 for number of volumes and dimension 2 for 3D overlapped sub-volume data). The Sub-volume template outputs data without any overlapping. Users can specify parameters about how to distribute data as well as the overlapped areas.

Figure 5.13 shows an example of the Line Template and Figure 5.14 shows the example of Sub-volume Template. Both of them are straight forward and require very few knowledge about parallel computing. The only thing users should be aware of is the distribution grain of input/output data, which is specified in the distribution parameter when users create the template instance. As shown in Figure 5.15, it demonstrates the execution of an Sub-volume Template instance, in which the distributed input sub-volume size in dimension I and J (for each parallel *proc()* callback function) is set to 26 x 21(The sub-volume size in dimension K is always the complete length of iTrace/xTrace, which is the minimal unit for seismic computation), and the overlapping size is 4 in both I and J direction (Each data split extends to 30 x 25 in I x J).

```
class LineApp(override val inputList:Array[SeismicVolume[Float]], override val distPrm:DistParams)
    extends SeismicAppLineTmpl[Float,Float](inputList, distPrm)
{
    override def proc(input:Array[Array[Array[Float]]]):Array[Array[Array[Float]]] = {
        val len = input.length;
        val output = new Array[Array[Array[Float]]](len);
        for(i <- 0 until len) {
            output(i) = input(len-1-i);
        }
        output;
    }
}
```

Figure 5.13. Sample Code of Line Parallel Template

```
class TraceOlpFilter(override val inputList:Array[SeismicVolume[Float]], override val distPrm:OverlapDistParams)
    extends SeismicAppTraceOverlapTmpl[Float,Float](inputList, distPrm)
{
    override def proc(inputData:Array[Array[Array[Array[Float]]]],
                      olpWindow:TraceOverlapWindow):Array[Array[Array[Array[Float]]]] = {

        val len = inputData.length;
        val outData = new Array[Array[Array[Array[Float]]]](len);

        for (i <- 0 until len) {
            val centerVolume = new Array[Array[Array[Float]]](olpWindow.olpCenterI);

            /*extract center data from olp window*/
            for (o1 <- olpWindow.olpHeadI until (olpWindow.olpHeadI + olpWindow.olpCenterI)) {
                val centerSlice = new Array[Array[Float]](olpWindow.olpCenterJ);

                for (o2 <- olpWindow.olpHeadJ until (olpWindow.olpHeadJ + olpWindow.olpCenterJ)) {
                    val trace = inputData(i)(o1)(o2);
                    centerSlice(o2 - olpWindow.olpHeadJ) = trace;
                }
                centerVolume(o1 - olpWindow.olpHeadI) = centerSlice;
            }
            outData(i) = centerVolume;
        }
        outData;
    }
}
```

Figure 5.14. Sample Code of Sub-volume Parallel Template

```
    /*create an overlap layout:
    I x J: 26 x 21 in center and with 4 traces overlapping in each of I and J direction
    */
    val olpSpec = new OverlapDistParams(26, 21, 4, 4);

    val olpTmpl = new TraceOlpFilter(invols, olpSpec);

    // Run the filter to produce filtered volume
    val fvol = olpTmpl.exec(INLINE);
    val res = fvol(0);
```

Figure 5.15. Sample Code of Executing Sub-volume Parallel Template

CHAPTER 6

EXPERIMENTS, RESULTS AND ANALYSIS

To verify and demonstrate the scalability that the user application could achieve through utilizing DMAT, we conducted a series of experiments, including data transposing and 3D stencil calculation, an overlapping-calculation application.

6.1 3D Volume Transposing

6.1.1 Use Case

This section presents the experiments of the 3D transposing problem mentioned in previous chapter. The dataset we used for transposing (from I to J direction) experiment is a 300GB seismic 3D volumetric data, which is 31017 x 97223 x 31 in I x J x K direction with float data type. We design the experiment to verify the performance of transposing is scalable and is mainly affected by the data distribution configuration and the amount of available hardware resource(the total count of cores).

6.1.2 Statistics and Analysis

6.1.2.0.1 Scalability to the Number of CPU Cores We conducted the transposing experiment on a cluster with 24 nodes, each node has 12 cores(24 cores in Hyper-threading) and 48GB available DRAM. The total CPU cores is 288(576 in Hyper-threading). Figure 6.1 shows the performance metrics of this experiment. The x in *Transpose[x]* stands for the aggregation parameter we applied on the dataset. As mentioned in previous chapter, the value of aggregation parameter x determines the number of planes of each partitions thus to control the size of each data split.

From the metrics, we got two bar chart which shows the time consuming for the

Cores	LoadFromFile(sec)	Transpose[1](sec)	Transpose[16](sec)	Transpose[64](sec)	Transpose[128](sec)
36	711.02	30246.483	604.083	444.867	433.9
72	531.849	15941.077	334.019	283.037	263.478
144	445.16	8865.005	210.677	178.212	168.96
288	404.325	5134.525	211.871	136.755	142.68
576	398.696	3914.722	179.026	131.292	141.675

Figure 6.1. Transposing Experiment Statistics on Cluster with 288(576) cores.

transposing of aggregated and non-aggregated data. It clearly demonstrates that the performance scalability on dimension of the count of CPU cores is promised, as shown in Figure 6.2 and Figure 6.3. The performance difference between the transposing on aggregated and non-aggregated seismic volume data will be addressed in the following paragraph.

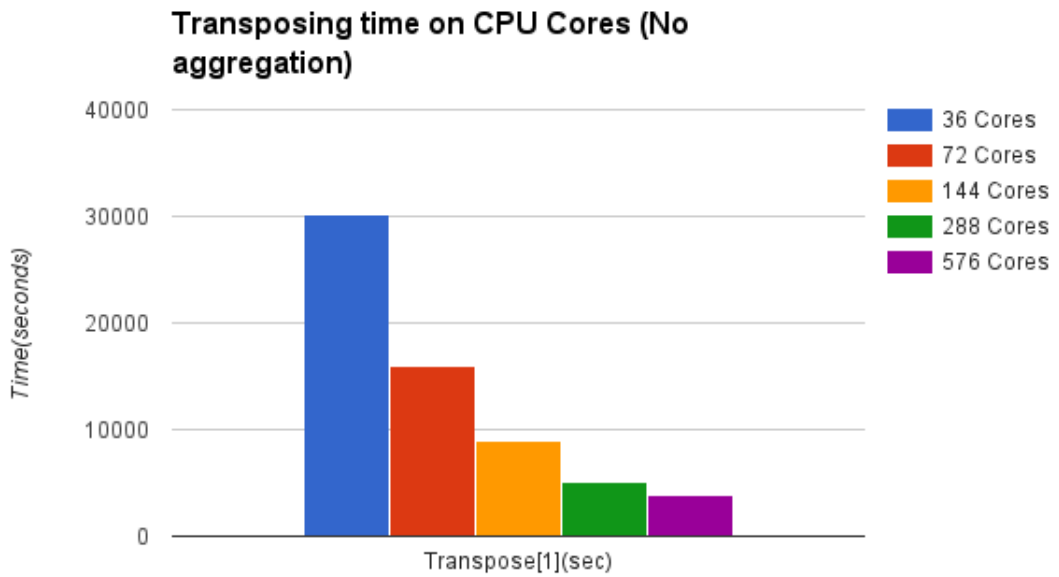

Figure 6.2. Transposing Time on CPU Cores without Aggregation

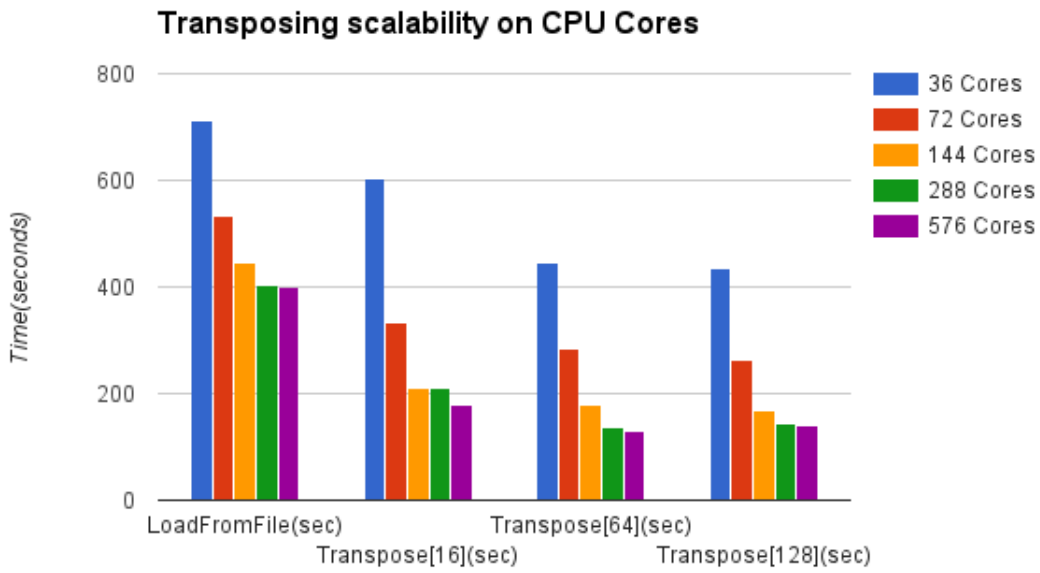

Figure 6.3. Transposing Time on CPU Cores with Aggregation.

6.1.2.0.2 Scalability to the Fashion of Data Distributions There is another important factor, the data distribution fashion, will also affect the performance of transposing. As we mentioned in previous section, the transposing program will perform some shuffle operation on volumetric RDD. Shuffle operations [10] is Spark's mechanism for re-distributing data so that it could change the data organization across partitions. This typically involves copying data across executors and nodes, which makes the shuffle a complex and costly operation. To reduce the shuffle costs, we could aggregate the SeismicRDD data splits to reduce the number of partitions thus to decrease the amount of data need to exchange during 3D transposing on each data split. Figure 6.4 shows the performance improvement when we increasing the aggregated planes number (x in *Transpose[x]*) of each data distribution.

However, the curve of performance trends to flat when the aggregation increases

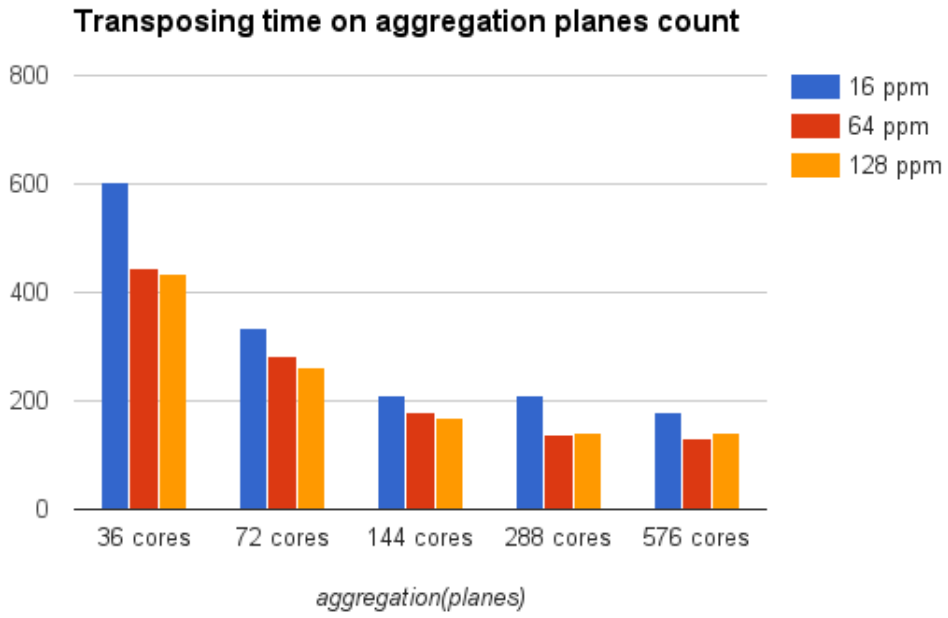

Figure 6.4. Transposing Time on Aggregated Planes(ppm: planesPerMap).

to a higher level. That is caused by the performance trade-off when reducing the distribution partitions, which will lead to a lower cluster CPU cores utilizing rate. Therefore, to achieve a reasonable performance for transposing operation, developer needs to considerate the trade-off between distribution configurations and data shuffling.

Figure 6.5 and Figure 6.6 show the CPU utilization and memory usage of each node when the transposing experiment was running on the cluster. These system statistic visualization views were generated by a free software NMONVisualizer [6], which is a Java GUI tool for analyzing Nmon performance files.

6.1.2.0.3 Cross-Verification on the Third-party Cloud Platform: XSEDE Cluster To further verify the scalability, we also setup the same experiment on the XSEDE supercomputing cluster [12]. We request 44 nodes from XSEDE cluster, which has

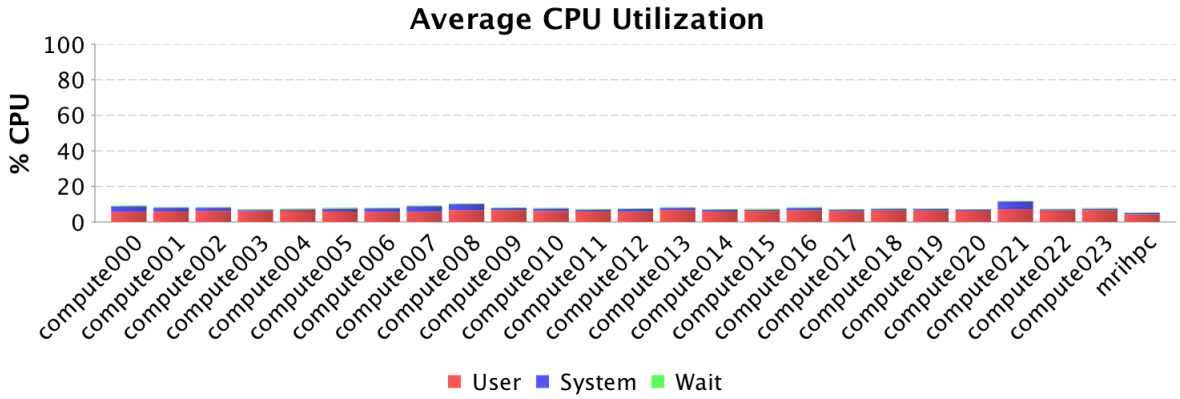

Figure 6.5. CPU utilization statistics from NMONVisualier.

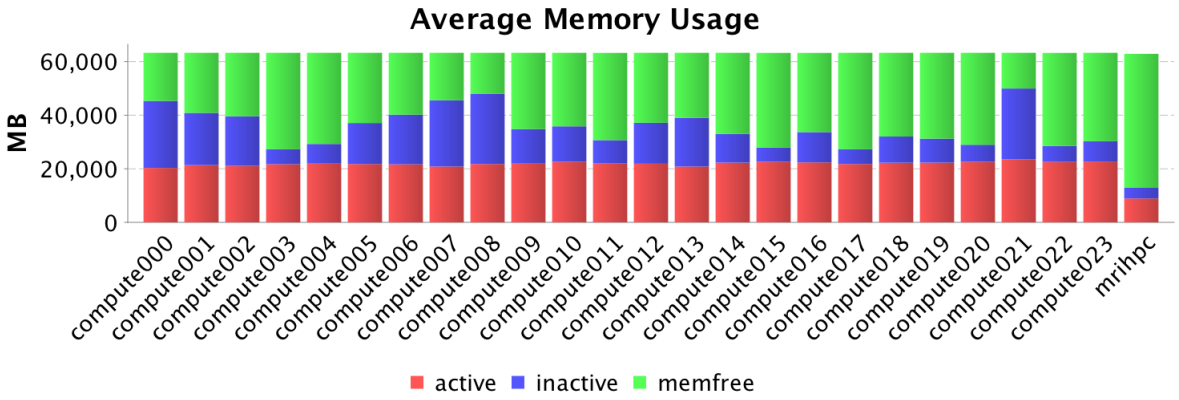

Figure 6.6. DRAM utilization statistics from NMONVisualier.

12 cores in each node. Figure 6.7 shows the result statistics of this experiment. As shown in Figure 6.8, we conduct the experiment to test the scalability of case transposing the same volumetric data from I to J direction, aggregation is 16 planes, and the result also verified the scalability of this distributed application.

Cores	Transpose Time(sec)
192 cores	494.434
240 cores	431.174
288 cores	417.077
336 cores	331.337
384 cores	286.763
432 cores	268.772
480 cores	259.086
528 cores	238.074

Figure 6.7. The Transposing Performance Statistics on XSEDE Cluster

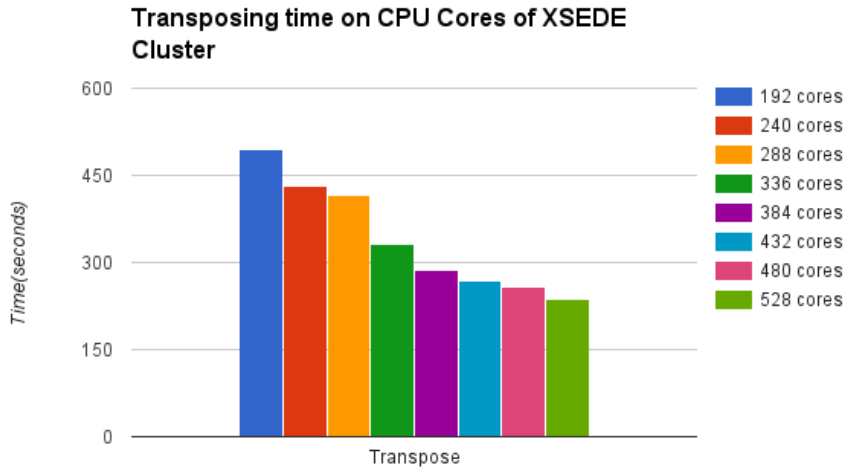

Figure 6.8. The Performance Scalability of Transposing on XSEDE Cluster

6.2 3D Stencil Application

6.2.1 Use Case

Stencil computations are most commonly used in context of scientific and engineering applications such as signal and image processing, computer simulations etc. Stencil itself represents an iterative kernel that updates array elements according to fixed pattens [11]. The optimization on stencil computation has been well studied in

[19],[21] and [17]. However, most of these optimizations focus on single node implementation with GPU or multi-core CPUs. In [24], [23] and [22], the authors provide a parallel implementation with Spark RDD, which gives a scalable solution for big data that can not host on a single node. In this experiment, we use the defined parallel templates in DMAT to implement stencil computations, and test their performance as well as scalability.

6.2.2 Statistics and Analysis

The dataset we choose to conduct our experiment is called Penobscot dataset, which is actual seismic image data with 3D dimension size 600x481x1501. The cluster consists of 8 nodes, in which one is management node and other 7 nodes are computation nodes. Each node was equipped with Intel Xeon E5-2690 Sandy Bridge CPU (2.9 GHz, 16 Cores or 32 Cores with Hyper-threading support), 128GB DDR3 memory and are inter-connected with 1GB ethernet. JDK 1.8.0_40, Hadoop 2.5.1 and Spark 1.6.1 are used for compiling and running applications. Wall clock is used to get the running time, and Nmon/Spark Web UI are used for performance analysis.

For the algorithm, we use a variant of Jacobi iteration, which uses a 3D subvolume with dimension size of 3x3x3 as input, and in the computation, each new output value at (i, j, k) is the average value of 26 surrounding samples plus itself. In the case of 3x3x3 subvolume, the overlap area is 1. For the sequential codes, we just split the big 3D data file into small partition and each partition includes several 2D planes (the overlap between partition is one 2D plane), and then use 3 nested loops to compute the average value. For the parallel codes using DMAT, we use the overlapped template, which specify parameters both in I and J directions. Different configurations of cores and numbers of planes in each partition are set to check performance and scalability. The numbers of cores (28, 56, 112, 224) are used for

each test case respectively to verify the scalability of parallel codes. Within each configuration of cores, we use different combinations of dimension size (1, 2, 4, 8) in I and J directions. For instance, I(4) and J(2) mean that the dimension size of input subvolume 6x4, which comes from (4+2*1)x(2+2*1) in the case of overlap is 1, which is shown in Figure 6.9

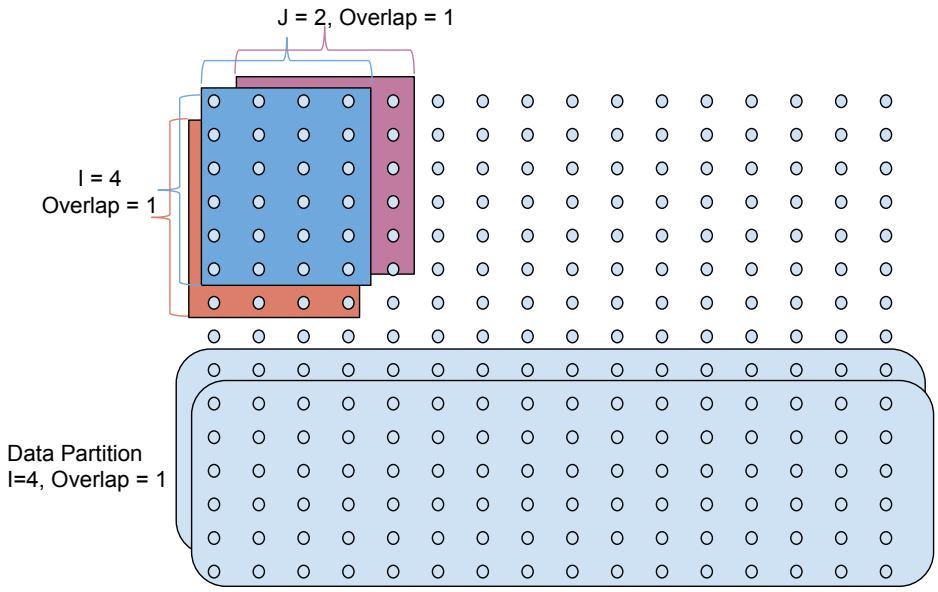

Figure 6.9. The Data Distributation and Input of Overlaop Template

Figure 6.10 and Figure 6.11 show the speed up of parallel codes on Spark with 28 cores and 224 cores to the sequential codes respectively. From these two figures, the changes on number of J have little impact on the performance, because it does not change the number of planes in each partition and the number of partitions. In the template implementation, two nested loops are used for feeding the input of each

stencil kernel. However, the change on number of I tells how the SDK distributes the data and the number of partitions, thus will determine how many tasks are need to finish that stage of in the job, and each task need to be assinged one thread or one core to undertake the computation.

In the case that size of I is 1, it gets the best speed up in all test cases. It seems to be abnormal that the performance decreases from 1 plane of I to 2 planes of I. Increasing I will enlarge the size of each partition as shown in Figure 6.9, however, the amount of computations keeps constant. In this stencil computation, we iterate several times to reach the balance.

At the beginning of each loop, the data need to be repartitioned based on the overlap parameter, since the latest edge data need to be updated to each partition for next iteration. In the process of repartition, it needs to get planes at the left and right edge, sort them by key and zip with original latest results, which trigger data shuffle in Spark. The bigger the size of partitions is, the more time it takes to shuffle them. The 'Shuffle Read Blocked Time' increases drastically from 1 plane of I to 2 planes of I, which accounts for the performance decreasing. Figure 6.12 shows the best speedup with different configurations of cores, in which the scalability is obvious while increasing the number of cores.

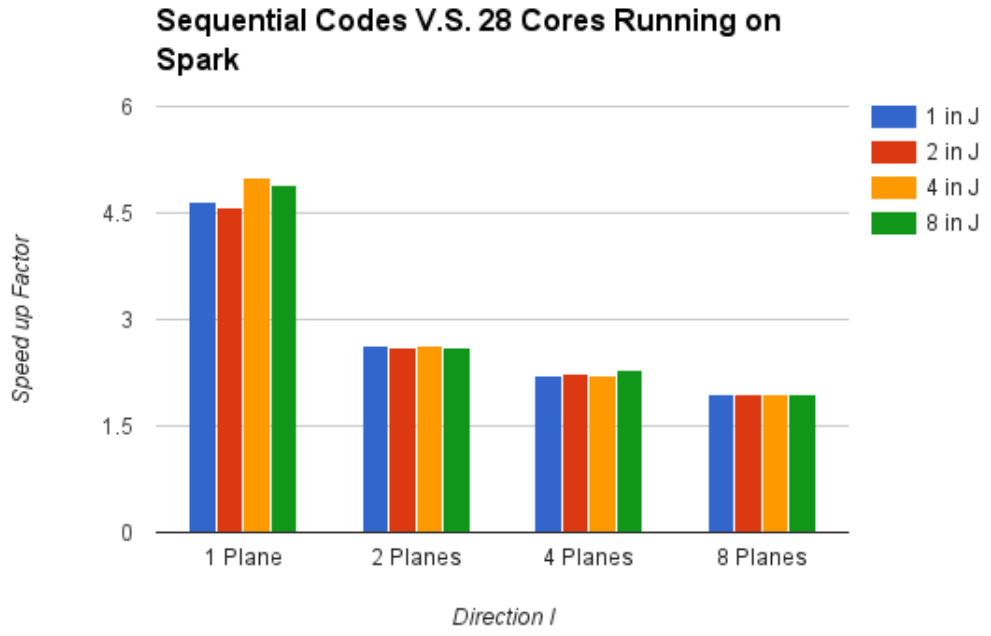

Figure 6.10. The Speedup of Parallel Template Codes with 28 Cores to Sequential Codes

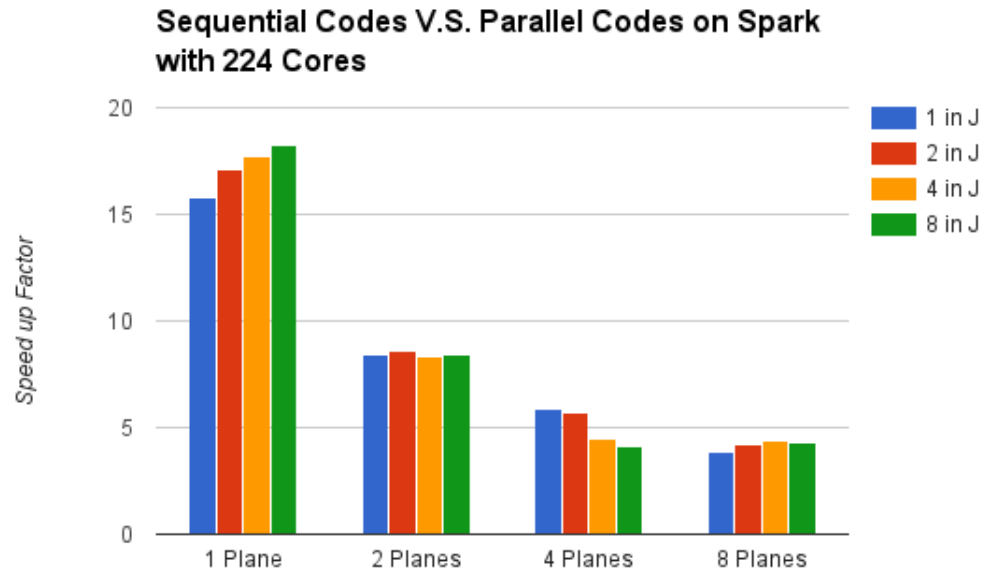

Figure 6.11. The Speedup of Parallel Template Codes with 224 Cores to Sequential Codes

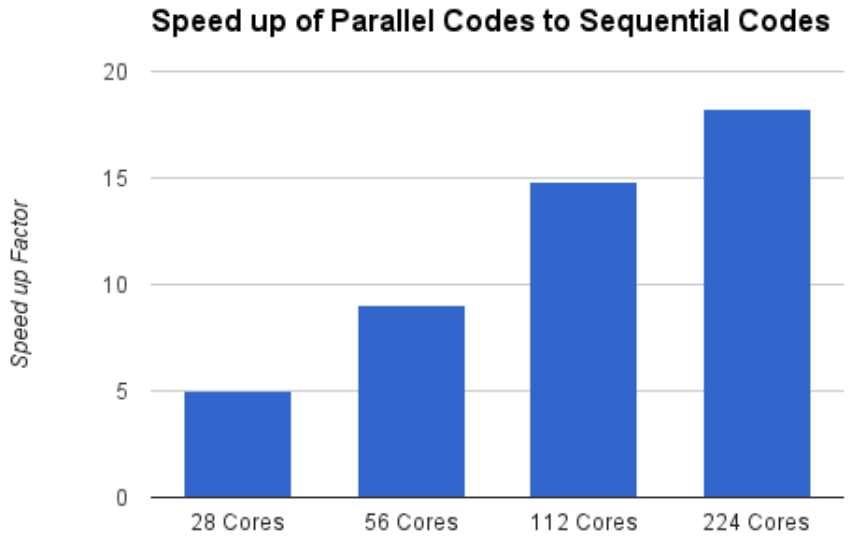

Figure 6.12. The Best Speedup of Parallel Templates for Stencil Computation

CHAPTER 7

SEISMIC DATA ANALYTICS PLATFORM

In oil & gas companies, the data analytics software is not only used by scientists and researchers, but also the employees who do not have the background of computer science. To allow those users to browse the seismic data and to generate analytics result, we developed some user-friendly utilities on top of Seismic Data Analytics SDK to simplify the way for deploying general purpose applications on Spark-Hadoop platform, which include a distributed data server for remote data access, the web interfaces for remote data visualization and user-defined workflow. With these tools, the users, not only the developers, are able to easily facilitate their works with this Seismic Data Analytics Platform in minimal efforts.

7.1 Data Server and Remote Web Visualization

In petroleum industry, an important application scenario is data visualization, which allows user to browse and analyze seismology features of seismic data in 3D spacing. With visualization tools, computer renders the seismic data to 3D graphic views and allows user to browse and manipulate the graph along any direction, which inspired the geophysicists and data scientists to develop various of useful models. However,traditional visualization tools in industry are only capable of handling small datasets, or render only few segments of the big datasets at a time. The performance of big dataset visualization has long been the critical bottleneck of regular workflow of industry.

To resolve this problem, we developed a web-based remote data visualization service which is able to load and render the seismic data for 3D visualization in

real-time. Figure 7.1 shows the framework of this service.

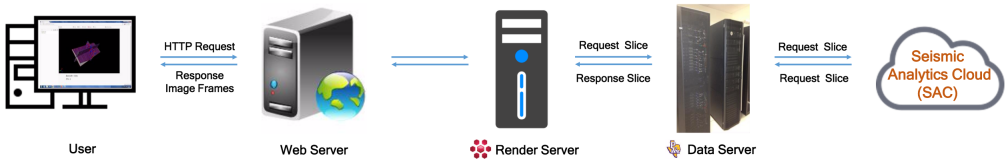

Figure 7.1. The Framework of Remote Web Visualization Service

This solution is developed on top of Seismic Data Analytics SDK APIs and FEI Digital Rock Visualization Packages. We design and implement the DataServer service on top of Seismic Data Analytics SDK, which could load big seismic datasets from HDFS, transpose and store them to SeismicRDD in backends then feed the requested data in any direction to FEI visualization service in real-time. The FEI Digital Rock Visualization package is developed by FEI, the company which has been focus on featuring Digital Rock technology and solutions many years [3].

Finally, the output rendered data view is presented through web interface and allows user to browse and manipulate in any direction. With this solution, user do not need to install complicated visualization tools and packages since the platform handles everything on server-side. More importantly, the performance of rendering big seismic data is improved and scalable after facilitated by Seismic Data Analytics SDK. Figure 7.2 show s the web interface of this solution.

7.2 Web-based Workflow Platform

Another service we developed is a web-based workflow platform, which provides a friendly web interface to make cloud platform easy to use without programming, with which users could create workflow with drag and drop, could run the created

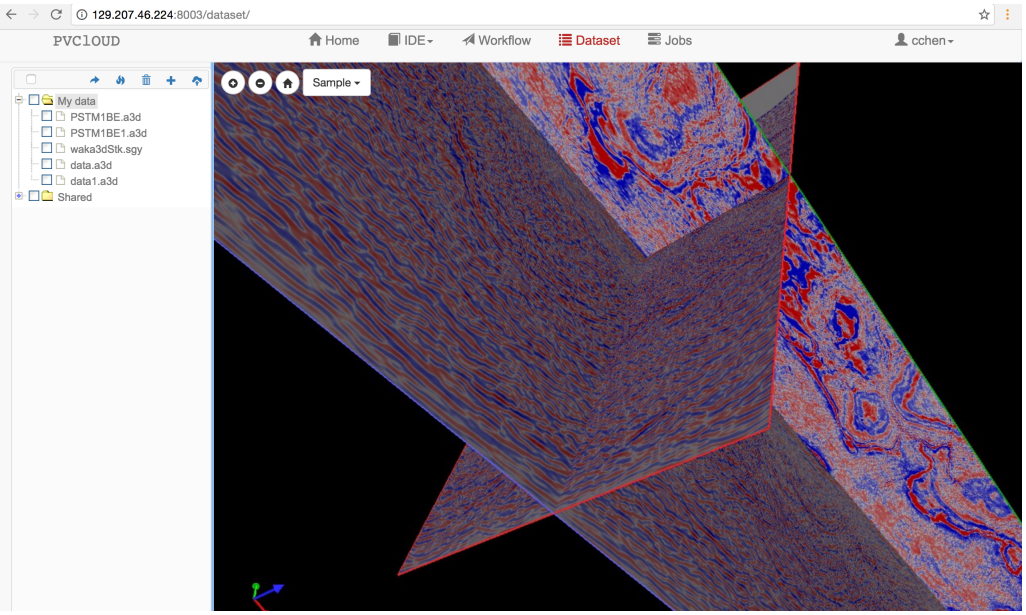

Figure 7.2. Visualization Web Interface

workflow and check the results through visualization model. The Workflow Web Service is implemented on an open source project Clowdflows [2], which provides a Django based framework to develop a customized widget and to manage widgets conveniently. As a free and open source web application framework written in Python, Django follows the Model, View and Controller (MVC) architectural pattern, so it is suitable for interact with both cloud service and web client.

The client side view of workflow is shown as Figure 7.3. Users could select widgets and specify seismic data files as input to construct and customize their workflows. Each widget in the workflow view is an independent application component which could be implemented on top of Seismic Data Analytics SDK. A widget acquires at least one port as input or output thus multiple widgets are able to be combined to a workflow by multiple pipelines which connect the ports of all widgets. Each pipeline in the workflow view indicates a data communication, which transports data between

widgets and drives the execution of whole workflow. The workflow framework implemented by Python code includes Django views (GUI), Django models (widgets data management) and topological sorting algorithm (connections check). The data files are stored in HDFS of the cloud platform and could be browsed and selected from the navigation trees on the editor page.

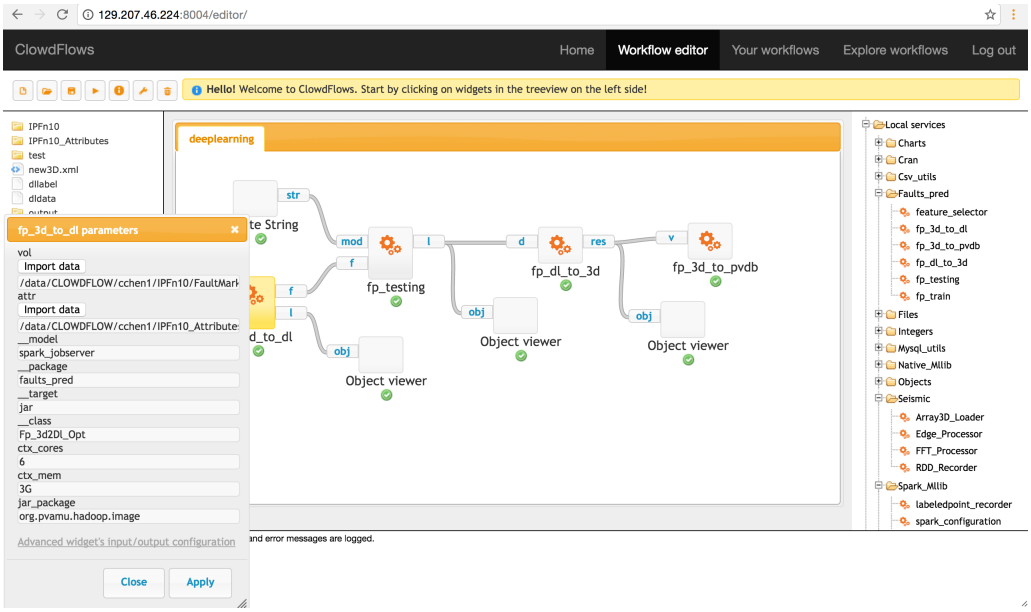

Figure 7.3. Workflow Web Interface

CHAPTER 8

CONCLUSIONS AND FUTURE WORK

The thesis presents a scalable and distributed Seismic Data Analytics toolkit that is implemented on top of Apache Hadoop and Spark big data platforms. It is an attempt to apply HPC optimizations to big seismic data analytics applications, and simplify the parallelism efforts. The experiments and related analysis given in this thesis also shows that this toolkit provides promised scalability, by which performance enhancement can be achieved by increasing cluster hardware resources or tuning distribution parameters without refactoring the application code.

In the future, advanced data distribution strategies such as tiling and bricking with 3D overlapping will be implemented for further improvement of the application performance. More application utilities and web service will be developed for data browsing and analytics purposes. We will put more efforts to provide a comprehensive Seismic Data Analytics Platform as a Service(PaaS) to facilitate the works of scientists.

REFERENCES

[1] 5-google-projects-changed-big-data-forever.

[2] Clowdflows. https://github.com/xflows/clowdflows/wiki. [Retrieved: June, 2016].

[3] FEI Digital Rock. https://www.fei.com/oil-gas/. [Retrieved: June, 2016].

[4] Hadoop Introduction. http://hadoop.apache.org/. [Retrieved: January, 2014].

[5] IndexedRDD for Apache Spark.

[6] NMONVisualizer. http://nmonvisualizer.github.io/nmonvisualizer/. [Retrieved: June, 2016].

[7] Reflection seismology. http://en.wikipedia.org/wiki/Reflection_seismology. [Retrieved: June, 2015].

[8] Seismic Inline. http://www.glossary.oilfield.slb.com/Terms/i/in_line.aspx. [Retrieved: July, 2015].

[9] Spark Lightning-fast cluster computing. http://spark.incubator.apache.org/. [Retrieved: January, 2014].

[10] Spark RDD Shuffle Operation. http://spark.apache.org/docs/latest/programming-guide.html#shuffle-operations. [Retrieved: June, 2016].

[11] Stencil code. https://en.wikipedia.org/wiki/Stencil_code. [Retrieved: June, 2016].

[12] The Extreme Science and Engineering Discovery Environment. https://www.xsede.org/overview. [Retrieved: June, 2016].

[13] What is AOV. http://www.agilegeoscience.com/blog/2011/6/6/what-is-avo.html. [Retrieved: July, 2015].

[14] What is Scala. http://www.scala-lang.org/what-is-scala.html. [Retrieved: June, 2016].

[15] Baaziz A. How big data is changing the oil and gas industry, December 2013.

[16] SEG Technical Standards Committee. SEG-D, Rev 2.1. http://www.seg.org/documents/10161/77915/seg_d_rev2.1.pdf, January 2006.

[17] Kaushik Datta, Mark Murphy, Vasily Volkov, Samuel Williams, Jonathan Carter, Leonid Oliker, David Patterson, John Shalf, and Katherine Yelick. Stencil computation optimization and auto-tuning on state-of-the-art multicore architectures. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, SC '08, pages 4:1—-4:12, Piscataway, NJ, USA, 2008. IEEE Press.

[18] Adam Farris. How big data is changing the oil and gas industry. http://analytics-magazine.org/how-big-data-is-changing-the-oil-a-gas-industry/, November 2012.

[19] Dongni Han, Shixiong Xu, Li Chen, and Lei Huang. PADS: A Pattern-Driven Stencil Compiler-Based Tool for Reuse of Optimizations on GPGPUs. In *the ICPADS Conference: The 17th IEEE International Conference on Parallel and Distributed Systems*. IEEE.

[20] Andrea De Mauro, Marco Greco, and Michele Grimaldi. A formal definition of big data based on its essential features. *Library Review*, 65(3):122–135, 04 2016.

[21] Anthony Nguyen, Nadathur Satish, Jatin Chhugani, Changkyu Kim, and Pradeep Dubey. 3.5-D Blocking Optimization for Stencil Computations on Modern CPUs and GPUs. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '10, pages 1–13, Washington, DC, USA, 2010. IEEE Computer Society.

[22] Y. Yan, C. Chen, and L. Huang. A productive cloud computing platform research for big data analytics. In *2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 499–502, Nov 2015.

[23] Y. Yan, L. Huang, and L. Yi. Is apache spark scalable to seismic data analytics and computations? In *Big Data (Big Data), 2015 IEEE International Conference on*, pages 2036–2045, Oct 2015.

[24] Yuzhong Yan. A template-based seismic data analytics cloud platform. Master's thesis, Prairie View A&M University, 100 University Dr, Prairie View, TX 77446, United States, 12 2015.

CURRICULUM VITA

Chao Chen

Prairie View A&M University
Department of Computer Science
P.O. Box 519, Prairie View, 77446

Phone: (832) 910-4615
Email: cchen.rough@gmail.com

Education

M.S. Computer Science, Prairie View A&M University, 2016.

B.A. Electronic Information Engineering, Southwest University of Science and Technology, 2005.

Employment

Prairie View A&M University, Research Assitant, 2015–2016.

Sunmedia Technology, Software Engineer, 2006–2014.

Professional Skills

Solid background of computer science and software engineering, as well as strong learning and communication ability.

Proficient at Linux, Android and Embedded OS development, strong products development experiences including digital camera, digital photo frame and tablet.

Familiar with big data and cloud computing platform (Hadoop, Spark).

Publications & Achievements

A Scalable and Productive Workflow-based Cloud Platform for Big Data Analytics, *2016 IEEE International Conference on Big Data Analysis*, (ICBDA 2016) Hangzhou, China, March 12-14, 2016

A Productive Cloud Computing Platform Research for Big Data Analytics, *In Proc. of IEEE CloudCom 2015*, Nov 30-Dec 3, Vancouver, Canada