Acquisition of Seismic, Hydroacoustic, and Infrasonic Data with Hadoop and Accumulo

by William N. Junek, Charles A. Houchin, Joseph A. Wehlen III, John E. Highcock II, and Marcus Waineo

ABSTRACT

Recent developments in distributed data systems now allow faster ingestion and processing of larger quantities of time-series data than available in current seismic, hydroacoustic, and infrasonic analysis platforms. However, the data model and storage architecture of these systems are significantly different than those in use today. We developed a data acquisition and signal analysis platform using a relatively inexpensive cluster of commodity computing hardware running a Hadoop Distributed File System, an Accumulo database infrastructure, and the Zeppelin web-based analytics tool suite. The Accumulo data model allows individual waveform samples and their associated metadata to be stored as discrete rows in the database. This is a significant departure from traditional storage practices, in which continuous waveform segments are stored with their associated metadata as a single entity. Our design allows for rapid table scans of large data archives within the Accumulo database for locating, retrieving, and analyzing specific waveform segments directly. The system infrastructure is horizontally scalable, which allows additional data acquisition and processing resources to be added simply through the addition of new data nodes. Easy scalability permits the system to accommodate the ingestion and analysis of new data as the network of seismic, hydroacoustic, and infrasonic sensors grows. The barrier of entry for establishing a functional data acquisition system is relatively low for which a proof of concept prototype can be developed on a limited budget and later scaled as storage and processing requirements dictate.

INTRODUCTION

The U.S. National Data Center (NDC) geophysical data processing system monitors international compliance with nuclear-test-ban treaties. Performing this task requires the near-real-time acquisition, processing, and analysis of waveform data from a global deployment of seismic, hydroacoustic, and infrasonic (SHI) sensors that comprise the United States Atomic Energy Detection System and International Data Center (IDC) networks. In addition, data from other publically available stations are used, on a case-by-case basis, to augment network coverage during event analysis. The current U.S. NDC processing infrastructure is nearly 30 years old and relies on a collection of C-based software applications to acquire and parse data. Parsed data are routed to a series of processing functions, where results are stored in an underlying UNIX-based file system and ORACLE relational database. This processing architecture has functioned well for nearly three decades but is becoming increasingly difficult to maintain and modify to accommodate the growing amount of data it is expected to process. However, recent innovations in distributed, data storage, and processing architectures provide a collection of new, efficient, scalable, and relatively inexpensive tools for handling large datasets.

The use of distributed data processing architectures in seismology is beginning to gain mainstream momentum. Several published studies have demonstrated the viability of these systems for processing decades of seismic waveforms that were acquired from a global distribution of hundreds of seismometers (Addair *et al.*, 2014; Dodge and Walter, 2015; Magana-Zook *et al.*, 2016). These studies have been successful in extracting meaningful geophysical insights through long-term trend analysis of data stored in offline databases that are not used in real-time processing.

In January 2016, the U.S. NDC initiated a research and development project to study the utility of distributed data processing architectures that are designed for storing large datasets and optimized for high-speed database input/output operations for near-real-time use. Built on the Hortonworks Data Platform, the experimental data acquisition system uses an Apache Hadoop Distributed File System (HDFS), an Apache NiFi data parsing and routing scheme, and an Apache Accumulo database infrastructure. NiFi provides a powerful, reliable, and scalable architecture for data collection, routing, and transformation, which permits the creation of custom processors to manipulate multiple file types and additional transformations. Accumulo is a sorted, distributed key–value, store with cell-based access control, which provides extremely fast access to data in massive tables (Cordova *et al.*, 2015). This development effort resulted in the creation of two new NiFi processors that facilitate the movement of data directly into Accumulo. We also deployed a collection of Apache
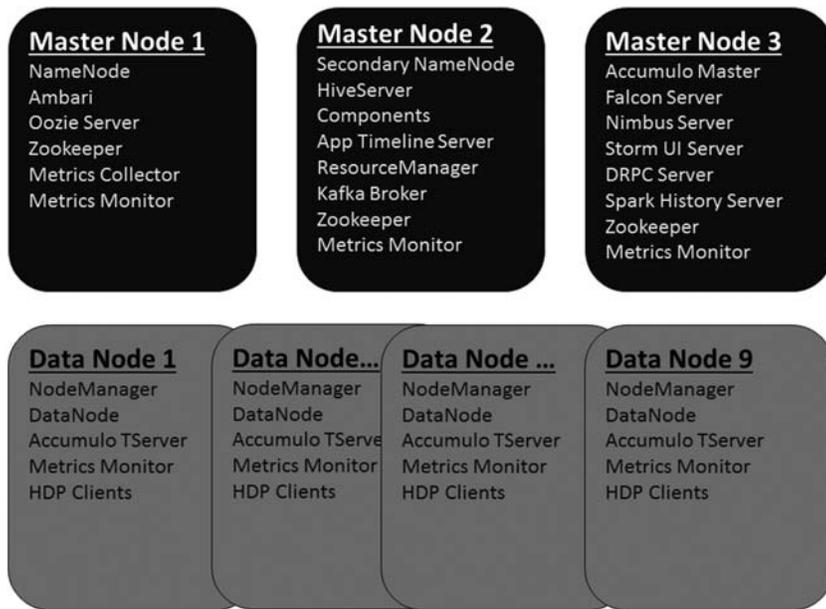
▲ **Figure 1.** U.S. National Data Center (NDC) Hadoop Distributed File System (HDFS) cluster hardware architecture block diagram. Current system configuration consists of three master nodes and nine data nodes, in which each master node hosts different components of the Hortonworks Data Platform. Each data node supplies services for data processing and storage.

Spark-based signal processing applications, which are developed in Java and Scala. Spark provides a fast general-purpose in-memory computing platform for large-scale data processing that runs seamlessly within HDFS. These processes provide the means to segment waveforms, perform band-pass filtering, and conduct limited signal feature extraction on waveform data pulled directly from Accumulo. Our processing results are visualized using the Apache Zeppelin web-based analytics engine. Zeppelin provides a flexible platform for developing web-based dashboard displays for reviewing intermediate signal processing
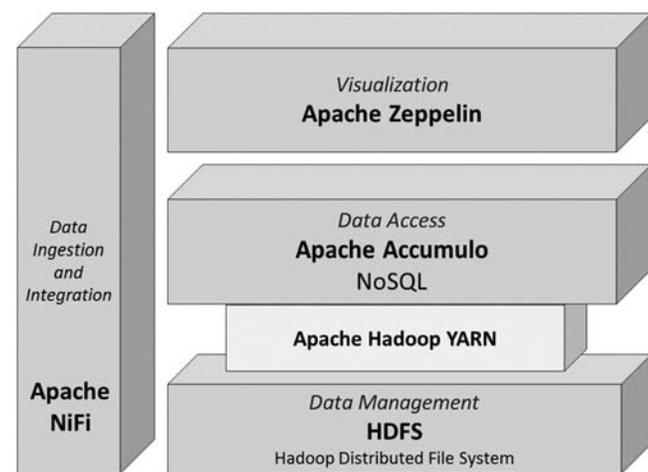
▲ **Figure 2.** U.S. NDC HDFS cluster application level system architecture block diagram.

results and graphically analyzing features extracted from the processed waveform data.

SYSTEM ARCHITECTURE

Figure 1 shows a schematic of the U.S. NDC HDFS cluster hardware configuration and Hadoop software ecosystem distribution. The system comprises 12 Dell PowerEdge R530 servers. Each server possesses two CPUs with six 3.0 GHz Cores, eight 4TB SAS Drives, and 128 GB RAM. The cluster is broken up into three master nodes and nine data nodes, which provide 180 TB of useable storage. The Hadoop processing and storage components reside across the nine nodes, in which individual applications are contained in the master nodes shown in Figure 1. Cluster resource management and performance monitoring is conducted using the Apache Ambari web-based user interface. This hardware and software configuration allows for easy horizontal scaling of the system's processing capacity through the addition of new data nodes.

A diagram describing the data acquisition system's application-level architecture is shown in Figure 2. The system is broken up into four primary components. The data ingestion and integration module funnels SHI data into the data access element using the NiFi component of Hortonworks Data Flow (HDF) tool set. Data access is managed by Accumulo, while HDFS provides the horizontally scalable underlying file system. Apache YARN allocates the resources necessary for reading waveform data from Accumulo, signal processing functions, and data formatting for visualization purposes.

Data Flow
The system uses NiFi to manage the flow of data into the database. NiFi relies on directed graphs to define and manage data routing and transformation. Figure 3 shows the NiFi dataflow used to route and transform our SHI data. Incoming files, containing 10 s of data per SHI sensor, are received in a landing zone approximately once per minute. Each data file contains a collection of seismic data frames that are formatted according to the CD-1.1 protocol, which is the communication standard used for transmitting data from SHI sensors to the IDC and U. S. NDC (Scientific Applications International Corporation [SAIC], 2002). CD-1.1 frames contain 10 s of binary waveform data from multiple channels and associated station metadata. Waveform data are parsed from the CD-1.1 frames and transformed for storage into the Accumulo database.

The NiFi application enables graphical development of scalable data routing and transformation processes. NiFi provides a straightforward means to automate and manage the flow of information between multiple systems. It also provides data provenance and documents the successful completion of
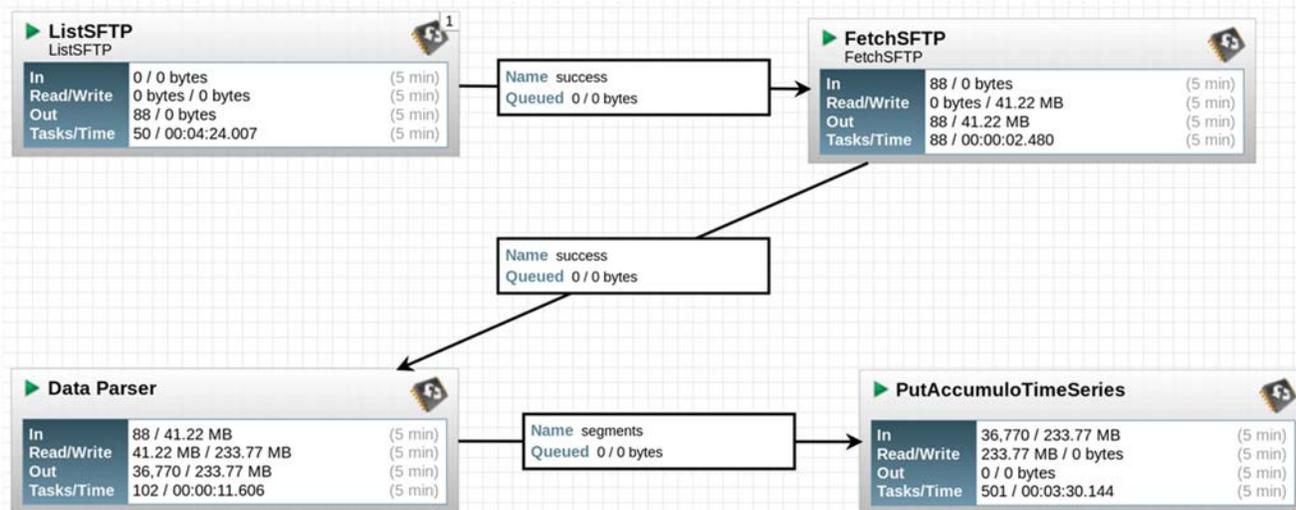
▲ Figure 3. NiFi data parsing and Accumulo time-series data input processor schematic. The color version of this figure is available only in the electronic edition.

processor specific data flow operations. NiFi's basic data unit is the FlowFile, which is a simple representation of each object flowing through the data system. FlowFile processors are used to route and transform the data units. NiFi also provides native solutions to many common data flow situations, which include common processors that interact via communication protocols and collection of processors that comprehend common data formats. In addition, NiFi offers a number of processors that can read and write data in common storage formats. NiFi also supports the creation of custom processors to address additional challenges encountered when connecting disparate systems.

Although the initial identification and retrieval of SHI data files was accomplished with stock NiFi processors, the tasks needed to ingest data from our source network, transform it, and inject it into a NoSQL database required the development of two custom Java processors. By extending the abstract base class provided by NiFi and implementing the appropriate methods, we were able to construct objects that parse, transform, and store our data files. The first processor, *DataParser*, accepts data files as input and transforms them into FlowFile attributes, essentially key–value pairs. The second, *PutAccumuloTimeSeries*, is responsible for receiving the generated FlowFiles from the parser, connecting to the Accumulo store, and writing the data to the appropriate table in the proper format.

Database Infrastructure

Accumulo serves as the data storage mechanism for the HDFS-based data acquisition system. Based on Google's Bigtable (Chang *et al.*, 2006), this Java-based database solution is a highly scalable structured data storage that leverages key–value pairs and operates on top of HDFS. Accumulo supports efficient storage and retrieval of structured data, including queries for ranges (Cordova *et al.*, 2015). It also features automatic

load balancing, partitioning, and data compression, which makes it a good candidate technology for establishing large SHI data repositories.

Accumulo stores and accesses data very differently than other database applications. To better utilize Accumulo's capabilities, we store each waveform sample (i.e., amplitude record stored as raw digitizer counts) and its associated metadata as a single row in the database (Cordova *et al.*, 2015). This method is a significant departure from traditional seismic data storage techniques, in which waveforms are generally stored as a continuous segment with its metadata as a single entity. As a result, our application can build a complete waveform segment in a single step. Using an Accumulo scan, waveform segments are assembled from the individual time-stamped samples extracted from the database as a collection of rows.

The Accumulo data model design can greatly improve or impair data scanning efficiency. Our data model design uses station, channel, and timestamp as the Row ID of the Key (see Fig. 4). This permits rapid scanning of station/channel for a given time range because Accumulo sorts the entries lexicographically by default (Sawyer *et al.*, 2013). Our data were inserted into the column family (CF) and column qualifier (CQ) with the structure of [CF]:[CQ]. This allows the metadata to be contiguous with the appropriate waveform data. The timestamp field is simply the timestamp from the Row ID. The visibility fields were not used in this study, but are available for future use if needed. The value is not used, because the scans that leverage this field are slower than retrieving data directly via the Key.

Figure 5 shows an example output from an Accumulo table scan. This example shows one 10-s CD-1.1 data frame which was extracted from the file retrieved from the previously mentioned landing zone. Recall that our Row ID is defined as the station, channel, and timestamp, which is shown in the first column of the scan output as TX32_BHZ_1472918700000.
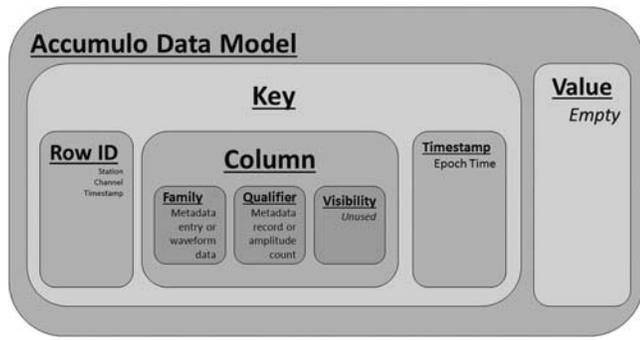
▲ **Figure 4.** Accumulo data model key–value construct, in which the key contains station metadata information, and the value contains a sensor measurement corresponding to a specific time-stamp.

This ensures that our data are sorted by station, channel, and time, which provide direct access to a waveform segment and its associated metadata. The waveform amplitude values for this segment are also associated with the Row ID with the time portion incremented by the sensor sample rate. The first 42 lines of the Accumulo output show a collection of metadata associated with the waveform segment, and the remaining rows are a truncated portion of the actual waveform time-series data for this 10 s frame. The number of lines in the full record is 400, which corresponds to the number of samples in this 10 s frame. However, the number of rows comprising a 10-s waveform segment is a function of the sensor's sample rate. Here, the CF is represented by a "t" and the signal amplitude values are stored as the CQ. The decision to store the signals in the key rather than in the value was deliberate. This design permits direct access to the data via table scan on the key in lieu of retrieving individual waveform samples through key–value lookup.

Data Visualization
A necessary component of waveform processing is the ability to visualize our results. Here, we use Zeppelin to plot intermediate processing results and features extracted from the waveforms. Zeppelin is a web-based notebook that enables interactive data analytics. It provides data ingestion, exploration, and visualization on Hadoop using Spark. Using the Zeppelin editor, Spark jobs, specified in Scala, access data directly from Accumulo and create visualizations that illustrate our immediate and final processing results. This allows the end user to inspect the raw waveforms as well as filtered results directly within a web browser, which can be used later for developing dashboards that monitor a process' progress.

A Scala code snippet illustrating data extraction from Accumulo is shown in Figure 6. Accumulo entries are retrieved using a table scan that spans a particular range of data. In this case, the start and end of the scan is defined for a specific time range for a specific station–channel pair. Scala was used for data retrieval because its syntax facilitates easy iteration over the totality of the result set. The scanner is connected to a specific Accumulo table via the connector object. In this case,

we are passing a default authorizations object, which reflects our currently unimplemented authorization tagging structure, that appears in the output table scan as empty brackets ("[]"). Each row is processed to determine whether it contains metadata or waveform data. If a given row contains metadata, the desired content can be obtained via the constructs implemented in the conditional statement. If waveform data exist, individual amplitude values are extracted as specified in the else condition. In this example, we are simply extracting the sampling period for a specific waveform as our metadata. However, additional pieces of metadata can be obtained as needed. We are also retrieving the waveform sample timestamp (i.e., epoch time) and raw amplitude value pairs.

Figure 7 shows an example of raw and processed data from the TXAR array. The waveforms shown here were extracted from the Accumulo database using the previously described Scala code over a 234-s time interval. The top panel of the plot shows raw waveform data from the TX32 vertical element. The center panel shows the waveform data filtered in the 1–5 Hz frequency band using a two-pole Butterworth filter. The bottom panel shows a trace consisting of peaks extracted from the filtered waveform segment. Peaks were identified by finding amplitude values greater than the scaled difference between the maximum and minimum peaks within the overall waveform segment. The peak trace serves as an example of how simple feature extraction algorithms can be deployed within Spark and results displayed in Zeppelin.

CONCLUSIONS

The first phase of this research and development effort was successful. Our HDFS-based prototype data acquisition and analysis system is currently acquiring data from over 200 SHI sensors. Peak ingest rates are approaching 500,000 entries per second, while preserving constant subsecond access times to any range of entries. Ambari's Accumulo monitoring tools show that the average load produced by the data ingest process is consuming less than 10% of available system resources. This modest use of resources leaves ample processing power for future data analytics applications.

This data acquisition and processing infrastructure offers several advantages over conventional methods. The hardware configuration and software architecture are designed for horizontal scalability using commodity hardware. It provides a means for large-scale storage of waveforms directly within a NoSQL database architecture, as opposed to populating a relational database with pointers to files residing on a UNIX-based file system. Accumulo allows for rapid searches of large waveform archives directly using a properly designed key–value indexing scheme. Direct data access is permitted through the Accumulo *DataInputFormat* class, which eliminates the intermediate creation and storage of data files in HDFS prior to analysis. The Accumulo data format is read directly into Spark, rather than treating the HDFS file output as a temporary Apache Hive table. Zeppelin provides a flexible interface for analyzing and plotting signal processing artifacts using a suite

```
TX32_BHZ_1472918700000 meta:filename:23310908063597819 [] 1472918700000
TX32_BHZ_1472918700000 meta:frame.csf.authenticationKeyIdentifier:1128 [] 1472918700000
TX32_BHZ_1472918700000 meta:frame.csf.authenticationOffset:944 [] 1472918700000
TX32_BHZ_1472918700000 meta:frame.csf.authenticationSize:40 [] 1472918700000
TX32_BHZ_1472918700000 meta:frame.csf.authenticationValue:m ...... [] 1472918700000
TX32_BHZ_1472918700000 meta:frame.csf.cd.authentication:1 [] 1472918700000
TX32_BHZ_1472918700000 meta:frame.csf.cd.calibrationFactor1:0.0063 [] 1472918700000
TX32_BHZ_1472918700000 meta:frame.csf.cd.calibrationFactor2:1.0 [] 1472918700000
TX32_BHZ_1472918700000 meta:frame.csf.cd.channelName:BHZ [] 1472918700000
TX32_BHZ_1472918700000 meta:frame.csf.cd.locationName:US [] 1472918700000
TX32_BHZ_1472918700000 meta:frame.csf.cd.optionFlag:1 [] 1472918700000
TX32_BHZ_1472918700000 meta:frame.csf.cd.sensorType:0 [] 1472918700000
TX32_BHZ_1472918700000 meta:frame.csf.cd.siteName:TX32 [] 1472918700000
TX32_BHZ_1472918700000 meta:frame.csf.cd.transformation:2 [] 1472918700000
TX32_BHZ_1472918700000 meta:frame.csf.cd.uncompressedDataFormat:s4 [] 1472918700000
TX32_BHZ_1472918700000 meta:frame.csf.channelLength:988 [] 1472918700000
TX32_BHZ_1472918700000 meta:frame.csf.channelStatusData:2016247 12:05:09.000 [] 1472918700000
TX32_BHZ_1472918700000 meta:frame.csf.channelStatusSize:32 [] 1472918700000
TX32_BHZ_1472918700000 meta:frame.csf.dataSize:839 [] 1472918700000
TX32_BHZ_1472918700000 meta:frame.csf.samples:400 [] 1472918700000
TX32_BHZ_1472918700000 meta:frame.csf.subframeCount:0 [] 1472918700000
TX32_BHZ_1472918700000 meta:frame.csf.subframeTimeLength:10000 [] 1472918700000
TX32_BHZ_1472918700000 meta:frame.csf.timeStamp:2016247 12:05:00.000 [] 1472918700000
TX32_BHZ_1472918700000 meta:frame.csh.channelStringCount:180 [] 1472918700000
TX32_BHZ_1472918700000 meta:frame.csh.frameTimeLength:10000 [] 1472918700000
TX32_BHZ_1472918700000 meta:frame.csh.nominalTime:2016247 12:05:00.000 [] 1472918700000
TX32_BHZ_1472918700000 meta:frame.csh.numberOfChannels:18 [] 1472918700000
TX32_BHZ_1472918700000 meta:frame.epoch.time:1472904338 [] 1472918700000
TX32_BHZ_1472918700000 meta:frame.header.frameCreator:TXAR [] 1472918700000
TX32_BHZ_1472918700000 meta:frame.header.frameDestination:0 [] 1472918700000
TX32_BHZ_1472918700000 meta:frame.header.frameType:5 [] 1472918700000
TX32_BHZ_1472918700000 meta:frame.header.sequenceNumber:4691931 [] 1472918700000
TX32_BHZ_1472918700000 meta:frame.header.series:0 [] 1472918700000
TX32_BHZ_1472918700000 meta:frame.header.trailerOffset:16808 [] 1472918700000
TX32_BHZ_1472918700000 meta:frame.name:TXAR:0\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00[]1472918700000
TX32_BHZ_1472918700000 meta:frame.size:16864 [] 1472918700000
TX32_BHZ_1472918700000 meta:frame.trailer.authenticationKeyIdentifier:1134 [] 1472918700000
TX32_BHZ_1472918700000 meta:frame.trailer.authenticationSize:40 [] 1472918700000
TX32_BHZ_1472918700000 meta:frame.trailer.authenticationValue: ...... [] 1472918700000
TX32_BHZ_1472918700000 meta:frame.trailer.commVerification:4521311049294015679 [] 1472918700000
TX32_BHZ_1472918700000 meta:path:./ [] 1472918700000
TX32_BHZ_1472918700000 meta:uuid:d870f936-87af-4914-a77b-d6dcb481df97 [] 1472918700000
TX32_BHZ_1472918700000 t:-36286 [] 1472918700000
TX32_BHZ_1472918700025 t:-45761 [] 1472918700025
TX32_BHZ_1472918700050 t:-48020 [] 1472918700050
TX32_BHZ_1472918700075 t:-33913 [] 1472918700075
TX32_BHZ_1472918700100 t:-16758 [] 1472918700100
TX32_BHZ_1472918700125 t:-10271 [] 1472918700125
TX32_BHZ_1472918700150 t:-14003 [] 1472918700150
TX32_BHZ_1472918700175 t:-12985 [] 1472918700175
TX32_BHZ_1472918700200 t:-756 [] 1472918700200
TX32_BHZ_1472918700225 t:7663 [] 1472918700225
```

▲ **Figure 5.** Example Accumulo table scan showing archived TX32 data. The last 10 rows of this example represent individual samples of an archived waveform, in which the number in the second field is the waveform amplitude record, the third field is the unpopulated data tag, and the last field is the sample timestamp shown as epoch time.

```
val startText: Text = new Text("TX32_BHZ_1472918699900")
val endText: Text =   new Text("TX32_BHZ_1472918934000")
val zkiConfig: ClientConfiguration  = new ClientConfiguration().withInstance("hdp-accumulo-instance").withZkHosts("< accumulo master server host>");
val instance: Instance = new ZooKeeperInstance( zkiConfig )
val connector: Connector = instance.getConnector( "<accumulo user>", new PasswordToken("<accumulo password>"))
val auths = new Authorizations()
val scanner = connector.createScanner("<accumulo table name>", auths)
val range = new Range( startText, endText )
scanner.setRange( range )

for(entry <- scanner.asScala ) {
    val row: String = entry.getKey().getRow().toString()
    if( entry.getKey().getColumnFamily().toString() contains "meta" ) {
        if( entry.getKey().getColumnFamily().toString() contains "samples" ) {
            val samples = entry.getKey().getColumnQualifier().toString()
            val rate: Double = 10 / samples.toDouble
        }
    }else {
        val amp: String = entry.getKey().getColumnQualifier().toString()
        val sta_chan_time: List[String] = row.split("_").map(_.trim).toList
        val etime: String = sta_chan_time(2)
    }
}
```

▲ **Figure 6.** Scala code snippet executed within Zeppelin notebook for extracting time-series data from Accumulo database.

of conventional file system scripting tools and HFDS-based applications.

In our opinion, the barrier of entry for enabling HDFS for acquiring and storage of SHI data is fairly low. The process of installing and configuring our hardware cluster, developing the data acquisition software, validating its functionality, and visualizing the results occurred over a few man weeks. However, smaller proof of concept systems could be established on a shoestring budget, using even a single node, which can be scaled later as storage and processing requirements dictate.
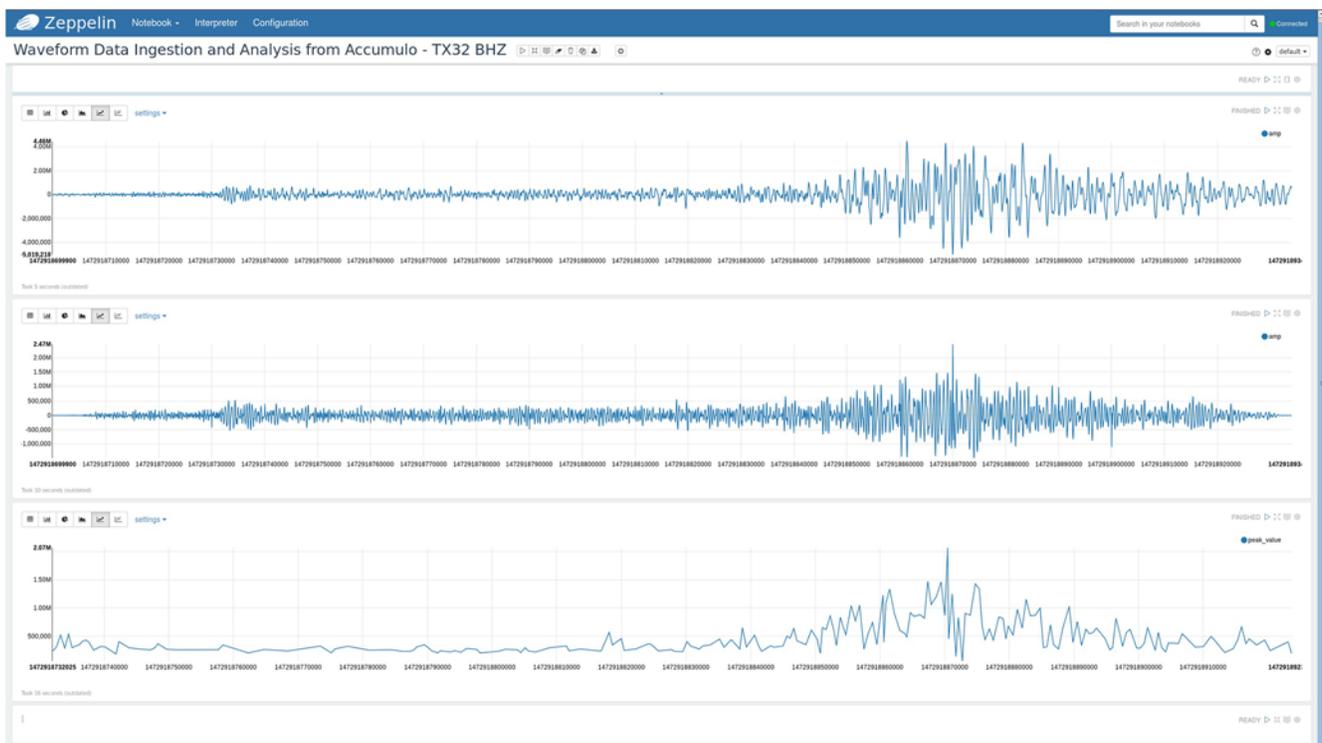
▲ **Figure 7.** Zeppelin screenshot showing 234 s of raw and filtered seismic data from the TXAR array located in Lajitas, Texas. (Top) Raw data from the TX32 vertical element, (middle) the same data filtered in the 1–5 Hz passband, and (bottom) a positive peak trace that was extracted from the filtered waveform. The color version of this figure is available only in the electronic edition.

Future work will explore refactoring of the Accumulo Row ID to use a "/" as a delimiter to enable compatibility with time-series tools such as Timely and Grafana. These technologies provide additional methods for developing more advanced visualization, analysis dashboards, and performance monitoring tools that are accessible through a web browser. In addition, we will also focus on feeding data stored in Accumulo directly into a collection of custom machine-learning algorithms designed to quantify waveform similarity. Real-time availability of waveform similarity metrics will provide the means to automatically recognize events that have been observed in the past (e.g., mining activity).

DATA AND RESOURCES

Seismic data from the TX32 can be obtained from the Incorporated Research Institutions for Seismology (IRIS) Data Management Center at http://ds.iris.edu/mda/IM/TX32 (last accessed June 2017). Apache foundation software is openly available and can be obtained from www.apache.org (last accessed March 2017). Hortonworks Data Platform (HDP) and Hortonworks Data Flow (HDF) software suites are freely available and can be obtained from www.hortonworks.com (last accessed March 2017).

ACKNOWLEDGMENTS

REFERENCES

Addair, T. G., D. A. Dodge, W. R. Walter, and S. D. Ruppert (2014). Large-scale seismic signal analysis with Hadoop, *Comput. Geosci.* **66,** 145–154, doi: 10.1016/j.cageo.2014.01.014.

Chang, F., J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber (2006). Bigtable: A distributed storage system for structured data, *OSDI'06: Seventh Symposium on Operating System Design and Implementation*, Seattle, Washington, November 2006

Cordova, A., B. Rinaldi, and M. Wall (2015). *Accumulo: Application Development, Table Design, and Best Practices*, O'Reilly Media, Inc., Sebastopol, California, ISBN-13: 978-1-4493-7418-1.

Dodge, D. A., and W. R. Walter (2015). Initial global seismic cross-correlation results: Implications for empirical signal detectors, *Bull. Seismol. Soc. Am.* **105,** 240–256, doi: 10.1785/0120140166.

Magana-Zook, S., J. M. Gaylord, D. R. Knapp, D. A. Dodge, and S. D. Ruppert (2016). Large-scale seismic waveform quality metric calculation using Hadoop, *Comput. Geosci.* **94,** 18–30, doi: 10.1016/j.cageo.2016.05.012.

Sawyer, S. M., B. D. O'gwynn, A. Tran, and T. Yu (2013). Understanding query performance in Accumulo, *High Performance Extreme Computing Conference (HPEC), IEEE*, 1–6.

Scientific Applications International Corporation (SAIC) (2002). *Formats and Protocols for Continuous Data CD-1.1*, as part of the International Data Centre Documentation, available at www.ctbto.org/fileadmin/user_upload/procurement/2011/Attachment_3_to_the_TOR.pdf (last accessed March 2017).

William N. Junek
Charles A. Houchin
Joseph A. III Wehlen
Air Force Technical Applications Center
10989 South Patrick Drive
Patrick Air Force Base
Florida 32925-3002 U.S.A.
william.junek@us.af.mil

John E. II Highcock
Marcus Waineo
Hortonworks Inc.
5470 Great America Parkway
Santa Clara, California 95054 U.S.A.

Published Online 27 September 2017