Seismic Data Collection with Shakebox and Analysis Using MapReduce

Bin Tang, Jianchao Han, Mohsen Beheshti, Garrett Poppe, Liv Nguekap, and Rashid Siddiqui
Department of Computer Science
California State University Dominguez Hills
Carson, CA 90747
{btang,jhan,mbeheshti}@csudh.edu, {gpoppe1, lnguekap1, rsiddiqi1}@toromail.csudh.edu

Abstract— In this paper we study a seismic sensing platform using Shakebox, a low-noise and low-power 24-bit wireless accelerometer sensor. The advances of wireless sensor offer the potential to monitor earthquake in California at unprecedented spatial and temporal scales. We are exploring the possibility of incorporating Shakebox into California Seismic Network (CSN), a new earthquake monitoring system based on a dense array of low-cost acceleration seismic sensors. Compared to the Phidget/Sheevaplug sensors currently used in CSN, the Shakebox sensors have several advantages. However, Shakebox sensor collects 4K Bytes of seismic data per second, giving around 0.4G Bytes of data in a single day. Therefore how to process such large amount of seismic data becomes a new challenge. We adopt Hadoop/MapReduce, a popular software framework for processing vast amounts of data in-parallel on large clusters of commodity hardware. In this research, the testbed-generated seismic data generation will be reported, the map and reduce function design will be presented, the application of MapReduce on the testbed-generated data will be illustrated, and the result will be analyzed.

Keywords—*seismic data; Shakebox; Hadoop MapReduce*

I. INTRODUCTION

Sitting on the tectonic boundary between the Pacific and the North American Plate, California is the state with second-most earthquakes in the United States. Large earthquakes are inevitable in California -- according to the 2007 Working Group on California Earthquake Probabilities (WGCEP 2007), the probability of a magnitude 6.7 or larger earthquake striking the greater Los Angeles area before 2038 is 67%. The recent 6.0-magnitude South Napa Earthquake, the strongest earthquake in 20 years, caused over $400 million in damage and served as a wakeup call of how serious the situation could be.

Currently California Integrated Seismic Network (CISN, http://www.cisn.org/) is the statewide system in California for earthquake monitoring, emergency response, and loss mitigation. It has deployed around 3000 seismic stations in California and western Nevada. A seismic station includes a sensor (short-period, broadband, and strong-motion) to record the ground motion, a computer to save the data, a GPS for accurate timing and location, telemetry or radio equipment to send the data back to a processing center and a power source to run the station. While these stations provide high fidelity and reliable data themselves, California still lags in the number of stations needed to provide high quality of earthquake information throughout the state. Besides, being bulky and power-hungry, these stations have been costly to install and maintain, preventing them from large-scale deployment. As a result, these sparsely distributed stations only provide limited coverage and coarse-grain monitoring for earthquake.

The California Seismic Network (CSN, www.csn.caltech.edu) is a transformative approach to earthquake detection, science, and outreach. The CSN is a new earthquake monitoring system based on a dense array of low-cost acceleration seismic sensors. By producing block-by-block measurements of strong shaking during an earthquake, which are called shake maps, it can help first responders, fire fighters, rescue workers to pin down accurately the damage area in block by block level. The technical idea of CSN is that a small seismometer is hosted in each residential home or office, such that when earthquake takes place, the very granular measurement can be immediately transmitted to a centralized server (a Google cloud server) via Internet for real-time analysis. Preliminary results have shown that this system is very effective to measure earthquake damage and alert the public [1][2][3][4].

This paper explores four aspects that further contribute to CSN:

1. The functioning of CSN assumes the existence of the communication infrastructure. However, when the Internet is torn down during earthquake, the seismic sensors are disconnected from Google cloud via the Internet. How to preserve the large amount of data it generates during earthquake becomes a new challenge.

2. Even if Internet connection still exists, the constant transmission of the readings of hundreds of seismic sensor back to the Google cloud will inevitably overburden the communication infrastructure. Therefore, the inner data processing among sensor nodes wirelessly become more relevant.

3. The current seismic sensor node does not have wireless

communication capabilities. Each seismic sensor used in CSN is operated by a small Ubuntu-based computer called SheevaPlug [7]. There is no well-defined solution to equip the current version of the SheevaPlug with wireless capability.

4. Finally, the SheevaPlug is a relatively lower-end seismic sensor, with very coarse sampling resolution.

Considering all above, we are exploring the possibility to integrate a high-precision, GPS-based, wireless, large-storage seismic sensor, called Shakebox, into CSN. In contrast to low-cost sensors used in CSN, the Shakeboxes are equipped with the most accurate strong-motion accelerometer defined by the ANSS standards. Therefore, Shakeboxes can possibly produce more precise measurements than the low-cost sensors of CSN. However, around 0.4 GB of data is generated by a Shakebox in a single day. How to process such large amount of seismic data becomes a new challenge.

In the past decade, the MapReduce programming model [10] has emerged as a popular framework for large data set analysis. The key idea of MapReduce is to divide the data into chunks, which are processed in parallel. Several open source MapReduce frameworks have been developed in the last years. In particular, Hadoop [11], the most prevalent implementation of MapReduce, has been extensively used by companies and research communities on a very large scale. In this paper, we adopt Hadoop and MapReduce for the data process and show and analyze our experimental results. Specifically, we design Map and Reduce functions that suit for the application of seismic big data.

The rest of the paper is organized as follows. Section II discusses the related work. In Section III, we introduce Shakebox seismic sensing platform. Section IV presents MapReduce and Hadoop Distributed File System. In Section V, we design MapReduce functions for Shakebox seismic data analysis. We show our experiment on data collection and data analysis in Section VI. We conclude our paper in Section VII and discuss some future work.

II. RELATED WORK

With the development of the micro-electro-mechanical systems (MEMS) low-cost accelerometers and the proliferation of mobile devices such as laptops and smartphones, several seismic monitoring networks that utilize low-cost and USB-based accelerometers are designed and deployed. We are aware of two projects in this line: The Community Seismic Network (CSN) [1][2][3][4] and the Quake-Catcher Network (QCN) [5,6].

The CSN envisions city-wide networks of community-owned sensor devices that perform large-scale seismic sensing. It is collaboration among geophysicists, civil engineers and computer scientists to develop the sensor technologies, scalable infrastructure, and algorithmic tools needed to reliably perform large-scale seismic sensing. In contrast to traditional seismic networks that contain a small number of highly accurate sensors, the CSN project focuses on large numbers of inexpensive, community-held sensors, such as those in personally owned devices like smart phones.

The QCN is a distributed computing seismic network that links internal (no cost, built-in) or external (low-cost, USB-based) accelerometers connected to any participating computer for earthquake research. It is based on a distributed computing platform called Berkeley Open Infrastructure for Network Computing (BOINC) [15]. The objective of QCN is to dramatically increase the number of seismic observations by exploiting recent advances in sensing technologies and cyber infrastructure capabilities for automated warning and alert for natural disasters.

Shakeboxes have been used in ShakeNet [8], a portable wireless sensor network for instrumenting large civil structures such as buildings and bridges. The ShakeNet software subsystem is built upon Tenet [14], a programmable wireless sensing software architecture designed for tiered sensor networks.

In contrast to all above Shakebox-related research, we are approaching the Shakeboxes from the perspective of big-data processing, and design Map and Reduce functions to process the seismic data generated by Shakeboxes.

There are a few researches in recent years that focus on big sensor data. Lee et al. [9] designed a platform that enables sensor data to be taken from collection, through use in models to produce useful data products. They propose a response through a sensor data platform “Concinnity”, which can take sensor data from collection to final product via a data repository and workflow system. They summarize the key features of their approach and explore how it enables value to be derived from sensor data efficiently. Liu et al. [12] propose an integrated method to address the heterogeneity issue in modeling big time series sensor data. They present both linear and nonlinear feature extraction techniques, as well as a procedure to determine the right extraction method for individual time series. Guo et al. [13] observe that model-based sensor data approximation reduces the amount of data for query processing. They propose an innovative index for modeled segments in key-value stores, called KVI-index. They show their approach outperforms in query response time and index updating efficiency both Hadoop-based parallel processing of the raw sensor data and multiple alternative indexing approaches of model-view data.

In contrast to above big sensor data researches, most of which focus on mathematical modeling of big sensor data, we are particularly interested in the whole process of real seismic sensor data collection, analysis, and visualization.

III. SHAKEBOX SEISMIC SENSING PLATFORM

Shakexbox is a wireless sensor node equipped with a low-noise and low-power 24-bit triaxial accelerometer. The use of low-noise and low-power 24-bit accelerometer puts ShakeBox near the US Geological Survey's 'Class A' device specifications for earthquake measuring instruments. The system comes preloaded with sensing software as well as deployment tools that enable rapid deployment. Small form factor and portable design makes Shakeboxes easy and fast to deploy. Shakeboxes have been used in structural health applications [8], since aforesaid small form factor and portable design help in capturing the structural health of the building, bridge, dam or tunnel expeditiously when compared to wired monitoring methods. Shakeboxes have their own power source, making them independent of the infrastructure. This is vital for remote locations or in disaster struck areas.

(a)

(b)

Figure 1. a) Shakebox with weatherproof enclosure with 6-inch ruler shown for scale. b) Placement of detailed modules with 6-inch ruler shown for scale. Source: http://nsl.cs.usc.edu/Projects/ShakeNet

The Shakeboxes were manufactured by Refraction Technologies (RefTek, http://www.reftek.com/). Figure 1 is a modular design paradigm for the ShakeBox. Figure 1 (a) shows several visible modules: Power, GPS, Radio, and Communication. Figure 1 (b) shows the CPU, Power and A/D modules inside the Shakebox. These modules are housed in a custom-made weatherproof casing.

The CPU module contains the system processor board (a Crossbow iMote2 mote) and the RT617 CPU carrier board and controls all system operations. The boards are housed in an electro magnetic shield to reduce external effects on the analog module.

The iMote2 mote controls the communication to other three modules. IMote2 is an advanced sensor network platform and consists of a Marvel PXA271A ARM CPU, which is a 32bit microcontroller, and a CC2420 radio, an 802.15.4 compliant 2.4GHz wireless communication radio with up to 256Kbps bit data rate. Dynamic scaling of core frequency of the PAX271 microcontroller from 13MHz to 208MHz provides a varied range of options for balancing processing power with energy usage.

The Power module provides the power requirements of the different components and consists of RT618 FPGA board and RT620 power board. RT618 provides communication with CPU module, a clock, control of the voltage monitor A/D converter, control of analog power supplies and board ID EEProms. RT620 provides an input power controller, switching supplies at different voltage levels, a 16-bit A/D monitor for supply voltages and input currents, and a board ID EEProms.

The sensor module consists of three Colibyrs SiFlex 1500 accelerometers, which are interfaced to the RT614 board in the A/D module. The SiFlex1500 operates from a bipolar power supply voltage that can range from ± 6V to ± 15V with a typical current consumption of 12mA at ± 6V. The linear full acceleration range is ± 3g with a corresponding sensitivity of 1.2V/g. The sampling rate of the sensor module is 10, 100, 200, or 1000 samples per second.

The weatherproof casing houses all the modules. Each module is electronically shielded to protect against electromagnetic disturbance. The lead acid battery used in the Shakebox is placed in a separate sealed compartment to isolate it from the electronics in case of battery leakage. The box provides serial connectors, connector for GPS, LEDs for display and feedback and antenna connector for high gain external antenna used by iMote2's radio. It has three screws and a spirit level for leveling. The prototype box in Fig. 1 is made up resin plastic but the production pieces will be metallic aluminum.

Finally, the Shakebox is equipped with one or two Compact Flash Type I or Type II storage media (disks). CF flash storage is available up to 16 GB capacity. For example, 4 GB is enough storage to hold more than 100 days of three channel, 100 sample per second data recorded with compression. Files are written in FAT32 format allowing high capacity disks to be used.

IV. MAPREDUCE AND HADOOP DISTRIBUTED FILE SYSTEM (HDFS)

MapReduce is a distributed processing framework that enables big data processing. The Hadoop distributed file system (HDFS) is a distributed, scalable, and portable file-system written in Java for the Hadoop framework. Below we present both and show how we adopt them in our seismic data analysis.

A. MapReudce

MapReduce [10] has two main components, a mapper and a reducer. A mapper works on each individual input record to generate intermediate results, which are grouped together based on some key and passed on to the reducers. A reducer works on the group of intermediate results associated with the same key and generates the final result using a result aggregation function. The processing units of the MapReduce framework are key-value pairs.

Mapper: The Mapper maps input of key/value pairs to an intermediate set of key/value pairs. A single map task transforms an input record to an intermediate record. Intermediate records can be of a different type than input records. Raw numerical data used as an input can be mapped to an array or records as the intermediate data. Based on the input map configuration and task specifications, mapping can result in many output pairs or zero output pairs. Hadoop MapReduce framework will create one map task for every input task specified by the input format for the job. A job is a method in which MapReduce assigns a task to be processed in a manner specified by the user created configuration to evaluate key/value pairs from input to intermediate records.

Output pairs are collected during job execution. During job execution reports can be generated to display application level status messages, update counters, or indicate jobs are still running. All intermediate values are given an output key and grouped by the framework based on the key. A user can specify the method of grouping the intermediate values. Among the configurations a user can specify the ability to compress the intermediate values for storage. Intermediate values are passed to the Reducer for final output. In order to cut down on the amount of data transferred to the reducer, local aggregation of intermediate values can be performed.

Reducer: The Reducer performs a series of tasks, configured by the user, to reduce the input set of intermediate values to a smaller set of values that have a common key. During the reduction process the Reducer traverses three different phases: shuffle, sort, and reduce.

Shuffle: During this phase the Reducer identifies all the mappers that have data to be exported to the reducer. Upon identification of the partitions where the data is stored, Reducer fetches the sorted output from the partitions (via HTTP if stored on external nodes).

Sort: The framework groups the inputs by keys. The map output of intermediate values may contain output pairs with the same key. The duplicates are merged at the same time as data is being fetched. Sorting configurations can be set to include rules that determine grouping of intermediate keys as well as assigning different comparison functions. Sorting functions can specify secondary sorts to more efficiently group data before the next phase.

Reduce: During a round of reduction, the Reducer compares key/lists by group and reduces grouped inputs. A number of reducing rounds will need to be completed before the entire data set is reduced. The output from the reducer is unsorted and written to the file system via a path specified by the user. The number of reduces and their associated scaling factor can be modified through user configuration settings. During the reduce phase a user can specify a reporter to report progress and application-level status messages, and update counters, or indicate overall status.

During the MapReduce process, multiple nodes are processing jobs. Each job can be assigned a timeout to end the task in case of process or node failure. The job can be reassigned to another process after timeout. The process of assigning multiple jobs over multiple nodes to be processed can be described as batch processing. Batch processing in a MapReduce framework allows a user to specify how jobs are executed and how results are stored and reduced. Each Mapper/Reducer task executes as a child process in a separate JVM. A user can specify the maximum virtual memory of a child-task.

B. Hadoop Distributed File System (HDFS)

Hadoop Distributed File System (HDFS) is a Java-based file system that provides scalable and reliable data storage that is designed to span large clusters of commodity servers. HDFS splits files into large blocks (default 64MB or 128MB) and distributes the blocks amongst the nodes in the cluster.

Each Java Virtual Machine (JVM) is given its own file system on the local machine. The file system includes: output directory, work directory, temp directory, and an xml file that specifies task localized job configuration. The task file system is a subsystem of the job file system. The job file system stores files needed for each task. The work directory stores files that will need to be shared by multiple tasks for a specific job. A jar directory contains the user specified program that controls the tasks. The job file system also contains an xml file that outlines the configuration for the localized job.

```
public void map(LongWritable key, Text value,  Context context)
               throws IOException, InterruptedException {
          try{
                 String myWord = "Quake " + inc + ": ";
                 String line = value.toString();
                 StringTokenizer tokenizer = new StringTokenizer(line, "\t");
                 word.set(myWord);
                 tokenizer.nextToken();
                 String x_value = tokenizer.nextToken();
                 String y_value = tokenizer.nextToken();
                 String z_value = tokenizer.nextToken();

              //Checking the X column
           if(!first_time){
                  if(count < xRange_array.length){
                         // X-column
                         new_x = Double.parseDouble(x_value);
                         xRange_array[index] = Math.abs(new_x - old_x);
                         xSum += xRange_array[index];
                         old_x = new_x;

                         // repeat for Y-column
                         // repeat for Z-column

                         index++;
                  }
                  else{
                         if(index >= xRange_array.length){
                                index = 0;
                         }
                         // X-column
                         xSum -= xRange_array[index];
                         new_x = Double.parseDouble(x_value);
                         xRange_array[index] = Math.abs(new_x - old_x);
                         xSum += xRange_array[index];
                         old_x = new_x;

                         // repeat for Y-column
                         // repeat for Z-column

                         index++;
                  }

                  if(((xSum / xRange_array.length) >= xThreshold) &&
                              ((ySum / yRange_array.length) >= yThreshold) &&
                              ((zSum / zRange_array.length) >= zThreshold)){
                         context.write(word, one);
                         quake_happening = true;
                  }
                  else if(quake_happening = true){
                         inc++;
                         quake_happening = false;
                  }
                  count++;
           }else{
                 old_x = Double.parseDouble(x_value);
                 old_y = Double.parseDouble(y_value);
                 old_z = Double.parseDouble(z_value);
                 first_time = false;
           }
           }catch(NoSuchElementException nsee ){

           }catch(NumberFormatException nfe){}

       }
    }
```

Figure 2. Map Function.

```
public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {
        private int inc = 1;
        private String quake = "Quake ";
       public void reduce(Text key, Iterable<IntWritable> values, Context context)
         throws IOException, InterruptedException {
           int sum = 0;
           for (IntWritable val : values) {
               sum += val.get();
           }
           if(sum >= 20 ){

                  key.set(quake + inc + ":");
                  context.write(key, new IntWritable(sum));
                  inc++;
           }
       }
    }
```

Figure 3. Reduce Function.

The user submits jobs through a client interface. The interface also allows the user to track progress, access reports and logs, and retrieve cluster status information. A job submission should include: checking the input and output specifications of the job, computing key/values for the job, set up requisite accounting information for the distributed cache of the job, copying the job's jar and configuration files to the file system, and submitting the job and monitoring it's status.

Job control can be used to chain MapReduce jobs to accomplish complex tasks that cannot be done by a single job. This can be accomplished by using the output of one job as the input of another job. If real time data needs to be processed this method can be used to process and buffer data.

HDFS is a cluster of nodes that is used as a framework for storing data. A NameNode manages the file system meta-data. The DataNodes store the actual data. The client interface contacts the NameNode for file meta-data or file modifications and performs the actual file I/O directly with the DataNodes.

The NameNode and DataNodes have built in web servers to easily interact with each other. The NameNode stores modifications to the file system as a log appended to the native file system. During NameNode start up, it reads the HDFS state from an image file and applies edits from the log file. The new HDFS state is saved to the image file and starts normal operation.

HDFS data is not always uniformly distributed across nodes. To help eliminate data loss failure to one of the nodes, the system replicates blocks of data across all of the nodes. This process also helps to balance the distribution of data. Placement or grouping of blocks of data can be done to increase processing or increase the efficiency of MapReduce. The NameNode can control clusters of thousands of DataNodes each with its own functionality to perform specific jobs.

V. MAPREDUCE FUNCTIONS FOR SHAKEBOX DATA ANALYSIS

The earthquake data is produced through experiment with the Shakebox. REF TEK 130S is used as input for the MapReduce program. The data in the file is formatted into four tab-delimited columns, each containing data of the type "double".

Figure 2 and Figure 3 show the Map and Reduce functions that we designed, respectively. When the program is launched, the map function reads the input file line by line. It then maps each data item from the line based on the algorithm provided. In our case, the algorithm takes, for each column, the first value and subtracts it from the next. The absolute value of the result is saved to an array of doubles. As the array is progressively filled up, the data contained in the array is summed and then divided by the number of elements in the array to find the average value.

If the result of this operation is less than a threshold value we hardcoded, then the program understands that nothing is happening. If the result of this operation is greater than the threshold value, then the program understands that an event is happening. The event in this case is the first wave of an earthquake. Therefore, the program generates an output pair such as ("Quake 1:", 1) to indicate that 1 tremor has been recorded for the first wave of the earthquake. If a second tremor occurs for the first wave, then another key/value pair ("Quake 1:", 1) is recorded. And so on.

The array represents a range over which the average of the differences between consecutive values has to be greater than the threshold value in order for us to consider that an earthquake is happening. We chose a threshold of 0.5 and an array of 100 elements for our range. When the end of the array is reached, the oldest value, which is located at index 0, is removed, and the next value from the input file is loaded at that index. Therefore, as the mapping programs goes through the file, we always have 100 values that are being considered to determine whether or not an earthquake is happening.

If the average of the difference of those 100 elements falls below the threshold value after being higher than that for a while, then we consider that the first wave of an earthquake that was happening has stopped. At this moment, we start looking for the next wave. Repeat this process until we reach the end of the file.

At this point, the data mapped into the context of the MapReduce program is shuffled and sorted. During this process, all the data belonging to "Quake 1", which indicates the first wave of the earthquake, is put into a key/value pair. The word "Quake 1" is used as the key, and every other item that was coupled with the key "Quake 1:" in the output of the map function is put into a list. That list is identified as the value. The same process is repeated for "Quake 2:", "Quake 3:", and so on, if they exist.

Then, each key/value pair is sent to the reduce function. This function identifies each key and iterates over the list of value to sum them up. The total for each key is saved to the output file. And by repeating this process for each key and its associated values, we obtain a file that shows how many waves of earthquake occurred, and for each wave the total number of tremors.

VI. EXPERIMENT

A. *Artificial Seismic Data Generation from Shakeboxes*

We connected the Shakebox to a PC and collected data for 10 minutes. The sample rate of the Shakebox is set as 200 samples per second (the maximum sampling rate is 1000 samples per second). During data collection, we jumped on the ground several times to create a few artificial earthquakes (otherwise, the data always stays the same, which does not help in our MapReduce analysis later on). Figure 4 is the data collection GUI, showing all the steps taken to collect data from Shakeboxes:

1) Connect data cable to com port on shakebox and usb port on pc.
2) Turn on shakebox with magnetic switch.
3) Click `imoteconsoleshuai.Exe` and it will show `usbxxxx` under `selected imote 2`, which means shakebox is detected and ready.
4) Run `./usbloaderhost.Exe –p shakebox.Bin.Out` in cygwin command line and it will start programming the shakebox.
5) On `imoteconsole`, select the `imote2 device (usbxxxx)` --> click `view buffer in window` --> click `connect` --> press `enter` in BluSH prompt.
6) Type `ls` to see all commands supported.
7) Type `startcollection 200 120000` to collect raw ADC data at 200hz for 10min, data will be saved in file `ad-data-raw`.
8) Use `stopcollection` to stop data collection.

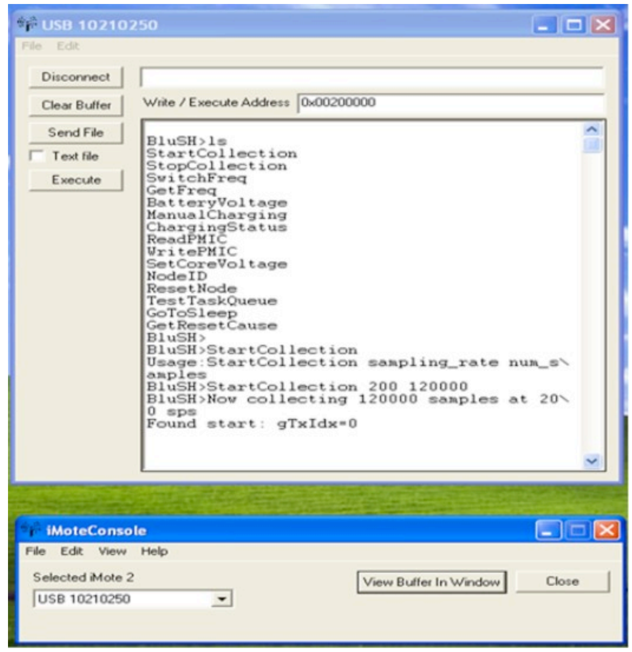


Figure 4. Data Collection GUI.

The raw data stored in `ad-data-raw` looks like below.

```
00000ecb  4ca5d00e  22d4c500  05c4d204  0509bb08
00000ecb  4de5d00e  22d5e200  05c4c404  0509dc08
00000ecb  4f25d00e  22d50c00  05c51004  0509dc08
00000ecb  5065d00e  22d4e700  05c55f04  050a0008
```

Each line of the raw data contains five 32-bit words (in hexadecimal) in following order: date, time, acceleration in channel 1 (X axis), acceleration in channel 2 (Y axis), acceleration in channel 3 (Z axis). Since we collected data for 10 minutes, with 200 data collected per second, it therefore collected 10*60*200=120,000 number of data (that is, there are 20,000 lines in the `ad-data-raw` text file). The total size of the data is 480K Bytes. Within one day, it collects 69.12 Mbyte of data. With maximum sampling rate of 1000, it could generate 0.4G Bytes of data in a single day.

Above raw data can be converted to seismic data as follows. In each line, the first data is the combined data and time, the second, third, and fourth items are the acceleration data in X, Y, and Z axis, respecively.

```
946.3065908203125          3.672951192799
0.608369062626 0.531226649424

946.3115908203125          3.673409770054
0.608346534372 0.531279744576

946.3165908203125          3.6730654348520004
0.6084688306079999 0.531279744576

946.3215908203125          3.6730059002610003
0.608595954327 0.5313376665599999
```

```
946.3265908203125         3.6731716316900003
0.608576644395 0.5314132869279999
```

The unit of above seismic data is g, the standard value of gravitational acceleration at sea level on Earth.[1]

B. *Data Analysis Using MapReduce*

In our data analysis, we have adopted 4 machines, each machine has Xeon processors, 2 TB disc storage, and 12 GB RAM. As raw data is received from the sensor nodes, it is saved to the HDFS. The data can be stored in the temp directory until processed. A job is created to process the data using the MapReduce functions. The output data is saved in a directory intended to be the input of a java visualization chart program. The program uses the data set start and finish points to output a graph depicting the changes in movement for the X,Y, and Z planes. The graphs can be saved for future analysis or analyzed in real time. The final graph file system will not include idle time data, which in turn will reduce the state space and search space for future analysis. Future programs to implement in the MapReduce environment can include mapping the comparison of the three graphs and reducing the data to anomalous events for future research.

Figure 5 shows that the initial mapping tasks are created as jobs and assigned to individual nodes for processing. The NameNode keeps track of each job and follows the user configuration of the mapping function to assign file location and control settings.

```
WARNING: Use GenericOptionsParser for parsing the arguments. Applications should implement Tool for the same.
Nov 21, 2014 3:35:16 PM org.apache.hadoop.mapred.JobClient copyAndConfigureFiles
WARNING: No job jar file set.  User classes may not be found. See JobConf(Class) or JobConf#setJar(String).
Nov 21, 2014 3:35:17 PM org.apache.hadoop.mapreduce.lib.input.FileInputFormat listStatus
INFO: Total input paths to process : 1
Nov 21, 2014 3:35:17 PM org.apache.hadoop.io.compress.snappy.LoadSnappy <clinit>
WARNING: Snappy native library not loaded
Nov 21, 2014 3:35:18 PM org.apache.hadoop.mapred.JobClient monitorAndPrintJob
INFO: Running job: job_local729372904_0001
Nov 21, 2014 3:35:18 PM org.apache.hadoop.mapred.LocalJobRunner$Job run
INFO: Waiting for map tasks
Nov 21, 2014 3:35:18 PM org.apache.hadoop.mapred.LocalJobRunner$Job$MapTaskRunnable run
INFO: Starting task: attempt_local729372904_0001_m_000000_0
Nov 21, 2014 3:35:18 PM org.apache.hadoop.util.ProcessTree isSetsidSupported
INFO: setsid exited with exit code 0
Nov 21, 2014 3:35:19 PM org.apache.hadoop.mapred.Task initialize
INFO:  Using ResourceCalculatorPlugin : org.apache.hadoop.util.LinuxResourceCalculatorPlugin@cbf30e
Nov 21, 2014 3:35:19 PM org.apache.hadoop.mapred.MapTask runNewMapper
INFO: Processing split: file:/home/os/workspace/lunwordcount/my_input/Quake_result -x.txt:0+4138767
Nov 21, 2014 3:35:19 PM org.apache.hadoop.mapred.MapTask$MapOutputBuffer <init>
INFO: io.sort.mb = 100
Nov 21, 2014 3:35:19 PM org.apache.hadoop.mapred.JobClient monitorAndPrintJob
INFO:  map 0% reduce 0%
```

Figure 5. Initial Mapping Output

After all mapping tasks have been executed as jobs, the status of those results is displayed, as shown in Figure 6. It also specifies some configuration settings as well as some logged statistics including resource calculations.

```
INFO: Finished spill 0
Nov 21, 2014 3:47:24 PM org.apache.hadoop.mapred.Task done
INFO: Task:attempt_local891637895_0001_m_000000_0 is done. And is in the process of commiting
Nov 21, 2014 3:47:25 PM org.apache.hadoop.mapred.LocalJobRunner$Job statusUpdate
INFO:
Nov 21, 2014 3:47:25 PM org.apache.hadoop.mapred.Task sendDone
INFO: Task 'attempt_local891637895_0001_m_000000_0' done.
Nov 21, 2014 3:47:25 PM org.apache.hadoop.mapred.LocalJobRunner$Job$MapTaskRunnable run
INFO: Finishing task: attempt_local891637895_0001_m_000000_0
Nov 21, 2014 3:47:25 PM org.apache.hadoop.mapred.LocalJobRunner$Job run
INFO: Map task executor complete.
Nov 21, 2014 3:47:25 PM org.apache.hadoop.mapred.Task initialize
INFO:  Using ResourceCalculatorPlugin : org.apache.hadoop.util.LinuxResourceCalculatorPlugin@104faf8
Nov 21, 2014 3:47:25 PM org.apache.hadoop.mapred.LocalJobRunner$Job statusUpdate
INFO:
Nov 21, 2014 3:47:25 PM org.apache.hadoop.mapred.JobClient monitorAndPrintJob
INFO:  map 100% reduce 0%
```

Figure 6. Final Mapping Output

Figure 7 shows that as jobs are completed from the mapping functions, the reduce functions process the intermediate values and the meta-data associated with the completed tasks are processed by the NameNode. The NameNode keeps track of how much data is written and where it is located and what the data represents. During this process all completed jobs are logged for analysis.

```
Nov 21, 2014 3:35:33 PM org.apache.hadoop.mapred.JobClient monitorAndPrintJob
INFO:  map 100% reduce 100%
Nov 21, 2014 3:35:33 PM org.apache.hadoop.mapred.JobClient monitorAndPrintJob
INFO: Job complete: job_local729372904_0001
Nov 21, 2014 3:35:33 PM org.apache.hadoop.mapred.Counters log
INFO: Counters: 20
Nov 21, 2014 3:35:33 PM org.apache.hadoop.mapred.Counters log
INFO:   File Output Format Counters
Nov 21, 2014 3:35:33 PM org.apache.hadoop.mapred.Counters log
INFO:     Bytes Written=89
Nov 21, 2014 3:35:33 PM org.apache.hadoop.mapred.Counters log
INFO:   File Input Format Counters
Nov 21, 2014 3:35:33 PM org.apache.hadoop.mapred.Counters log
INFO:     Bytes Read=4138767
Nov 21, 2014 3:35:33 PM org.apache.hadoop.mapred.Counters log
INFO:   FileSystemCounters
Nov 21, 2014 3:35:33 PM org.apache.hadoop.mapred.Counters log
INFO:     FILE_BYTES_READ=8301494
Nov 21, 2014 3:35:33 PM org.apache.hadoop.mapred.Counters log
INFO:     FILE_BYTES_WRITTEN=149621
Nov 21, 2014 3:35:33 PM org.apache.hadoop.mapred.Counters log
INFO:   Map-Reduce Framework
Nov 21, 2014 3:35:33 PM org.apache.hadoop.mapred.Counters log
INFO:     Map output materialized bytes=23586
```

Figure 7. Reduce Output

Figure 8 shows that the output data from the MapReduce process contains specific data sets that were sought from our MapReduce program. The output is specific sections of the input data that meet our minimum requirements for identifying an earthquake. Data that was recorded when no earthquakes were occurring is omitted from the results. This reduces the amount of data that needs to be stored.

If calibrated correctly, the acceleration numbers should be around 0g. We are currently calibrating Shakeboxes. [1]

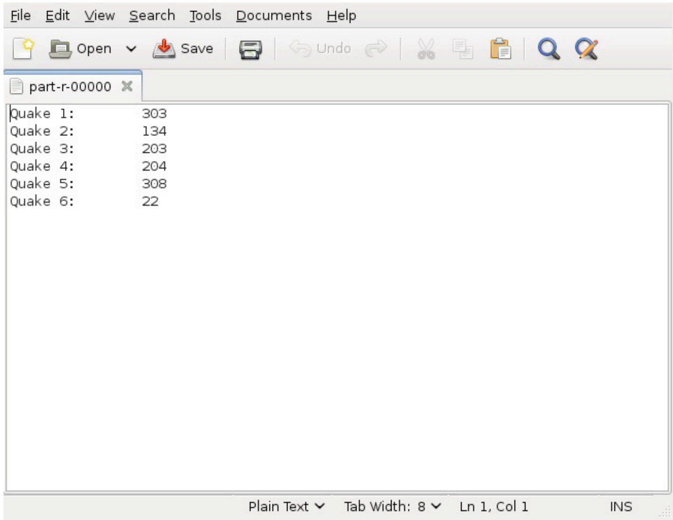


Figure 8. Reduce Output

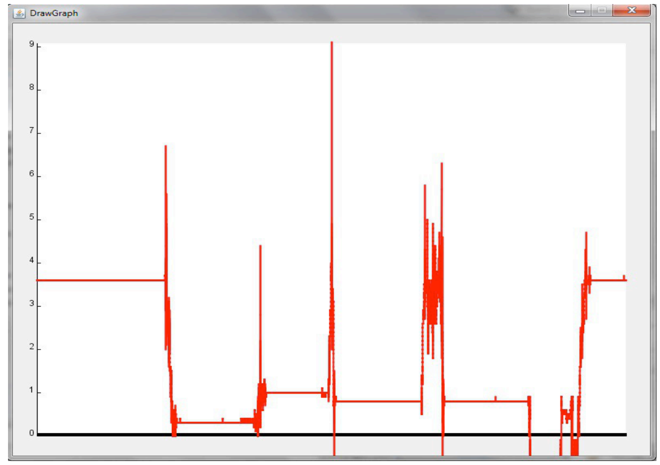
Finally, Figure 9 shows a visualization of a specific seismic data set from one Shakebox output, which is processed by our MapReduce functions on the larger data set. Figure 9 (a) (b) and (c) show the Shakebox movement along X-, Y-, and Z-axis respectively. The X plane graph is the movement along the X-axis of the shakebox (if you view it from above); the Y plane graph represents the movement along the Y-axis as you look at the box from above; the Z plane graph represents the movement of the shake box when it is lifted up and down.

VII. CONCLUSION AND FUTURE WORK

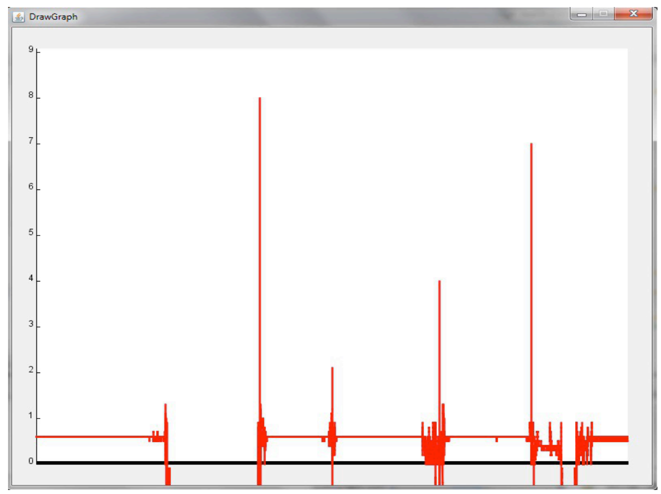
In this paper we studied a seismic sensing platform using Shakebox, a low-noise and low-power 24-bit wireless accelerometer sensor. To process large amount of seismic data from this platform, we adopted Hadoop/MapReduce. We designed map and reduce functions on the testbed data and analyzed the result. As ongoing effort and future work, we are working on the following two directions. Currently, the Shakeboxes are invoked from command line and only collect data for a specified amount of time. We are configuring Shakeboxes such that it can constantly collect data. This is critical to catch real earthquake occurrences. Second, we are planning to incorporate real earthquake models into our Haddop/Mapreduce analysis, to better evaluate its efficacy.

ACKNOWLEDGMENT

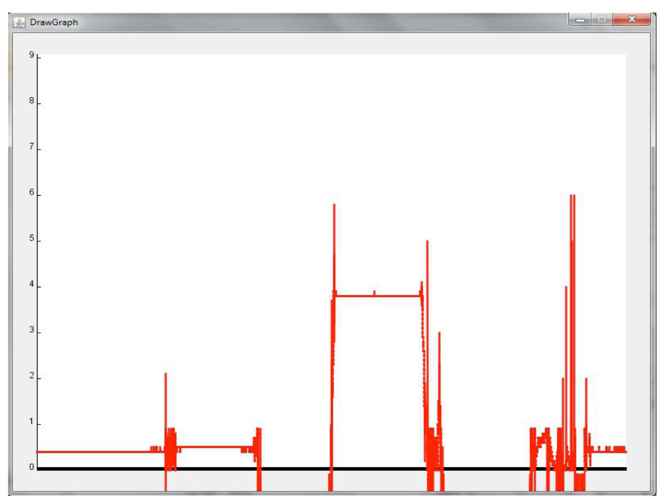
We would like to thank Dr. Monica Kohler for providing the Shakeboxes and many helpful suggetions. We would like to thank the Caltech Community Seismic Network team for many opportunities of participation and discussion. This work was supported in part by the NSF Grant CNS-1116849.



(a) X Plane



(b) Y Plane



(c) Z Plane

Figure 9. Shakebox Movement.

REFERENCES

[1] Matthew Faulkner, Robert Clayton, Thomas Heaton, K. Mani Chandy, Monica Kohler, Julian Bunn, Richard Guy, Annie Liu, Michael Olson, MingHei Cheng, and Andreas Krause, "Community Sense and Response Systems: Your Phone as Quake Detector," Communications of the ACM, vol. 57, no. 7, pp. 66–75, 2914.

[2] R. W. Clayton, T. Heaton, M. Chandy, A. Krause, M. Kohler, J. Bunn, R. Guy, M. Olson, M. Faulkner, M. Cheng, L. Strand, R. Chandy, D. Obenshain, A. Liu, and M. Aivazis, "Community Seismic Network," Annals of Geophysics, vol. 54, no. 6, 2012.

[3] Michael Olson, Annie Liu, K. Mani Chandy and Matthew Faulkner, "Rapid Detection of Rare Geospatial Events," 5th ACM international conference on Distributed event-based system, 2011.

[4] M Faulkner, M Olson, R Chandy, J Krause, KM Chandy, "The next big one: Detecting earthquakes and other rare events from community-based sensors,"10th International Conference on Information Processing in Sensor Networks (IPSN), 2011.

[5] Cochran, E.S., Lawrence, J.F., Kaiser, A., Fry, B., Chung, A., Christensen, C., "Comparison between low-cost and traditional MEMS accelerometers: a case study from the M7.1 Darfield, New Zealand, aftershock deployment," Annals of Geophysics, 54 (6), 728-737.

[6] Cochran E., Lawrence J., Christensen C., Chung A., A novel strong-motion seismic network for community participation in earthquake monitoring, IEEE Inst & Meas, 12, 6, 8-15, 2009.

[7] https://www.globalscaletechnologies.com/t-sheevaplugs.aspx

[8] Nilesh Mishra, Shuai Hao, Monica Kohler, Ramesh Govindan, and Robert Nigbor (2010) ShakeNet: A Tiered Wireless Accelerometer Network for Rapid Deployment in Civil Structures. http://nsl.cs.usc.edu/Papers/?action=download&upname=Mishra10a.pdf

[9] Chun-Hsiang Lee, David Birch, Chao Wu, Dilshan Silva, Orestis Tsinalis, Yang Li, Shulin Yan, Moustafa Ghanem, Yike Guo, "Building a Generic Platform for Big Sensor Data Application", IEEE International Conference on Big Data, 2013.

[10] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," 6th conference on Symposium on Opearting Systems Design & Implementation , pp. 137–150, 2004.

[11] "The Apache Hadoop Framework," http://hadoop.apache.org, 2013.

[12] Bin Liu, Haifeng Chen, Abhishek Sharma, Guofei Jiang, and Hui Xiong, "Modeling Heterogeneous Time Series Dynamics to Profile Big Sensor Data in Complex Physical Systems", IEEE International Conference on Big Data, 2013.

[13] Tian Guo, Thanasis G. Papaioannou and Karl Aberer, "Model-View Sensor Data Management in the Cloud", IEEE International Conference on Big Data, 2013.

[14] Jeongyeup Paek, Ben Greenstein, Omprakash Gnawali, Ki-Young Jang, August Joki, Marcos Vieira, John Hicks, Deborah Estrin, Ramesh Govindan, and Eddie Kohler, "*The Tenet Architecture for Tiered Sensor Networks",* ACM Transactions on Sensor Networks (TOSN), 6(4), 2010.

[15] http://boinc.berkeley.edu/