Big Data Analytics in Online Structural Health Monitoring

Guowei Cai[1], Sankaran Mahadevan[2]

[1,2] *Vanderbilt University, Nashville, TN, 37235, United States*
guowei.cai@vanderbilt.edu
sankaran.mahadevan@vanderbilt.edu

ABSTRACT

This manuscript explores the application of big data analytics in online structural health monitoring. As smart sensor technology is making progress and low cost online monitoring is increasingly possible, large quantities of highly heterogeneous data can be acquired during the monitoring, thus exceeding the capacity of traditional data analytics techniques. This paper investigates big data techniques to handle the high-volume data obtained in structural health monitoring. In particular, we investigate the analysis of infrared thermal images for structural damage diagnosis. We explore the MapReduce technique to parallelize the data analytics and efficiently handle the high volume, high velocity and high variety of information. In our study, MapReduce is implemented with the Spark platform, and image processing functions such as uniform filter and Sobel filter are wrapped in the mappers. The methodology is illustrated with concrete slabs, using actual experimental data with induced damage

1. INTRODUCTION

During the span of a structures service life, conditions such as wear, overload, environmental degradation, and natural disasters may accelerate the degradation of the material and the structure. Structural health monitoring (SHM) is a vital tool to ensure that the structure is reliable within the design life, and also to potentially extend the service life beyond the designed life (Naus, 2009). SHM techniques can be either data-driven or model-based. In both cases, the data is often obtained using non-destructive evaluation (NDE) techniques, which can be divided into active and passive techniques. Examples of active NDE techniques are electromagnetic testing (ET) (Nagy, 2016) and ultrasonic guided wave testing (UGWT) (Yan, Royer, & Rose, 2010). Examples of passive NDE techniques are acoustic emission (Nair & Cai, 2010), digital image correlation (DIC) (Roux, Rthor, & Hild, 2009), fiber-optic sensing (FOS) (López-Higuera, Cobo, Incera, & Cobo, 2011). Some other NDE techniques can be used in either active or passive modes, such as infrared thermography (IR) (Bagavathiappan, Lahiri, Saravanan, Philip, & Jayakumar, 2013). From the data type point of view, these monitoring techniques acquire either wave signals (ET, UGWT, AE), or images (DIC, IR). Data acquisition and analysis are crucial elements in structural health monitoring.

However, due to current trends in monitoring the entire structure, instead of focusing on only suspicious locations, the amount of acquired data is growing, which gradually increases the demands pressure on the data acquisition and analysis techniques. For example, 26 sensor arrays on the Vincent Thomas Bridge (VTB) in San Pedro, California generate 3 terabytes (TB) per year (Kallinikidou, Yun, Masri, Caffrey, & Sheng, 2013); in the health monitoring of wind turbine blades, over 300 GB acoustic emission data were sampled during 6 months (Anastasopoulos, Lekou, & Mouzakis, 2012); 7 GB of data were sampled per day in the Confederation Bridge Monitoring Project in Canada (Desjardins, Londono, Lau, & Khoo, 2006); over 20 GB of data were obtained during automated railway inspection in the city of Brockton, MA ((Zhang, Qiu, Shamsabadi, Birken, & Schirner, 2014). All these applications call for the introduction of big data analytics into structural health monitoring. (Mahadevan, Adams, & Kosson, 2012) pointed out the need for big data analytics in structural health monitoring, as one of the four elements in prognostics and health monitoring framework of concrete structures. The big data issue mainly affects two elements in structural health monitoring: data acquisition and data analytics. For data acquisition, data synchronization is a critical problem to solve, especially in a wireless sensor network. Several studies such as (Araujo et al., 2012), (Gandhi, Chang, & Trivedi, 2007), and (Yu, 2012) have studied this problem. This manuscript only focuses on the data analytics issues in SHM, in the presence of big data.

There are two different directions to pursue in solving the big data problem. First, when the data is too large to process, in order to reduce the computational cost, it may sometimes be desirable to compress the data before processing. Data compressed into feature vectors can help to reduce the dimension

of data, by exploiting statistical redundancy of the raw data (Sohn, Farrar, Hunter, & Worden, 2001). Additionally, another kind of reduction can be achieved via reducing the data size using samples of the data, known as compressive sensing. One example is the compressive sampling of accelerometer signals (Bao, Beck, & Li, 2010). While it seems to be a reasonable way to handle the voluminous data, one of the issues in data compression is the reduced accuracy of the detection, which sometimes leads to the low quality of the structural health monitoring, resulting in unreliable decision making.

In contrast to data compression, parallel and distributed computing offer alternatives to reduce the time cost of data analytics, without causing any precision loss. Parallel computing is more tightly connected to multi-threading, or how to make full use of a single CPU; Distributed computing refers to the notion of divide and conquer, executing subtasks on different machines and then merging the results. Theoretically, distributed computing is much more powerful, since more memory and CPU resources (from the cluster) are available, although the bandwidth among the connected computers can sometimes become the main limitation. Message Passing Interface (MPI) is one of the most popular distributed computing methods used for a long time, and applications can be found in structural health monitoring (Kiepert & Loo, 2012), (Chakraborty et al., 2009). MPI's goals are high performance, scalability, and portability. Another conceptually similar approach in the context of big data is MapReduce. Utilizing a cluster of nodes, MapReduce performs two essential functions: It parcels out work to various nodes within the cluster, and it organizes and reduces the results from each node into a cohesive answer to a query (Dean & Ghemawat, 2008).

Although the main purpose of both MPI and MapReduce is to improve the efficiency via parallelization, there are several differences between them. First, MPI is designed to handle large amounts of data exchange between computers, while MapReduce focuses on embarrassingly parallel implementation (no much information exchange among computers). Second, MPI is appropriate for iterative algorithms that are computationally expensive, whereas MapReduce is fit for the case where the expense is mainly caused by the data itself. Third, although MPI can also be built to be scalable and fault-tolerant, it needs much effort to ensure the performance and reliability of such a system, MapReduce on the other hand, is created to be easily scalable and fault-tolerant. A detailed discussion about relations between MPI and MapReduce can be found in (Chen, Song, Bai, & Lin, 2011).

The main contribution of this paper is to investigate image processing with the MapReduce technique to handle the big data challenge in structural health monitoring. The particular focus in on thermal image processing, and the implementation of MapReduce for edge detection using the Sobel filter, in order to detect the damage. The proposed approach is demonstrated with a concrete slab.

The rest of this paper is organized as follows. Section 2 provides a background review of the basic concepts related to SHM and big data analytics. Section 3 develops the big data analytics approach to structural damage diagnosis, especially the implementation of MapReduce to edge detection. Section 4 implements the proposed approach using an illustrative example of detecting holes in a concrete slab using infrared thermography, and discusses the performance of the MapReduce methodology. Section 5 provides concluding remarks.

2. BACKGROUND

2.1. Online Structural Health Monitoring

The purpose of online structural health monitoring is to detect the damage in the structure, analyze future risk, predict the remaining useful life, and guide the maintenance/repair actions if needed. In the context of pattern recognition (Farrar, Doebling, W., & Nix, 2001), a four-step procedure is described: (1) Operational evaluation, (2) Data acquisition and cleansing, (3) Feature selection, and (4) Statistical model development. Operational evaluation defines what will be monitored and how the monitoring process will be implemented. Data acquisition and cleansing will define what data will be sampled and processed, and how the data will be sampled (i.e., in what frequency, how long it will be recorded, and how it will be preprocessed). The feature selection step defines the features that will be selected and the statistical distributions of the features. In the statistical model development step, the model will be developed to detect the damage, predict remaining useful life, and quantify the uncertainty. In this paper, only deterministic structural health monitoring is considered in the context of big data, so step (4) will develop a deterministic model which will be used only to detect the damage, without considering any uncertainty sources.

2.2. Image Processing

Digital image is one type of data format acquired and used in structural health monitoring. Damage is detected, located and quantified by comparing the image of the damaged structure against that for the intact structure, using image processing techniques. The general procedure described in (Baxes, 1994), is shown in Figure 1. After obtaining the raw image, preprocessing techniques (e.g. cropping, baseline removal and noise reduction) can be applied to prepare for edge detection, which can lead to damage detection. Noise reduction and edge detection are computationally expensive, and can benefit from the application of big data techniques.

2.3. MapReduce Framework

MapReduce is a framework designed for processing large datasets, by utilizing multiple nodes (machines) for the com-

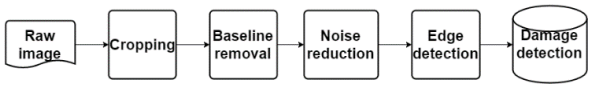

Figure 1. The image to be processed

putations. It takes key/value pairs as inputs and generates the other key/value pairs as outputs. A key-value pair (KVP) is a set of two linked data items: a key, which is a unique identifier for some item of data, and the value, which is either the data that is identified or a pointer to the location of that data. As mentioned earlier, the MapReduce framework can be split into two steps: *Map* and *Reduce*, both of which are created by the user. Before applying the MapReduce model, the user will need to write the input as the key/value pair. The key/value pair (k1, v1) will then be input to the *Map* function, which will generate the intermediate key/value pairs (k2, v2). Then the intermediate key/value pairs are passed to the reduce function, which merges together these values to form a smaller set of values. The process is shown below, which allows us to handle lists that are too large to fit into the memory:

$$\text{map(k1, v1)} -> \text{list(k2, v2)}$$
$$\text{reduce(k2, list(v2))} -> \text{list(v3)}$$

An example of counting words occurance in documents can help to understand the MapReduce process (Dean & Ghemawat, 2008). In Pseudocode 1, the *Map* function emits each word plus an associated count of accurrences (just '1' in this example). The *Reduce* functions sums together all counts emitted for a particular word.

The usage of MapReduce is very flexible, depends on the problems to be solved. One typical usage of MapReduce is shown in the 'Word Count' example above, where the intermediate key/value pairs (k2/v2) are the word/counts. Here k1/v1 refer to the file/contents. Sometimes when the *Map* function is used only, intermediate key/value pairs are the actual results, as we have in our application on image processing.

```
Pseudocode 1:
Map(String key, String value):
    // key: document name
    // key: document contents
    for each word w in value:
      EmitIntermediate(w, "1");

Reduce(String key, Iterator values):
  // key: a word
  // key: a list of counts
  int resut = 0;
  for each v in values:
    result += ParseInt(v);
    Emit(AsString(result));
```

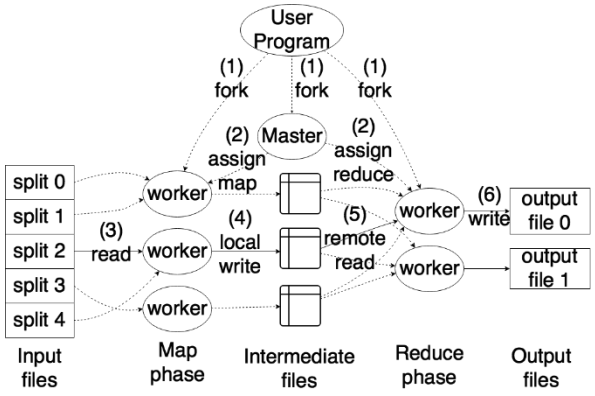

Figure 2. MapReduce execution overview

A cluster of computers (nodes) are used to implement this framework (Figure 2). One of them is the master node and the others are slave nodes (workers). MapReduce framework will split one path of execution into multiple concurrent paths of execution. Each path is called a fork. In figure 2, it splits execution onto master node and slave nodes. The way master node assign *Map* or *Reduce* task is controlled by the cluster manager (such as Mesos in Apache Spark (Zaharia et al., 2012)). First, the input files are parsed and split into smaller pieces (size 16MB to 64MB). The cluster manager will first assign *Map* tasks to all available slave nodes, and assign them *Reduce* tasks when any of the *Map* tasks finished. Notice that the number of workers assigned with *Map* tasks and *Reduce* tasks can be different, depends on the number of nodes available when assigning the tasks. Local write means the *Map* works write outputs (intermediate key/value pairs) to its own hard disk, and remote read means the other workers who were assigned *Reduce* tasks retrieve the intemediate key/value pairs remotely.

2.4. Spark

While there are different implementations of MapReduce, Apache Spark (Zaharia et al., 2012) is the one chosen in this study. Spark is an open source cluster computing framework, and API (Application Program Interface) for Java, Scala and Python are available, which is convenient for non-computer science programmers. Beside the basic capability of using the MapReduce methodology, Spark employs Resilient Distributed Datasets (RDD) that enable efficient data reuse in a broad range of applications. RDD is a read-only, partitioned collection of records, which can only be created through deterministic operations on either (1) data in stable storage or (2) other RDDs. An RDD has enough information about how it was derived from other datasets to compute its partitions from data in stable storage (Zaharia et al., 2012). Transformations are the manipulations on the data to be analyzed. There are two types transformations: coarse-grained transformation and fine-grained transformation. Coarse-grained transforma-

tion is applied on the entire dataset (e.g. filtering the whole datasets), while fine-grained transformation works on a single data point (e.g. index of certain data point). Spark applies coarse-grained transformations (e.g., map, filter and join) to allow the fault-tolerance feature. Instead of storing the actual data, the logging of the transformation can ensure that there is enough information to redo the operation if an RDD is lost. Due to the adoption of RDD, iterations in the computational algorithm do not need to repeatedly execute the reading and writing operations on the file system; this greatly reduces the computational cost in machine learning (Figure 3).

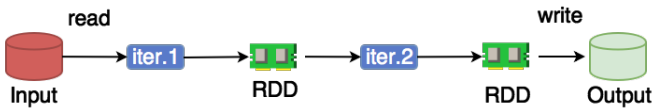

Figure 3. Use of resilient distributed datasets (RDD) in Spark

3. METHODOLOGY

When numerous images (Gigabytes or Terabytes of data) are collected in structural health monitoring, the data is big and a traditional data processing framework (storage, processing and manipulating) is not feasible; therefore a big data analytics framework needs to be employed. The methodology to apply the big data technique in online health monitoring will be developed in detail in this section. Structural health monitoring systems have the following elements: structure, sensors, data acquisition system, data transfer and storage mechanisms, data processing, and data manipulation. Each elements relation to big data are discussed below. Large volume of data can be caused by the size of the structure being monitored, or by the number of sensors. The structure gives the scope, and the sensors give the resolution.

3.1. Structure

In SHM, the engineering structure is the target to be monitored and regarding which the decision needs to be made (whether to use, maintain, repair or retire the structure based on the diagnosis result). For example, suppose instead of the piers of the bridge to be monitored, the health of the whole bridge (deck, load-carrying elements, piers, and foundations) is able to be evaluated, with the processing ability of big data. In this case, the resolution is not changing, but the data volume is greatly enlarged.

3.2. Sensors

As mentioned earlier, another cause of big data in SHM is resolution. Similar to the monitoring scope, the number of sensors can be increased with the data processing ability provided by the big data techniques. With more sensors used in monitoring, more information will be available for analysis. The sensor configuration will be discussed in the section below.

3.3. Data Acquisition, Transfer and Storage

In the monitoring process, data will be generated by sensors, and then interpreted and transferred to the data processing computer, via data acquisition system (DAQ). The sampling rate is controlled by the DAQ device, which directly affects the resolution and data size. After acquired by the DAQ device, the data is stored in the computer (often a laptop) connected with the DAQ device. The next step is to transfer the data to the cluster. For the Linux or Mac system, the command for data uploading is scp. The syntax of scp is:

scp -r /local/path/to/foo user@your.server.example.com:/cluster/path/to/foo

Here /local/user/path/to/foo means the local folder, while user@your.server.example.com:/cluster/path/to/foo means the target folder in the cluster, and -r means recursively copy the files in the folder. In computer science, 'foo' is commonly used as a placeholder name. When the operating system for the client computer is Windows, a similar command can be used after installing WinSCP or PuTTY. The transferring speed is limited by the devices on both ends, and by the bandwidth of the connection between the client and cluster.

Normally the MapReduce application is automatically paired with the corresponding file system, such as Hadoop with HDFS (Hadoop Distributed File System), Amazon EMR with Amazon S3, and Windows Azure and WASB (Windows Azure Storage Blobs). However, the user can also choose a different file system other than the default paired one, when it is more applicable to do so. For example, here we use Spark, paired with GPFS (General Parallel File System). Additionally, the distributed file system will divide the large data file into blocks (normally 64 MB to 128 MB, and normally the user is allowed to change the block size in the actual application of MapReduce).

3.4. Data Processing

As reviewed previously, there might be different data formats to be processed in structural health monitoring. Here we consider thermal image processing as an example. The common procedure for processing digital images is: cropping, baseline removal, noise cancellation and feature extraction. Each image is composed by pixels (Figure 12 for example), where each pixel represents the temperature of the location .

3.4.1. Baseline Removal

Baseline removal subtracts pixel values by the corresponding pixel from an image of the control group. It happens when the control group is available. This can enhance signal characteristics for diagnosis.

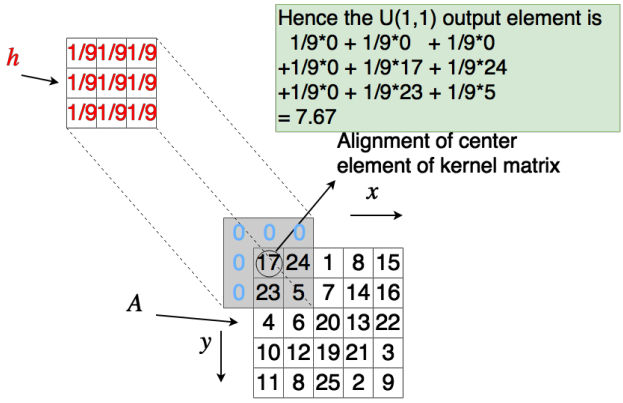

Figure 4. The uniform filtering example

3.4.2. Cropping

The cropping is realized by only storing and plotting the corresponding part of the target structure we analyze. Compared with the raw image, the temperature contour of cropped image is zoomed in (Figure 13). Normally since in the observation procedure, of the locations of the structure and camera do not change, the cropping pixel range for all the images is the same.

3.4.3. Noise Cancellation

Uniform filtering is used for the purpose of noise cancellation. The basic idea is to average each pixel by the value of adjacent pixels. Notice that uniform filtering is different from simple moving average (SMA), in thatuniform filter is doing averaging by putting the target point in the center while SMA is doing biased averaging. Mathematically, the uniform filtering process is basically a 2D convolution operation. To illustrate the convolution operation, the 1D convolution operator formula is defined in Eq. 1, in which f is the uniform kernel, and g is the image matrix to be operated on. The kernel can be of different sizes, and Figure 4 shows how a kernel with size 3×3 works on a 5×5 target matrix. To perform convolution, first align the center element of the kernel matrix with the the element on the target matrix, and then sum up the multiplication between all aligned element-pair. For example, the convolution on the element (1, 1) is 7.67, as is shown in Figure 4. Move the kernel along x and y axis until convolution of all elements are carried out. (Jain, Kasturi, & Schunck, 1995) can be referred for detailed implementation. After the uniform filtering, the image is smoothed, i.e.,, more continuous everywhere (Figure 14).

$$\begin{aligned} m(f * g)(t) &\stackrel{\text{def}}{=} \int_{-\infty}^{\infty} f(\tau) g(t-\tau) d\tau \\ &= \int_{-\infty}^{\infty} f(t-\tau) g(\tau) d\tau \end{aligned} \tag{1}$$

(a) (b)

Figure 5. The Sobel fileter kernels: (a) kernel for x direction, and (b) kernel for y direction

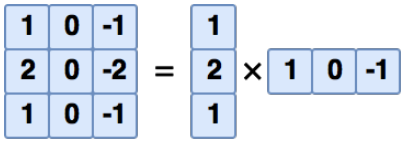

Figure 6. Split of Sobel filter kirnel (x direction) into averaging and differencing

3.4.4. Feature Extraction

Sobel filter is used here for the feature extraction, based on the image obtained after uniform filtering. The other edge detection algorithms such as Canny, Prewitt, Robert, Laplacian and Laplacian of Gaussian filters were tried and found that Sobel filtering performed best in our problem. The selection of algorithm would be problem-dependent and any desired algorithm can be plugged in our big data analytics framework in the same way as Sobel filtering. The basic idea behind Sobel filter is similar to the uniform filter, which is also a 2D convolution operation, where the only difference is the filter kernel. Similar to the uniform filter, Sobel filter can also be performed with different sizes. The difference is that for uniform filtering, there is only one kernel, which is a nxn matrix filled with the value $1/n^2$. For Sobel filtering, the filters for x and y directions can be different (Figure 5). Additionally, the kernel can be split into the product of two 1D kernels, for averaging and differencing in two directions (Figure 6). To differentiate the damaged area, gradient ranges in both x and y directions are needed, and thresholds will be applied to detect the edges of damages.

3.5. MapReduce Implementation

The basic idea of the application of data processing in MapReduce is to divide the files into different partitions (each partition contains multiple files), and then perform the mapping and reducing operations separately. To fully use the resources, the number of partitions is always greater than the number of instances (i.e. cores, of which each node might contain a multiple). For example, if the number of files to be analyzed is 100, and the number of cores available is 20, the number of partitions should be at least 20. Otherwise some of the cores will be idle.

In structural health monitoring, the data is normally sampled

as separate files (images or signals). For each image and signal, a separate processed result is obtained, without combination (Figure 7). In that case, the *Reduce* function is omitted, and only the *Map* function remains. All the data processing functions on the assigned files are combined within a single *Map* function. The *Map* function is defined by the user, in which the reading, processing, and writing functions are all included, as shown by the pseudocode below:

```
Pseudocode 2:
Map(x):
    function InputData = ReadData(x);
    function OutputData = Processing(InputData);
    function WriteData(OutputData);
    return (x, 0)
SparkContext(appName="myApp").parallelize(Filelist,
N).map(mapper).count()
```

The above pseudocode has two steps. First, a *Map* function is defined (mapper), within which all the actual data processing functions are defined (reading, filtering, writing). The argument x is the file to be analyzed, which is assigned by the task manager. As discussed previously, since there is only the *Map* function, the input file can be mapped with any value (here we mapped x to 0). The reason it can be any value is here we only use the Map function to trigger the parallelization, without caring for the output of the Map function. A 'Word Count' example has been added. The second step, SparkContext, represents the connection to the cluster, which is the main class in Spark; parallelize is the method to split the input files into N partitions; and *map* is the method to call the *Map* function defined in the first step and to pass the input file to it. The count method is used to count the number of outputs. The number of outputs is not of interest, since the result has already been obtained in the *Map* function. However, it is needed since the transformations (parallelize, map) only created the RDD instance, which needs some actions to execute it.

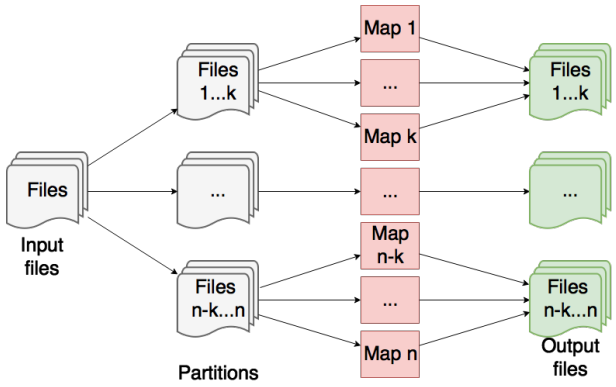

Figure 7. Schematic description of the MapReduce process

After the cluster finishes all the tasks, the results are stored in the designated directory defined in the WriteData function. Then the next step is to retrieve the data files from the cluster to the local computer, since normally it is not convenient to visualize the data remotely on the cluster. To transfer data back from the cluster, the user can use the scp command similar to the one used for transferring the data to the cluster.

Operations for image processing (cropping, uniform filtering, and Sobel filtering) need to be applied on all the images, with all parameters (cropping range, uniform filtering kernel size, and Sobel filtering gradient cutoff) remaining unchanged. As defined earlier, the reading, writing and processing functions are all included within the *Map* function. There are three sub-functions: Cropping, UniformFilter and SobelFilter. The processing function is defined using the psudocode below:

Several remarks about the processing function are in order. First, the input data is no longer a key/value pair but is an actual image (pixel matrix). Second, the sub functions inside will be sequentially executed, since the outputs of each sub function will be fed into the next sub function as inputs. Third, the sub functions (Cropping, UniformFilter, and SobelFilter) can be replaced easily with other functions according to the actual data processing task.

In summary, the steps for the big data analytics of image processing in structural health monitoring are: (1) upload the acquired data from the local computer to clusters; (2) prepare the image processing functions, and substitute into the Map function shown in Pseudocode 2; and (3) run Spark to process and retrieve the data files from the cluster back to the local computer.

4. NUMERICAL EXAMPLE

This example illustrates the basic application of big data analytics in structural health monitoring. The purpose of the monitoring in this example is to detect holes drilled into a $15.5 \times 15.5 \times 2\ in^3$ concrete slab (Figure 8) using infrared thermography imaging. Holes of 1/2 inch, 3/8 inch, and 5/16 inch diameter (all of them 4.45 inch deep) were drilled into the side of the concrete slab, as shown in Figure 9. The holes are required to be detected by the monitoring technique in this example. Since the focus of this paper is the application of big data technique to structural diagnosis, we use the holes only to illustrate this capability. In this case, the ground truth is known, which facilitates performance evaluation of the monitoring technique. In realistic situations, concrete damage could be of many types (physical, chemical, and mechanical), due to various causes such as freeze-thaw, chloride penetration, alkali-silica reaction etc. Temperature, humidity, and the properties of the concrete constituents (cement, aggregates, reinforcing steel, water content, and chemical admixtures) play a crucial role in the evolution of various types of damage. Damage in concrete eventually manifests

as cracks, delamination, spalling etc., and the edge detection approach illustrated here could be applied to different situations.

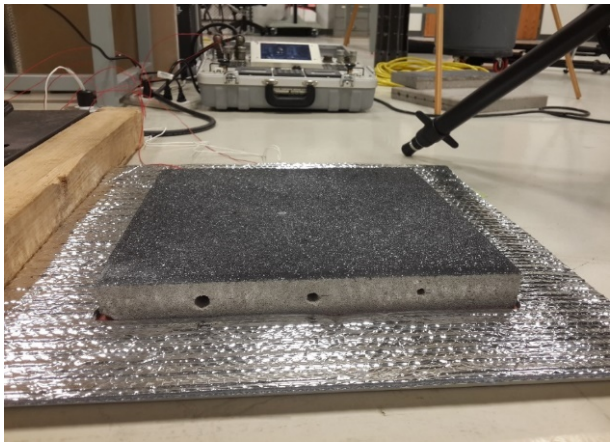

Figure 8. Thermography camera and the specimen to be monitored

4.1. Experimental Setup

The mechanics of damage detection using infrared thermography is based on the differences in heat transfer properties of different materials. The air in the drilled holes in the structure has much lower thermal conductivity coefficient than concrete, which will lead to a lagging phenomenon, i.e., the heating and cooling time of the hole are slower than the surrounding solid region. The slab is placed on a HEATCON thermal blanket and uniformly heated from below. The infrared thermography camera can detect the temperature of the surface of the slab (Figure 8, Figure 9) and store the temperature values as images via the DAQ system. We also place reflective material around the slab, in order to prevent direct heat transfer from the thermal blanket to the air around the slab; thus the thermal camera detects the temperature change on the top surface slab mainly caused by the heat transfer from the blanket through the slab.

4.2. Thermal Loading

Each thermal cycle has a total duration of 70 minutes. The heating profile is shown in Figure 10. A HEATCON composite system controller was connected to the thermal blanket and used to program a defined thermal cycle that can be repeated as many times as needed for a test. Two thermocouples were used to measure and monitor the heat applied by thermal blanket. One thermocouple was placed beneath the blanket and the other thermocouple was placed between the thermal blanket and the concrete sample (Figure 11).

For thermographic imaging, a FLIR Infrared (IR) camera is used to detect the temperature contours on the surface of the concrete slab. These contours can be analyzed to detect flaws

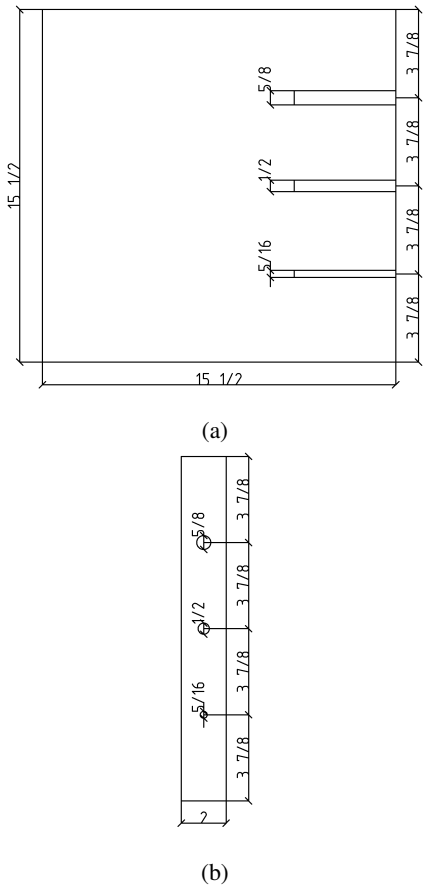

Figure 9. Sketch of the specimen (a) top view (b) side view

or defects inside the slab that cannot be easily detected by visual inspection. The FLIR IR camera was setup to capture images of the concrete slab every 1 second.

4.3. Data Acquisition System

The FLIR IR software is an integrated environment that allows the user to configure the sampling rate, resolution, and storage. Also the software can visualize the current captured image, and store the images in the designated path in the .tls format, which is specially used by this software.

4.4. Data Transfer and Storage Mechanism

After the sampling is completed, the data stored in the file *.tls can be exported in different format, such as .csv, .m, .txt, .jpeg. In this study, we used .csv to represent each image. For the heat loading period considered, 4231 images were sampled, and the total size is 19.4 Gb. The *.tls file is stored in the computer connected with the DAQ system, and the size is much smaller. The exported .csv files were stored in a portable drive, through which they were transferred to the analysis computer client. In order to use MapReduce to analyze the data, the data was uploaded to the cluster, which

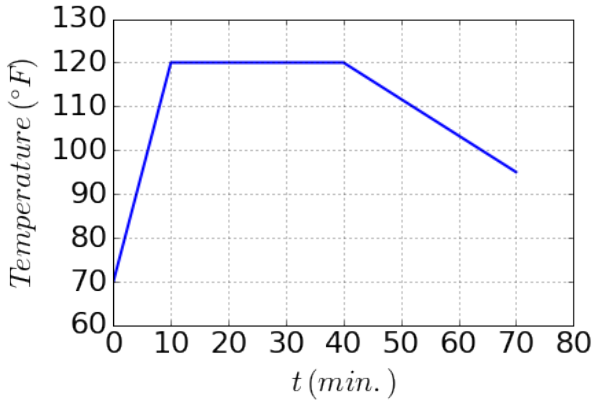

Figure 10. Thermal loading time history (scaled values)

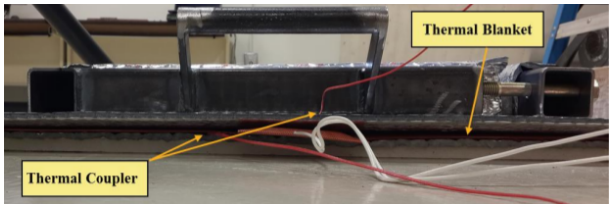

Figure 11. Thermal blanket and thermo couple

in this case was ACCRE (Advanced Computing Center for Research and Education) at Vanderbilt University.

4.5. Data Processing

The implementation of various steps in processing the thermal image data are discussed in detail and the results are presented below.

4.5.1. Baseline Removal

As reviewed previously, the common procedure for processing digital images consists of: cropping, baseline removal, noise cancellation and feature extraction. In this example, results can be obtained without control group. Thus there is no baseline removal needed here. This can save almost half the cost of data storage. For each image, the resolution is 640×512 pixels (Figure 12).

4.5.2. Cropping

Figure 12 shows the raw thermography image of the top surface of the slab and reflective material, 2835 seconds after start of the heating. Notice that the area corresponding to the slab has much higher temperature compared to the surrounding reflective material. Thus the image needs to be cropped in order to achieve greater resolution in analyzing the temperature distribution within the slab. After several trials, the appropriate pixel range for cropping was found to be $[83 : 518, 25 : 460]$. The cropped image is shown in Figure 13.

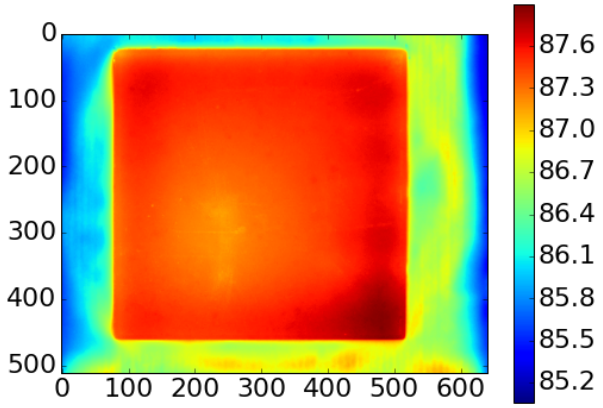

Figure 12. Example of raw image before cropping (t = 2835 s)

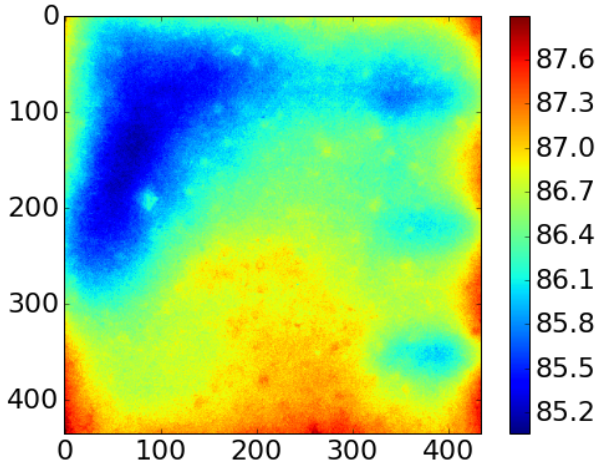

Figure 13. Cropped image (t = 2835s)

The image shows boundary effects, where additional heat may be introduced from the area around the slab, since the reflective material may not block all of the heat from the thermal blanket, especially since there was a small gap between the slab and the reflective material. It is also seen that there is a large area on the upper left quadrant, where the temperature is low. It may be due to the non-uniformity of the heating setup (such as lack of contact between slab and blanket), and heterogeneity of the concrete slab; the feature extraction step will reveal whether these effects are significant. As explained in the methodology section, the cropping pixel range for all the images are the same.

4.5.3. Noise Cancellation

A 22×22 kernel uniform filtering is used for noise cancellation, as shown in Figure 14. It can be observed that after the uniform filtering, the image is smoother. By doing this,

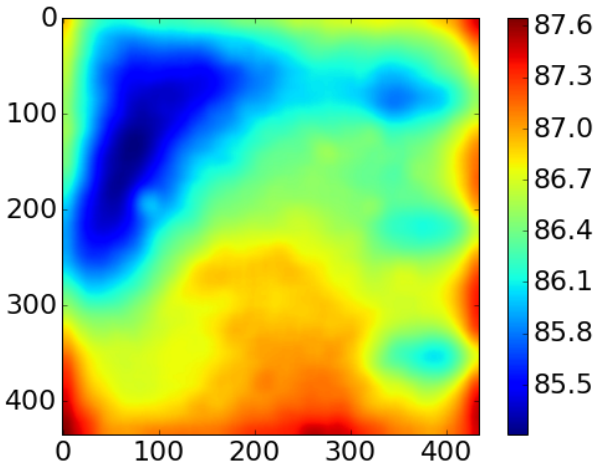

Figure 14. Image after uniform filtering (t = 2835 s; 22x22 kernel)

the noise in the image is greatly reduced. Note that Figure 14 roughly indicates the three holes in the right hand side. There is also a large, low temperature area on the left, but this gets eliminated in the subsequent feature extraction step.

4.5.4. Feature Extraction

Sobel filter is used for the feature extraction, based on the image obtained after uniform filtering. After applying Sobel filtering, the image shows the detected holes in the slab (Figure 15 (a)). The holes are detected by first obtaining the upper edges (yellow region on the right hand side in Figure 15 (a)) and lower edges (red region on the right hand side in Figure 15 (a)), and then plot the region between. The thresholds for obtaining upper edges are $[-0.050, 0.050]$ for x and $[0.020, 0.050]$ for y, and the thresholds for obtaining lower edges are $[-0.050, 0.050]$ for x and $[-0.100, 0.013]$ for y. Notice that the thresholds for x for both cases are the same, this is due to the hole directions being horizontal so that only the gradient in x direction is enough for the detection. For a more complicated hole or damage area, gradients in both x and y are needed for the detection of edges. Also notice that some noise is found on the left side of the slab, as shown in 15 (a). This is mainly due to the heterogeneity of concrete, and also uneven heating by the thermal blanket. The comparison of detected region and actual holes is shown in Figure 15 (b), and visual comparison shows good agreement; a more quantitative comparison is discussed below.

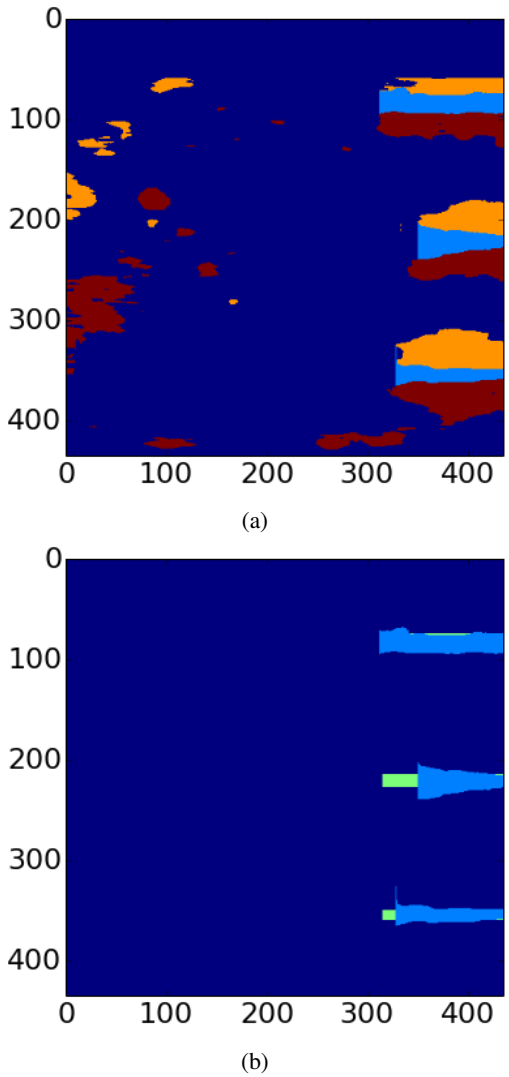

Figure 15. Image after Sobel filtering (a) holes detection based on the upper and lower edges (b) comparison between detected holes and ground truth; blue: detected holes, green: ground truth

4.6. Performance Discussion

Now we discuss the hole detection performance for different sample rates. In order to evaluate the performance quantitatively, a score is defined as the ratio of correctly detected area to the total detected area. As the sampling rate increases, the score grows accordingly (Figure 16). The score increases by almost 40% (i.e., $100\% \times (0.723 - 0.523)/0.523$), as the sampling interval decreases from 2 mins to 1 second. This indicates that by increasing the sample rate, the damage detection performance can be greatly improved. However, this increases the demand on the data analytics computation, which is resolved by the MapReduce technique.

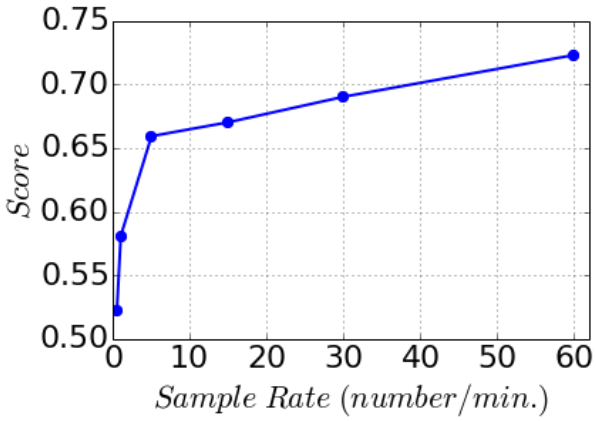

Figure 16. Detection performence vs. sampling rate

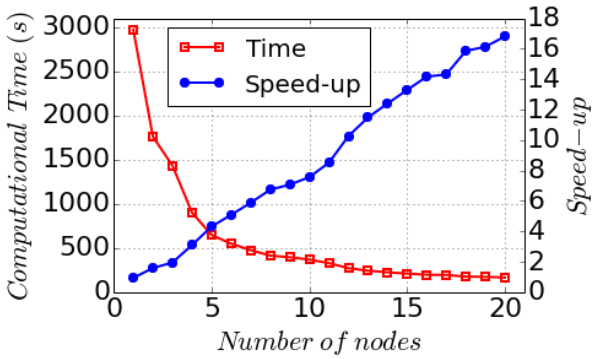

Figure 17. Performence vs. Number of nodes (left y-axes: computation time; right y-axis: speedup)

Table 1. Time cost of traditional method and MapReduce method

Method	Time (s)
Traditional	1560
MapReduce (20 nodes)	163

Table 2. Configuration of computers used by traditional method and MapReduce method

Method	GPU (GHZ)	Memory (GB)
Traditional	3.4 x 8	12
MapReduce (20 nodes)	2.3	5

Table 3. Time cost of individual steps in data processing

Step	Time (s)
Data Reading	0.14
Cropping	0.08
Uniform filtering	0.08
Sobel filtering	0.07

Compared with the traditional single machine computation, the computational expense (time cost) is greatly reduced as shown in Table 1. Notice that via distributed computation, the time cost is only 10% of local computation. It can be seen that as the number of nodes being used increases, the corresponding speedup increases almost linearly, which illustrates the scalability of MapReduce. Also notice that as the number of nodes increases, the computational time decays similar to exponential decay (Figure 17).

However, the time spent by the traditional method is 1560 s, while the MapReduce method on a single node takes as much as 2971s. This is due to two reasons. First, the operations related to MapReduce such as data transferring, data splitting, task managing, and mapping cost additional time. Second, the CPU and memory of the cluster node is less powerful (in this example) than the computer client used for local traditional computation (Table 2). The time cost of individual step in data processing (for one image) is shown in Table 3. For this simple case, data reading accounts for a large portion of the total time. However, for more complicated data processing, actual processing is expected to occupy a much larger portion.

5. CONCLUSION

This paper developed a framework for applying a big data technique to online structural health monitoring. The popular MapReduce approach was applied in the proposed framework, and realized via Apache Spark. Structural damage detection was parallelized via MapReduce, by transforming inputs and outputs as key-value pairs. Sobel filter was used for illustration of the image processing. It can be easily replaced with other appropriate techniques for different scenarios. Results show that the processing effort scaled well, in an almost linear trend. The approach was illustrated for the processing of thermal images obtained for a concrete slab, and the data volume is less than 20 Gb. For practical structural health monitoring for the whole structure in the field, the data can be very large, thus considerably increasing the advantage of MapReduce in realistic application.

Future research needs to address several extensions. This paper only considered the application of big data techniques to deterministic structural health monitoring; extension to uncertainty quantification in diagnosis needs to be considered in future work. Second, this paper did not consider the complexity problem of parallelization in MapReduce, which can lead to different parallelization options via splitting the task data-

wise or functionally. Third, fault-tolerance is an important issue in big data analytics, which needs to be incorporated in future work.

ACKNOWLEDGMENT

This study was partly supported by funding from the U.S. Department of Energy (DOE) through Light Water Reactor Sustainability (LWRS) Program (Monitors: Vivek Agarwal and Bruce Hallbert, Idaho National Lab). The support is gratefully acknowledged.

REFERENCES

Anastasopoulos, A., Lekou, D. J., & Mouzakis, F. (2012, September). Health monitoring of a neg-micon nm48/750 wind turbine blades with acoustic emission. In *European conference on acoustic emission testing & 7th international conference on acoustic emission university of granada* (p. 12-15). Granada, Spain.

Araujo, A., Garca-Palacios, J., Blesa, J., Tirado, F., Romero, E., Samartn, A., & Nieto-Taladriz, O. (2012). Wireless measurement system for structural health monitoring with high time-synchronization accuracy. *IEEE Transactions on instrumentation and measurement*, *61*(3), 801-810.

Bagavathiappan, S., Lahiri, B. B., Saravanan, T., Philip, J., & Jayakumar, T. (2013). Infrared thermography for condition monitoringa review. *Infrared Physics & Technology*, *60*, 35-55.

Bao, Y., Beck, J. L., & Li, H. (2010). Compressive sampling for accelerometer signals in structural health monitoring. *Structural Health Monitoring*, *0*(0), 1-12.

Baxes, G. A. (Ed.). (1994). *Digital image processing: principles and applications*. John Wiley & Sons.

Chakraborty, D., Kovvali, N., Wei, J., Papandreou-Suppappola, A., Cochran, D., & Chattopadhyay, A. (2009). Damage classification structural health monitoring in bolted structures using time-frequency techniques. *Journal of Intelligent Material Systems and Structures*, *20*(11), 289-305.

Chen, W. Y., Song, Y., Bai, H., & Lin, E. Y., C. J.and Chang. (2011). Parallel spectral clustering in distributed systems. *IEEE transactions on pattern analysis and machine intelligence*, *33*(3), 568-586.

Dean, J., & Ghemawat, S. (2008). Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, *5*(1), 107-113.

Desjardins, S. L., Londono, N. A., Lau, D. T., & Khoo, H. (2006). Real-time data processing, analysis and visualization for structural monitoring of the confederation bridge. *Advances in Structural Engineering*, *9*(1), 141-157.

Farrar, C. R., Doebling, W., S., & Nix, D. A. (2001). Vibrationbased structural damage identification. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, *359*(1778), 131-149.

Gandhi, T., Chang, R., & Trivedi, M. M. (2007). Video and seismic sensor-based structural health monitoring: Framework, algorithms, and implementation. *IEEE Transactions on intelligent transportation systems*, *8*(2), 169-180.

Jain, R., Kasturi, R., & Schunck, B. G. (1995). *Machine vision* (Vol. 5). McGraw-Hill New York.

Kallinikidou, E., Yun, H. B., Masri, S. F., Caffrey, J. P., & Sheng, L. H. (2013). Application of orthogonal decomposition approaches to long-term monitoring of infrastructure systems. *Journal of Engineering Mechanics*, *139*(6), 678-690.

Kiepert, J., & Loo, S. M. (2012). A unified wireless sensor network framework. In *Systems conference (syscon), 2012 ieee international* (p. 1-6).

López-Higuera, J. M., Cobo, L. R., Incera, A. Q., & Cobo, A. (2011). Fiber optic sensors in structural health monitoring. *Journal of Lightwave Technology*, *29*(4), 587–608.

Mahadevan, S., Adams, D., & Kosson, D. (2012). Challenges in concrete structures health monitoring. In *In proceedings, annual conference of the prognostics and health management society.*

Nagy, P. B. (2016). Electromagnetic nondestructive evaluation. *Ultrasonic and Electromagnetic NDE for Structure and Material Characterization: Engineering and Biomedical Applications*, 169.

Nair, A., & Cai, C. S. (2010). Acoustic emission monitoring of bridges: Review and case studies. *Engineering structures*, *32*(6), 1704-1714.

Naus, D. J. (2009). The management of aging in nuclear power plant concrete structures. *Journal of Metals*, *61*(7), 35-41.

Rens, K. L., Wipf, T. J., & Klaiber, F. W. (1997). Review of nondestructive evaluation techniques of civil infrastructure. *Journal of performance of constructed facilities*, *11*(4), 152-160.

Roux, S., Rthor, J., & Hild, F. (2009). Digital image correlation and fracture: an advanced technique for estimating stress intensity factors of 2d and 3d cracks. *Journal of Physics D: Applied Physics*, *42*(21), 214004.

Sohn, H., Farrar, C., Hunter, N., & Worden, K. (2001, Jan.). *Applying the lanl statistical pattern recognition paradigm for structural health monitoring to data from a surface-effect fast patrol boat* (Tech. Rep.).

Yan, F., Royer, R. L., & Rose, J. L. (2010). Ultrasonic guided wave imaging techniques in structural health monitoring. *Journal of Intelligent Material Systems and Structures*, *21*(3), 377-384.

Yu, L. (2012). Acoustic emission source localization on con-

crete structures with focusing array imaging. In *In 6th european workshop on structural health monitoring.*

Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., & Stoica, I. (2012, Apr.). Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *In proceedings of the 9th usenix conference on networked systems design and implementation* (p. 2-2).

Zhang, J., Qiu, H., Shamsabadi, S. S., Birken, R., & Schirner, G. (2014, Jul.). Sirom3–a scalable intelligent roaming multi-modal multi-sensor framework. In *Iin 38th ieee international conference on computers, software and applications* (p. 446-455).