Smart Structures and Systems, *Vol. 17, No. 4 (2016) 669-690*
DOI: http://dx.doi.org/10.12989/sss.2016.17.4.669

A NoSQL data management infrastructure for bridge monitoring

Seongwoon Jeong*[1], Yilan Zhang[2], Sean O'Connor[2], Jerome P. Lynch[2], Hoon Sohn[3] and Kincho H. Law[1]

[1]*Department of Civil and Environmental Engineering, Stanford University, 473 Via Ortega, Stanford, CA 94305-4020, USA*
[2]*Department of Civil and Environmental Engineering, University of Michigan, 2350 Hayward St., Ann Arbor, MI 48109-2125, USA*
[3]*Department of Civil and Environmental Engineering, Korea Advanced Institute of Science and Technology, 291 Daehak-ro, Yuseong-gu, Daejeon 305-701, Republic of Korea*

(Received December 22, 2015, Revised February 12, 2016, Accepted February 17, 2016)

Abstract. Advances in sensor technologies have led to the instrumentation of sensor networks for bridge monitoring and management. For a dense sensor network, enormous amount of sensor data are collected. The data need to be managed, processed, and interpreted. Data management issues are of prime importance for a bridge management system. This paper describes a data management infrastructure for bridge monitoring applications. Specifically, NoSQL database systems such as MongoDB and Apache Cassandra are employed to handle time-series data as well the unstructured bridge information model data. Standard XML-based modeling languages such as OpenBrIM and SensorML are adopted to manage semantically meaningful data and to support interoperability. Data interoperability and integration among different components of a bridge monitoring system that includes on-site computers, a central server, local computing platforms, and mobile devices are illustrated. The data management framework is demonstrated using the data collected from the wireless sensor network installed on the Telegraph Road Bridge, Monroe, MI.

Keywords: cyber infrastructure; bridge monitoring; bridge information modeling; NoSQL database

1. Introduction

As sensor technologies mature, there have been increasing interests in the deployment of sensors for large scale infrastructure monitoring. Many bridges are now instrumented with dense sensor network to collect valuable information for management purposes (Jang *et al.* 2010, Zhou and Yi 2013, Koh *et al.* 2013). The advent of wireless sensor technologies has led to significant reduction in the installation cost of sensor network on bridge structures (Lynch and Loh 2006, Lynch *et al.* 2009). Developments of advanced nondestructive evaluation technologies have facilitated the assessment of the integrity and health of a structure by enabling the detection of the onset of damages (Sohn *et al*. 2015). With the permanent installation of sensors, recent research efforts have been attempted to extract statistically meaningful information and to apply data-driven predictive analysis with the collected long term sensor data (Cross *et al.* 2013, O'Connor *et al.*

∗Corresponding author, Graduate student, E-mail: swjeong3@stanford.edu

2014). Until now, structural health monitoring research efforts have been mostly focused on the developments of new sensor technologies and data analysis techniques. Very little efforts have been devoted to deal with the fundamental issues of data management. The data issues need to be dealt with properly in order to facilitate long term lifecycle bridge monitoring and management.

Information models and interoperability standards have been proposed to support data management platform in various engineering disciplines (Ray 2002, Cheng *et al.* 2010). In the building and construction domains, building information modeling (BIM) has been widely employed to support integrated project delivery process and data exchange (Eastman *et al.* 2011). The development of BIM standard has enabled software to support data exchange among different application platforms. Research efforts have also been initiated towards developing bridge information modeling (BrIM) standards for bridge structures (Chen and Shirolé 2006, Shirolé *et al.* 2008, Samec *et al.* 2014). Current BrIM efforts focus primarily on the geometric information and material properties (Karaman *et al.* 2013, Ali *et al.* 2014). Standard markup language, such as XML, is employed as the modeling language to facilitate data interoperability. In order to be useful for comprehensive bridge lifecycle management, BrIM needs to be extended to include descriptions of sensor data and integrated with a bridge monitoring system.

One conventional approach to handle sensor data in structural monitoring applications is to employ traditional relational database management systems (RDBMS). The key advantages of RDBMSs are their reliability, convenient query language, and the extensive user base. For example, Smarsly *et al.* (2013) have proposed a cyber infrastructure for wind turbine monitoring using MySQL database, and Li *et al.* (2006) have utilized SQL Server 2000 for health monitoring system for the Shandong Binzhou Yellow River highway bridge. However, recent studies have identified the limitations of RDBMSs, in particular, for the scalability and flexibility issues (Hecht and Jablonski 2011, Han *et al.* 2011, Padhy *et al*. 2011). With the amount of data collected from a dense sensor array, using RDBMS as a backend database for a bridge monitoring system is neither efficient nor desirable. Furthermore, the basic data structure for schema representation in RDBMS as tables is inefficient to handle the BrIM and XML-based schemas, which typically involve hierarchical and unstructured data structure.

Advances in cyber physical systems and cloud computing services share many significant technologies that can be deployed for the management of infrastructure monitoring data. Cloud computing can be broadly defined as a utility over a network model that has emerged as a cost-effective and efficient model to enable and deliver business and engineering services (Law *et al.* 2016). Driven by the need for storing, managing and retrieving large online data records with heterogeneous formats, much research have been devoted to develop non-relational database and non-traditional file management systems. Examples of open source databases that have been deployed by cloud service providers include Apache Cassandra, Apache H-Base and MongoDB (Grolinger *et al.* 2013). These non-traditional database systems are noted as NoSQL (Not only SQL) database systems which are designed to handle unstructured data, which are the types of data commonly found in engineering models and structural monitoring systems. Recent studies have shown that NoSQL database systems have significant advantages over RDBMS in terms of flexibility and scalability (Hecht and Jablonski 2011, Han *et al.* 2011, Padhy *et al.* 2011). For example, Le *et al.* (2014) proposed an Internet of Things (IoT) platform to handle the data collected by sensors and concluded that NoSQL database systems, such as Apache Cassandra, consistently have better performance than relational database systems for handling and managing sensor data. Furthermore, NoSQL database systems have been shown to have better scalability in handling massive IoT data and have better query performance for sensor network data (Li *et al.*

2012, Thantriwatte and Keppetiyagama 2011)

This study investigates a NoSQL data management framework which is designed for bridge monitoring applications. The system is designed not only to support the management of bridge monitoring data but also to facilitate data utilization by engineering design and analysis platforms. Based on the needs of the data management framework, Apache Cassandra and MongoDB are selected as the backend database systems to support pertinent data archiving and efficient querying. For interoperability purpose, we adopt OpenBrIM 2.0 (an open source XML based BrIM schema) to represent the bridge information and SensorML (a standard for sensor and IoT applications) to describe sensor information. In addition, software tools and interfaces are developed to support automated data flow and to enhance data interoperability. To demonstrate software integration, external analysis modules such as structural analysis and machine learning modules are employed. Lastly, a mobile interface is developed to allow users to easily access information stored in the database and to retrieve meaningful information from the server. The NoSQL database management system is demonstrated using the bridge information model and the monitoring data of the Telegraph Road Bridge (TRB) in Monroe, Michigan.

2. Selection of data management tools

This section discusses a sensor data management framework and the selection of data standards and the data management tools. There exist many NoSQL database systems, each has its own strengths and disadvantages. Careful evaluation of the tools is necessary for successful development of a data management system. Furthermore, use of standard modeling languages to store the metadata is important to facilitate interoperability of managed information. In this section, we first describe the overall data management system infrastructure for bridge monitoring applications. NoSQL database tools are then selected based on the defined requirements. Lastly, open standards for bridge information modeling and engineering applications are introduced to store the metadata of the system.

2.1 Sensor data management system framework

There have been few research efforts focusing on the data management infrastructure for structural monitoring (McNeill 2009, Zhang *et al*. 2012, Smarsly *et al*. 2013, Law *et al*. 2014). As shown in Fig. 1, a typical data management system for infrastructure monitoring consists of four main components: (1) onsite computers, (2) main (data repository) server, (3) local (desktop) computers, (4) and web or mobile user interfaces. The role for each of the components can be described as follows:

- An onsite computer is an autonomous in situ system that stores sensor data temporarily and serves as a buffer between the sensor network and the main server. If necessary, the onsite computer also performs pre-process of raw data or simple analysis. Once the measured sensor data is transmitted from the sensor network installed on bridge structure, the onsite computer stores the data to its file or data management system and sends the data to the main server.
- The main server plays a pivotal role for a bridge monitoring system: the main server not only persistently stores all the sensor data, the analysis results, and other metadata including bridge information model and sensor information, but also allows local desktop computers or end-users to easily access the database and to retrieve the data. Therefore, the main server

needs to adopt a database system which is scalable and flexible to handle the amount of the data which are continuously acquired from the sensor network. Furthermore, the system should adhere to the standard data structure commonly used to represent engineering models and sensor data and to facilitate easy data exchange and utilization.

- A local (desktop) computer serves as a computing platform that engineers employ to carry out the computational tasks involved in the bridge monitoring and management system. While the role of the main server is to maintain its desirable performance and stability as a centralized data archive, a local desktop computing platform periodically retrieves sensor data along with relevant metadata from the server, performs analysis, and sends the analysis results back to the main server.
- Finally, a user interface allows the mobile users or engineers direct, real time access to the computational tools as well as the in-situ information at the bridge site a via web-interface or a mobile device.

According to the data requirements, the main data server will potentially handle significant amount of data records, which are not necessarily homogeneous or of the same data types. Therefore, the backend database for the main server should primarily focus on flexibility and scalability that would allow long term data archival and extendibility that will support multi-tier service developments. On the other hand, an onsite computer or a local desktop computer only needs to temporarily store a limited amount of data. Therefore, the focuses of the database system for an onsite or a local desktop computer are not necessarily related to the long-term archiving of large amount of data, but should be on efficient data retrieval to support data parsing and analysis.

2.2 Selection of NoSQL database tools

There are many existing NoSQL database systems with different features and properties. Since NoSQL database tools have been developed to support specific data types required by the applications, selecting an appropriate database tool for specific application is very important for successful deployment of data management system (Hecht and Jablonski 2011). Based on the data types, current NoSQL database tools can be categorized into column family stores, document-oriented stores, key-value stores, and graph databases (Hecht and Jablonski 2011, Han *et al.* 2011, Padhy *et al*. 2011).

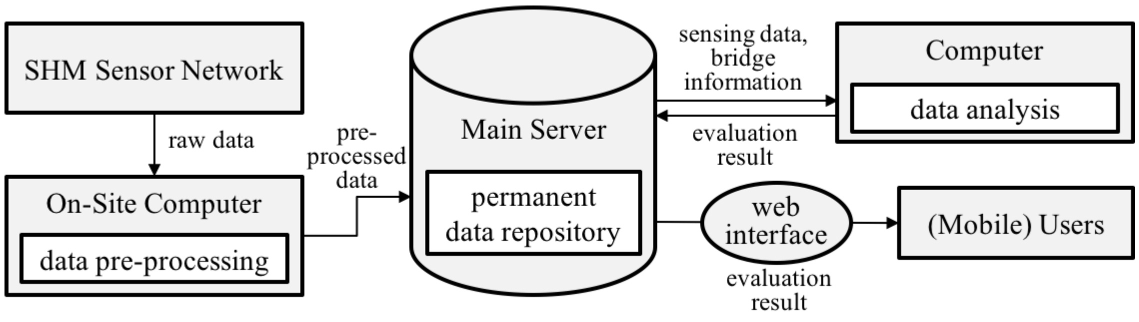


Fig. 1 Data management system for infrastructure monitoring

- The column family databases have the advantages for large scale distributed data storage.
- The document oriented databases support schema-less data structure and powerful query performance for heterogeneous data format.
- The key-value stores show very fast read and write speed utilizing in-memory operation.
- A graph database is optimized to manage data records that can be represented as a graph data structure.

In this study, we employ Apache Cassandra, a column family database to satisfy the data requirement of the main server, and MongoDB, a document oriented database to satisfy the data requirement of the onsite computer and the local computer. Key-value stores, while suitable for efficient data retrieval, are ruled out in this study, because of their limited data capacity. Lastly, the data schemas, to be described in the latter section, do not lend themselves suitable for the graph database.

2.2.1 Apache Cassandra: Database system for supporting persistent archiving

Apache Cassandra database, one of the most popular column family data storage systems, has been developed and utilized to support large scale management and data processing systems (Hewitt 2010, Hecht and Jablonski 2011). The fundamental data structure of Apache Cassandra consists of key space, column family, row, and key-value pairs. Although Apache Cassandra does not support all the functionalities of RDBMSs, Apache Cassandra is able to handle many of the emergent big data issues. For example, Apache Cassandra database system shows not only consistent performance regardless of the size of the data, but also fast performance based on hash algorithm and efficient write operation (Hewitt 2010, Hecht and Jablonski 2011, Le *et al*. 2014). Moreover, the system is highly available by guaranteeing failure at any single point would not cause total system failure (Hewitt 2010). On the other hand, Apache Cassandra currently supports only limited query and data aggregation.

Furthermore, the flexible data schema of Apache Cassandra has the advantages on storing heterogeneous data by allowing different attribute sets for different rows (Hewitt 2010). In the bridge monitoring applications, bridge metadata such as bridge information model and sensor information usually involves hierarchical and heterogeneous data, respectively. The flexible data schema feature of Apache Cassandra is particularly useful for managing metadata for bridge monitoring. As an example, in the building and construction application, Cheng and Das (2013) have implemented the BIM-PDE server using Apache Cassandra.

Because of availability, scalability and schema flexibility, many organizations have shifted to Cassandra NoSQL database system to manage high volume of data (read and write) transactions (Branson 2014, Datastax 2011, Datastax 2012). In this study, we employ Apache Cassandra to support long term data archival and system extendibility in the main server.

2.2.2 MongoDB: Database system for supporting efficient data retrieval

MongoDB is another popular document oriented database systems designed for schema-less data structure with high performance and scalability. The data structure of MongoDB consists of the database, collection, and binary JSON (BSON) schema-less documents (Chodorow 2013). The JSON document enables easy change or extension of the data model and human-understandable data structure such as object-oriented data format. MongoDB also has the advantages on representing complex data structure by enabling relationships between documents and supporting hierarchical data structure. Moreover, MongoDB dramatically improves read and write performances at the cost of join operation and transactions (Chodorow 2013). Although MongoDB does not support some of query and aggregation functions of RDBMSs, it still supports a rich set of query operations including indexing, range query, and aggregation operations. With the flexible schema and high performance, MongoDB is particularly suitable when expensive queries and transactions are not required.

Based on its flexibility, performance and scalability, MongoDB has been widely used in many fields including Internet of Things (IoT) applications and real-time analysis (Chodorow 2013, Hows *et al*. 2014). To support flexible data schema and high query performance, the database management system for bridge management employs MongoDB for onsite computers and local engineers' desktop computers.

2.3 Selection of standardized modeling language

Information models and interoperability standards have been proposed as a means to support integrated project delivery process and lifecycle management in engineering domain. By adhering to the data exchange standard, information models can be translated into different file formats for different applications in a seamless manner, which can reduce work loads and human errors on manual file conversion (Eastman *et al*. 2011, Bernstein *et al*. 2012). There have been several research efforts to develop information modeling standards for bridge engineering applications (Chen and Shirolé 2006, Shirolé *et al.* 2008, Karaman *et al.* 2013, Ali *et al.* 2014). To facilitate interoperability, semantically meaningful languages, such as extensible markup languages (such as XML), are employed to represent the bridge model. Research efforts have also been attempted to integrate bridge management information to bridge information models (Marzouk and Hisham 2011, Samec *et al.* 2014), and these efforts show great potentials of Bridge Information Modeling to better support integrated data management for bridge monitoring.

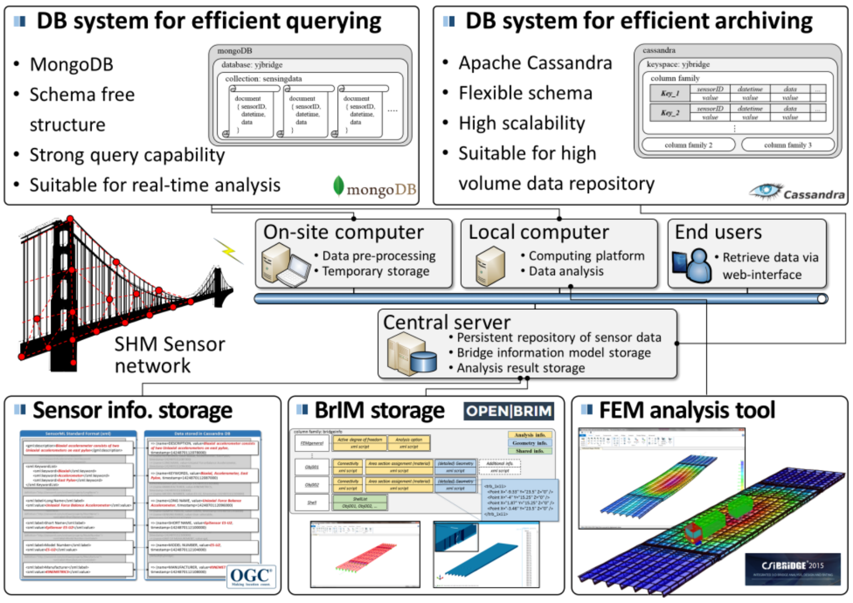


Fig. 2 One of the PLVM of the undamaged structure

In this study, we utilize the open-source XML-based OpenBrIM data schema to represent the bridge model and the relevant information (Chen 2013). OpenBrIM describes a bridge information model as a set of hierarchical objects, where an object contains information such as coordinates or material properties. OpenBrIM also allows users to define template element for parametric design. Although the BrIM model written in XML usually involves complex data structures, which are not easy to manage using traditional RDBMS, the flexible data schema of NoSQL database systems can elegantly handle the complex BrIM data.

While the current OpenBrIM schema can describe the basic elements of bridge information model, it lacks essential elements for bridge monitoring and management applications such as sensor metadata and analytical model information. Therefore, we need to supplement the database system with additional components, so that the system can manage the necessary information for bridge monitoring. To achieve this goal, we adopt Sensor Model Language (SensorML), a standard for defining measurement and post-measurement processes proposed by the Open Geospatial Consortium (OGC), to store the sensor information in the main server (Open Geospatial Consortium, 2014). SensorML is written in XML, and it provides extensive metadata for storing sensor information. In addition, we also investigate the data schema of CSI Bridge (2015) to add analytical model information to the data management system. Fig. 2 illustrates the overall data management framework reflecting the selected database tools and standard modeling languages.

3. Infrastructure system for bridge monitoring

This section describes the detailed architectural design for the bridge monitoring system based on NoSQL database tools. Data schema and interface software are developed to facilitate data utilization and data integration. We also employ several programming libraries to support remote connection to the NoSQL database systems as well as seamless data flow. In this section, first, data schema descriptions for sensor data, bridge information model, and sensor information are described. We then focus our discussion on the architecture of the proposed system and its individual components.

3.1 Data schema description

An appropriate data schema can significantly facilitate system automation and improve data management efficiency. It should be noted that with NoSQL database, the defined data schema can be easily revised and scaled according to user needs. There are three basic types of data in the monitoring system: sensor data, sensor information, and bridge information model. The analysis results can share the same schema for the sensor data. In the proposed system framework, MongoDB installed on an onsite computer and a local computer requires data schema for the sensor data, while Apache Cassandra requires data schema for sensor information and bridge information model in addition to the sensor data. The standard modeling languages such as OpenBrIM and SensorML are employed to define the data schema for interoperability.

3.1.1 Sensor data (MongoDB)

Fig. 3 describes the data schema for sensor data defined for MongoDB. In the current implementation, the database is named after the bridge structure. In addition, we use a single collection named *repos* to manage the sensor data. We take advantage of MongoDB's hierarchical data structure to categorize sensor data for ease of data retrieval (Jeong *et al.* 2015b). The root node for a single data acquisition (DAQ), named *daqevent*, contains the timestamp of the DAQ event. The non-leaf nodes, named *group* and *sensor*, not only categorize sensor data according to user defined sensor group and sensor id/channel, but also provide metadata of DAQ such as sampling rate. The leaf document named *sensordata* collects a list of measured data over a certain time period along with the timestamp. Currently, the interface program is tuned to allow each document to store the sensor data measured over a period of one second (Jeong *et al.* 2015a). For example, if the sampling rate of a sensor is 5Hz, then the measured data is discretized into buckets where each bucket has five consecutive data and is stored in a single *sensordata* document. Since the upper limit of data size of a document is 16MB, this discretization strategy is required to prevent exceeding the maximum data size which can be caused by sensors that have high sampling rate (Jeong *et al.* 2015a).

3.1.2 Sensor data (Apache Cassandra)

Fig. 4 shows the data schema defined for sensor data in Apache Cassandra. In the current implementation, the key space is named after the bridge structure, and the column family is named *sensordata*. While the consistent hashing algorithm of Apache Cassandra has great advantages on managing big data with distributed computing nodes, the partitioning strategy could deteriorate the query performance for sequential data by distributing them to different physical locations. To deal with this problem, we implement a time-series data modeling scheme for Apache Cassandra (McFadin 2015). As shown in Fig. 4, the row key is defined according to the sensor id and the year and month of timestamp in the form of *sensorID|yyyymm*. Furthermore, the timestamp of the sensor data is used for the name of a column, while the corresponding sensor data is used for the value of the column. With this time-series modeling scheme, the time-series data can be stored sequentially to disk in sorted order, thereby enhancing range query efficiency. Currently, the interface program is tuned to make each column to store up to one second of time-series data acquired by a single sensor channel.

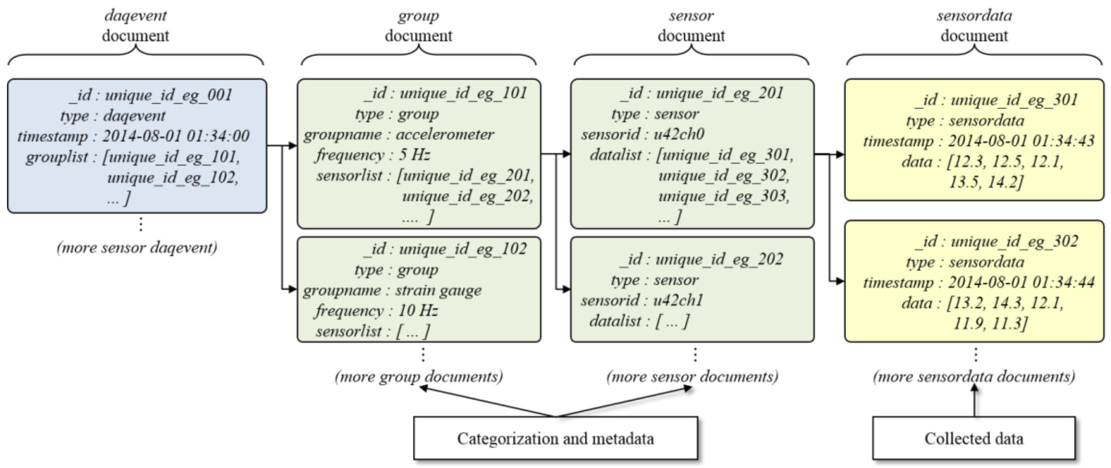


Fig. 3 Data schema of sensor data on MongoDB

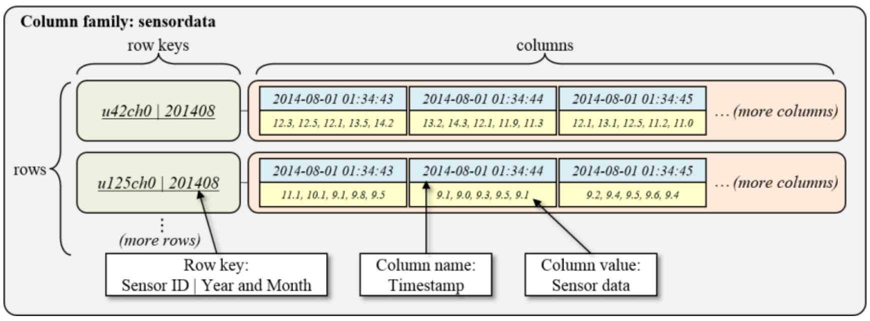


Fig. 4 Data schema of sensor data on Apache Cassandra

3.1.3 Sensor information

The main server also manages the sensor information such as sensor id, sampling rate, and output type, and allows user to utilize those information by different applications including data analysis and management of sensor. For interoperability, we implement SensorML, a standard for IoT applications, and define a list of metadata for bridge monitoring applications (Open Geospatial Consortium 2014). To manage sensor information, a column family named *sensorinformation* is prepared in Apache Cassandra. Fig. 5 illustrates the data schema defined for the sensor data. A single row is assigned to store a single sensor's metadata. The primary key that uniquely identifies a sensor consists of the sensor id and the installation date of the sensor, since there could be different sensors sharing the same sensor id over time. In addition, index is defined on the output of a sensor (e.g., strain, acceleration, and temperature), since same type of sensors are often utilized together. Although the sensor information is typically heterogeneous, the flexible data schema of Apache Cassandra can handle the unstructured information elegantly. For example, the sensor information of *u42ch0* in Fig. 5 contains incomplete data set due to the lack of *output_uom* entity. While traditional relational database systems enforce identical set of attributes to every single row, the flexible data schema of Cassandra allows different attribute set for each row, and thus, elegantly handles incomplete data sets that do not contain all the components defined by SensorML (Hecht and Jablonski 2011, Hewitt 2010).

3.1.4 Bridge information model

Bridge information model repository in the main server stores all the information about a bridge structure including, but not limited to, geometric and analytic model information. For example, an element in a bridge information model includes not only the detailed coordinate information (geometric information), but also the connectivity and load information (analytic model information). This study employs OpenBrIM 2.0 schema by Chen *et al*. (2013) as the basis for the data schema representing the bridge information model. OpenBrIM uses XML as the language-neutral data format to facilitate data exchange and improve interoperability. Since the OpenBrIM lacks schema for analytical model, we investigate the data schema of CSI Bridge (2015), a finite element (FE) analysis program, and extracts the important elements that are needed for structural analysis.

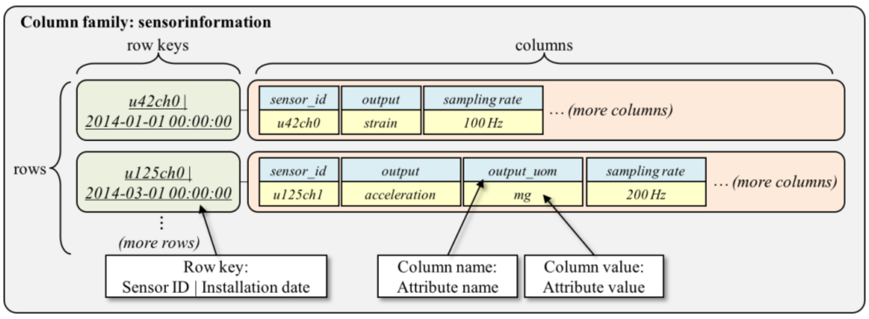

Column family: sensorinformation
row keys
columns
rows
u42ch0 | 2014-01-01 00:00:00
sensor_id
output
sampling rate
u42ch0
strain
100 Hz
… (more columns)
u125ch0 | 2014-03-01 00:00:00
sensor_id
output
output_uom
sampling rate
u125ch1
acceleration
mg
200 Hz
… (more columns)
(more rows)
Row key: Sensor ID | Installation date
Column name: Attribute name
Column value: Attribute value

Fig. 5 Data schema of sensor information on Apache Cassandra

Fig. 6 shows the data schema for BrIM repository in the Apache Cassandra database. Bridge information model data is stored in a column family named *bridgeinformation*. Most of the rows in the column family represent a unit element of the bridge information model, and their row keys are defined according to the id of the elements. Each row stores the element's attributes such as geometry and nodal connectivity in separate columns. The value of each column is a XML string containing appropriate information based on OpenBrIM schema. Some special rows contain general information for FE analysis including node information, material properties, and load information. Although a bridge information model usually involves objects that may have different attributes-value pairs, their heterogeneous object sets can be handled rather flexibly within a single column family with Apache Cassandra (instead of using multiple relational tables) (Hewitt 2010).

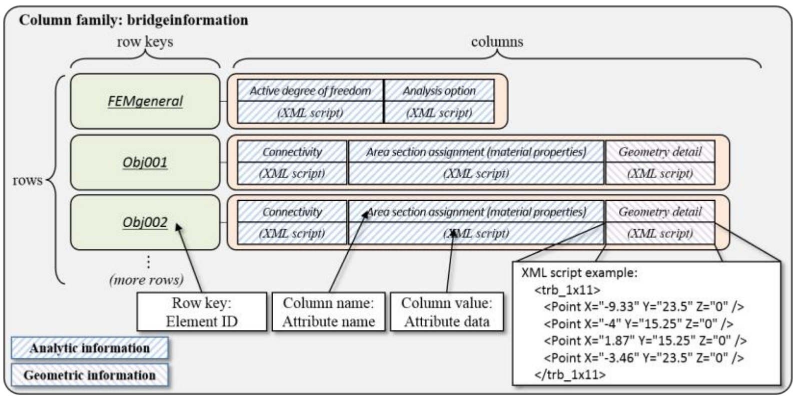
3.2 System architecture

Fig. 7 shows the overall architecture of the bridge monitoring system. As described in Section 2, the system consists of four major components including onsite computers, main server, local computers, and user interfaces. Interface software tools are developed to support many functions including data processing, network handling, and connection to the database systems. In addition, the interface software for each component is developed to enable seamless data flow. Various Application Programming Interface (API) tools are available for implementation. For example, MongoDB provides APIs supporting many programming languages to help the users to easily utilize the database system. Similarly, Apache Cassandra provides convenient APIs as well as a Cassandra Query Language (CQL), which is very similar to the structured query language (SQL). In this study, Python is chosen as the primary programming language to implement the functions needed for the proposed system including data processing and data transmission.

3.2.1 Onsite computer

An onsite computer receives the sensor data from sensor network, stores the data in MongoDB, and sends the data to the Apache Cassandra database in the main server. For this study, we employ an older version of MongoDB (version 2.0.6) since the onsite computers and controllers installed in some bridge monitoring and sensor network systems employ older versions of the Microsoft operating system and do not support a recent version (version 2.2 or higher) of the MongoDB system. Two interface programs written in Python are developed to automate the data flow (Jeong *et al*. 2015b). The first program, named *onsite.py*, is in charge of sending new sensor data to the MongoDB's repository. Once the new sensor data is transmitted from a sensor network, *onsite.py* parses the raw sensor data into the defined data schema and is stored in the database using PyMongo, a MongoDB API for Python.

Similarly, the second program, named *tomain.py*, parses the sensor data in the MongoDB into the defined data schema for Apache Cassandra and sends the data to the main server. The *tomain.py* employs an API for Apache Cassandra called Cassandra Driver. Since the in situ condition is not necessarily stable, we implement error handlers that can handle errors due to unstable network connection. In addition, bridge monitoring system typically involves large amount of sensor data, even though the network speed on site is typically slow in comparison to the data rate. To handle a timeout error due to slow network connection, we loosen the connection timeout constraint of Cassandra Driver. The second program also records whether a data bucket has been successfully sent to the main server, so that we can prevent unnecessary duplicate data transmission to the Cassandra database when the onsite system is accidently rebooted.



```
<trb_1x11>
  <Point X="-9.33" Y="23.5" Z="0" />
  <Point X="-4" Y="15.25" Z="0" />
  <Point X="1.87" Y="15.25" Z="0" />
  <Point X="-3.46" Y="23.5" Z="0" />
</trb_1x11>
```

Fig. 6 Data schema of bridge information model on Apache Cassandra

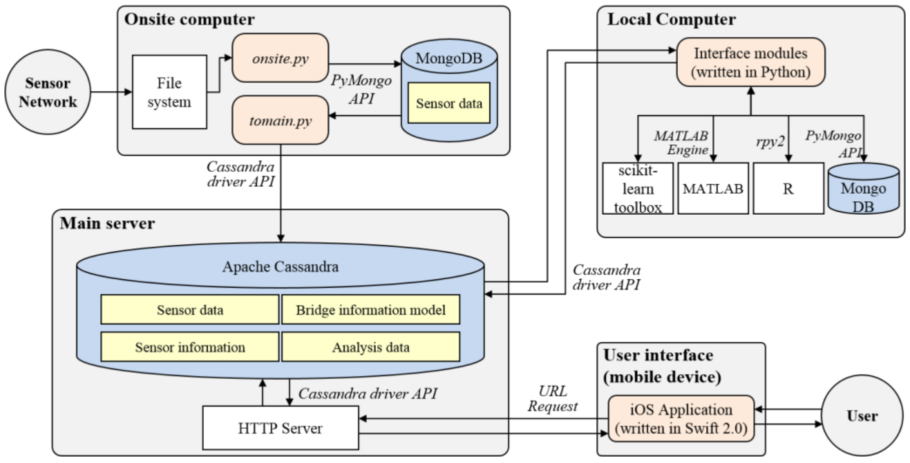


Fig. 7 System architecture of cyber infrastructure

3.2.2 Main server

The main server serves a central data repository of the bridge monitoring system. Apache Cassandra (version 2.0.16) is implemented as the backend database for the main server. Apache Cassandra in the main server is designed to store mainly four kinds of data such as sensor data, sensor information, bridge information model, and analysis result. Apache Cassandra keeps listening to the request from onsite computers and local computers through allocated ports on the network. Once a request from Cassandra Driver API is delivered, Apache Cassandra automatically handles the request and sends the appropriate response back to the device. Since the size of sensor data is usually quite large, the setting of Cassandra is tuned to use as much as memory as possible for efficient data processing.

In addition to the Cassandra database, the main server also implements HTTP server to handle requests from the users using BaseHTTPServer, a Python module for Internet protocol. The HTTP server keeps listening to the user request from a user’s mobile device. Once a URL request is received, the HTTP server parses the URL into query and parameter, retrieves relevant data from Apache Cassandra database using Cassandra Driver, and returns the data to the user. Currently, the HTTP server only supports simple GET requests to query sensor data and sensor information.

3.2.3 Local computer

A local computer is essentially a desktop-based computing platform that retrieves data, perform analysis, and push the analysis results back to the Apache Cassandra in the main server. Since some data analysis modules require very expensive computational costs, the decentralized strategy helps the main server to be isolated from such operations and to maintain its performance as the central data repository. To automate data flow from Apache Cassandra database in the main server to the analysis software in the local computer, several interface programs written in Python are developed (Jeong *et al*. 2015b). In addition to Cassandra Driver, we use MATLAB Engine (an interface between MATLAB and Python), scikit-learn (a tool for machine learning in Python), and rpy2 (an interface to R for Python process) to demonstrate a diverse set of data analysis platforms. In addition, the local computer also employs MongoDB in case the user wants to temporarily store the data in the local computer.

3.2.4 User interface

Development of user interface is an important task to facilitate the utilization of bridge monitoring data for bridge management and decision making processes. In this study, we develop a prototype iOS application using Swift 2, a programming language for iOS. This application deploys many Swift APIs including view controller (UIViewController), button (UIButton), table (UITableView) and map view (UIMapView). The application provides functions to retrieve sensor data and sensor information from the main server. For example, if a user touches “sensor information” button on the screen, the application sends a GET request to the HTTP server in the main server using the networking API of iOS. The HTTP server then processes the request and return relevant data to the user’s mobile device. Once the mobile device receives the data, the application displays the retrieved sensor information as tables for viewing. Currently, the prototype application supports simple data retrieval for sensor data and sensor information.

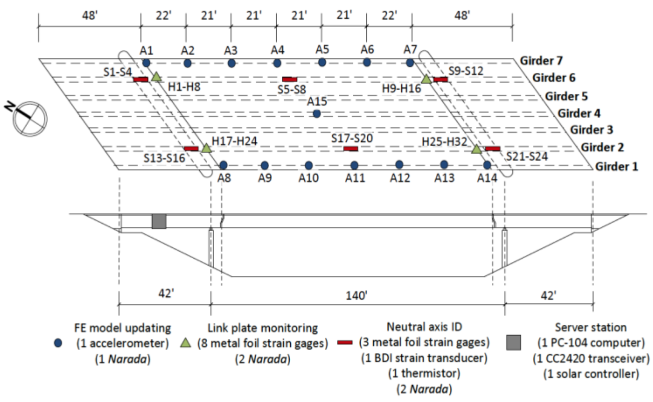
4. Implementation

To test the data management infrastructure, we use the sensor data sets collected from Telegraph Road Bridge (TRB) in Monroe, Michigan (shown in Fig. 8 (a)) and its bridge model (modelled in CSI Bridge (2015)) and sensor information (previously stored in Microsoft Excel). The sensor network installed on the Telegraph Road Bridge consists of 14 accelerometers, 40 strain gauges, and 6 thermistors, as described by O'Connor et al. (2014, 2015). Fig. 8(b) shows the layout of the sensor network (O'Connor *et al*. 2015). The data sets include seven weeks of sensor data: one week per month from August 2014 to February 2015. The sensor network acquires data for a one-minute time duration on every 2 hours interval. While the accelerometers collect the measurements at the sampling rate of 200 Hz, the sampling rate of the strain gauges and the thermistors is set at 100 Hz.

4.1 Data management: storage and retrieval

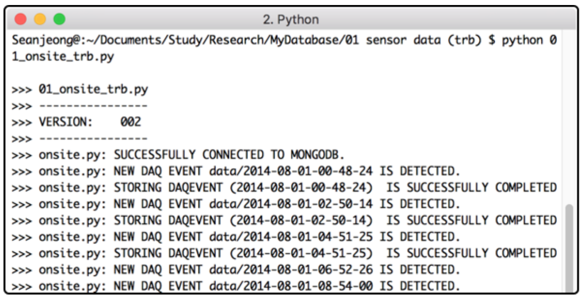
To simulate the in situ bridge monitoring scenario, a script written in Python that periodically sends the sensor data sets to the onsite computer. As discussed in Section 3, once the sensor data is delivered to the onsite computer, the interface program (*onsite.py*) on the onsite computer automatically re-structures the raw data according to the defined data schema for MongoDB and stores the parsed data to MongoDB. Fig. 9 shows a screenshot of the *onsite.py* in operation. Once a data set for a single data acquisition event is stored in MongoDB in the onsite computer, another interface program on the onsite computer (*tomain.py*) parses the data set stored in MongoDB to the defined data schema for Apache Cassandra and uploads the data to the Apache Cassandra database in the main server. Fig. 10 shows a screenshot of the *tomain.py* in operation. The sensor data stored in Apache Cassandra in the main server can be retrieved using Cassandra Driver (for local computer) or URL request (for mobile device). Fig. 11 shows an example of data retrieval for a desired time period using a mobile device.

(a) Side view


48'
22'
21'
21'
21'
21'
22'
48'
A1
A2
A3
A4
A5
A6
A7
S1-S4
H1-H8
S5-S8
A15
H9-H16
S9-S12
Girder 7
Girder 6
Girder 5
Girder 4
Girder 3
Girder 2
Girder 1
N
H17-H24
S17-S20
H25-H32
S13-S16
S21-S24
A8
A9
A10
A11
A12
A13
A14
42'
140'
42'
● FE model updating (1 accelerometer) (1 *Narada*)
▲ Link plate monitoring (8 metal foil strain gages) (2 *Narada*)
▬ Neutral axis ID (3 metal foil strain gages) (1 BDI strain transducer) (1 thermistor) (2 *Narada*)
■ Server station (1 PC-104 computer) (1 CC2420 transceiver) (1 solar controller)

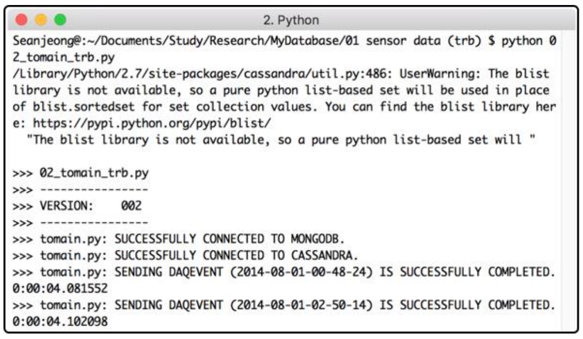
(b) Type and location of sensors installed on Telegraph Road Bridge (O'Connor *et al*. 2015)

Fig. 8 Telegraph Road Bridge, Monroe, MI



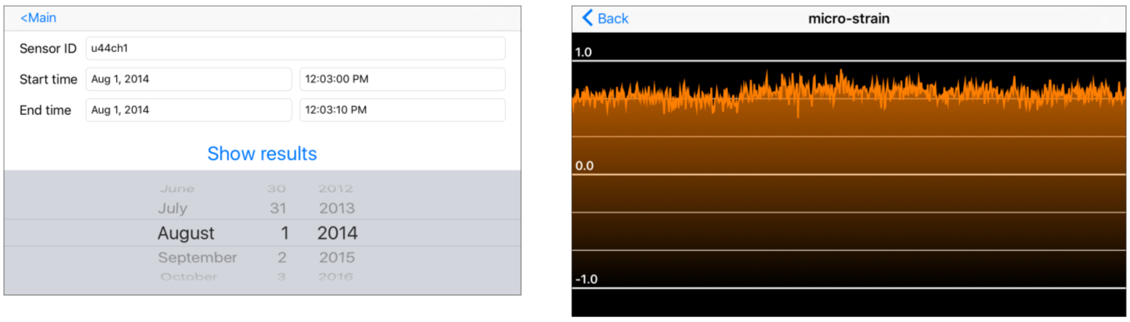
```
Seanjeong@:~/Documents/Study/Research/MyDatabase/01 sensor data (trb) $ python 0
1_onsite_trb.py

>>> 01_onsite_trb.py
>>> ----------------
>>> VERSION:    002
>>> ----------------
>>> onsite.py: SUCCESSFULLY CONNECTED TO MONGODB.
>>> onsite.py: NEW DAQ EVENT data/2014-08-01-00-48-24 IS DETECTED.
>>> onsite.py: STORING DAQEVENT (2014-08-01-00-48-24)  IS SUCCESSFULLY COMPLETED
>>> onsite.py: NEW DAQ EVENT data/2014-08-01-02-50-14 IS DETECTED.
>>> onsite.py: STORING DAQEVENT (2014-08-01-02-50-14)  IS SUCCESSFULLY COMPLETED
>>> onsite.py: NEW DAQ EVENT data/2014-08-01-04-51-25 IS DETECTED.
>>> onsite.py: STORING DAQEVENT (2014-08-01-04-51-25)  IS SUCCESSFULLY COMPLETED
>>> onsite.py: NEW DAQ EVENT data/2014-08-01-06-52-26 IS DETECTED.
>>> onsite.py: NEW DAQ EVENT data/2014-08-01-08-54-00 IS DETECTED.
```

Fig. 9 onsite.py: Python script storing sensor data to MongoDB in on-site computer



```
Seanjeong@:~/Documents/Study/Research/MyDatabase/01 sensor data (trb) $ python 0
2_tomain_trb.py
/Library/Python/2.7/site-packages/cassandra/util.py:486: UserWarning: The blist
library is not available, so a pure python list-based set will be used in place
of blist.sortedset for set collection values. You can find the blist library her
e: https://pypi.python.org/pypi/blist/
  "The blist library is not available, so a pure python list-based set will "

>>> 02_tomain_trb.py
>>> ----------------
>>> VERSION:    002
>>> ----------------
>>> tomain.py: SUCCESSFULLY CONNECTED TO MONGODB.
>>> tomain.py: SUCCESSFULLY CONNECTED TO CASSANDRA.
>>> tomain.py: SENDING DAQEVENT (2014-08-01-00-48-24) IS SUCCESSFULLY COMPLETED.
0:00:04.081552
>>> tomain.py: SENDING DAQEVENT (2014-08-01-02-50-14) IS SUCCESSFULLY COMPLETED.
0:00:04.102098
```

Fig. 10 tomain.py: Python script storing sensor data to Apache Cassandra database in main server



(a) User interface for querying (b) Retrieved sensor data

Fig. 11 Sensor data retrieved from Apache Cassandra in main server using mobile device

The bridge information model is parsed into the defined data schema and stored in the Apache Cassandra in the main server. For storing the bridge information model, we export the FE model into Microsoft Excel format using CSI Bridge's exporting function. The exported model is then converted into the defined data schema and stored in the Apache Cassandra in the main server.

Once the data are stored, the bridge information model can be retrieved in different file formats. Figs. 12(a) and 12(b) show the retrieved FE model (Excel file format) visualized using CSI Bridge

(2015) and the retrieved BrIM geometry model (XML file format) visualized using the OpenBrIM viewer, respectively. In this study, we develop and utilize Python scripts to convert, store, and retrieve the bridge information model. Similarly, sensor information also needs to be stored in the Apache Cassandra in the main server according to the defined data schema. To conduct this task, we develop a Python script to parse and send the sensor information to the main server. Fig. 13 shows the sensor information retrieved using CQLSH, a command line client for Apache Cassandra.

4.2 Data analysis using long term sensor data

To demonstrate the utilization of local computer as a computing platform (typically employed by engineers), we employ two analysis modules: modal analysis module and machine learning module. The implementation of these modules for the Telegraph Road Bridge has been previously illustrated using a cyber-enabled wireless monitoring system (O'Connor *et al.* 2014, Zhang *et al.* 2016). In this study, we employ the modules by utilizing the proposed NoSQL data management infrastructure for bridge monitoring. We utilize a Matlab-based Subspace Identification module (Overschee 2012) to extract modal properties from the acceleration data stored in the main server. The modal properties are computed on the local computer using the following steps (Jeong *et al.* 2015b)

(1) Retrieve the sensor IDs of all the accelerometers that operated during the defined period.
(2) Retrieve the sensor data collected by the sensors identified in step (1).
(3) Send the retrieved sensor data to the Matlab-based Subspace Identification module (Overschee 2012) and calculate the modal properties.
(4) Upload the calculated modal properties to Cassandra database in the main server.
(5) Retrieve the sensor IDs of all the thermistors that operated during the same period.
(6) Retrieve sensor data collected by the sensors identified in step (5).
(7) Plotting the first modal frequencies calculated in step (3) along with the temperature data acquired in step (6).

The first modal frequencies computed using sensor data collected from August 2014 to February 2015 are plotted in Fig. 14(a) along with the temperature measurements.

The analysis results stored in the Apache Cassandra in the main server can be utilized for additional analyses. We employ Gaussian Process for Machine Learning (GPML) module in the local computer to predict the effect of temperature changes on the bridge's behavior. The Gaussian Process Regression interface written in Python retrieves the first modal frequencies along with the temperature measurements from the Apache Cassandra in the main server. The retrieved data sets are then sent to the GPML module provided by the scikit-learn (a Python-based machine learning) package (Pedregosa *et al.* 2011). Once the GPML module completes the analysis, the prediction results for the first modal frequencies for different temperatures are returned. Fig. 14(b) illustrates the prediction results for the natural frequencies based on the sensor data collected from August 2014 to February 2015. The predicted first natural frequencies show the bilinear relationship between temperature and the frequency with the pivot at 0℃; the results are in good agreements with the study by O'Connor *et al.* (2014).

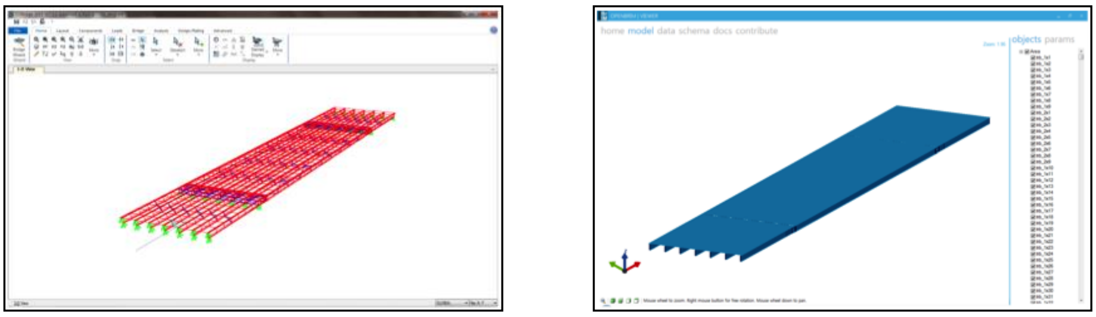
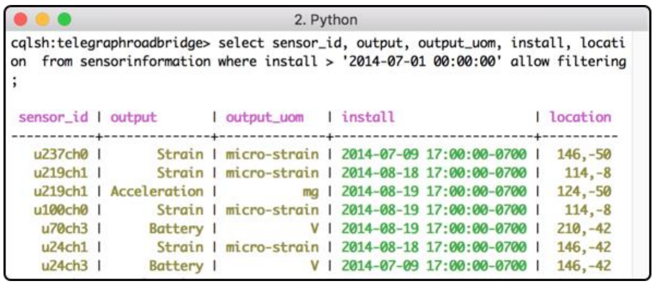

(a) Finite element model (b) Bridge information model (geometry)

Fig. 12 Bridge information model retrieved from Apache Cassandra in the main server

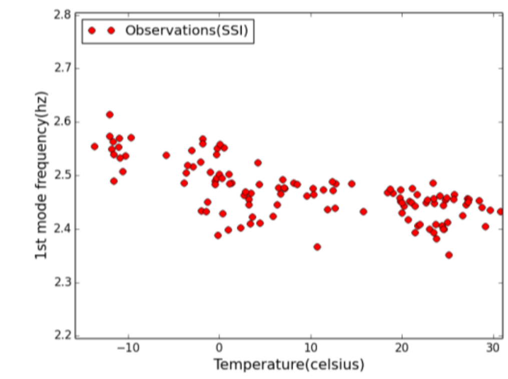

```
cqlsh:telegraphroadbridge> select sensor_id, output, output_uom, install, locati
on  from sensorinformation where install > '2014-07-01 00:00:00' allow filtering
;

 sensor_id | output       | output_uom   | install                  | location
-----------+--------------+--------------+--------------------------+----------
   u237ch0 |       Strain | micro-strain | 2014-07-09 17:00:00-0700 |   146,-50
   u219ch1 |       Strain | micro-strain | 2014-08-18 17:00:00-0700 |    114,-8
   u219ch1 | Acceleration |           mg | 2014-08-19 17:00:00-0700 |   124,-50
   u100ch0 |       Strain | micro-strain | 2014-08-19 17:00:00-0700 |    114,-8
    u70ch3 |      Battery |            V | 2014-08-19 17:00:00-0700 |   210,-42
    u24ch1 |       Strain | micro-strain | 2014-08-18 17:00:00-0700 |   146,-42
    u24ch3 |      Battery |            V | 2014-07-09 17:00:00-0700 |   146,-42
```
Fig. 13 Sensor information retrieved from Apache Cassandra in the main server

4.3 Influence line analysis using sensor data and bridge model

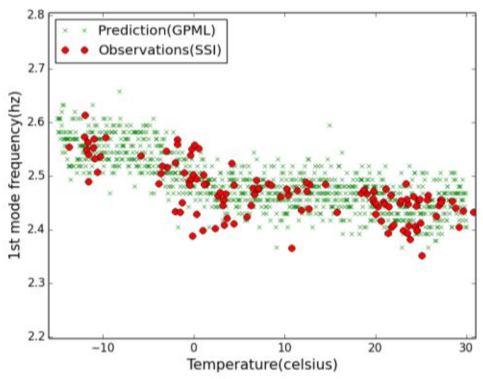
To take advantage of the integrated bridge monitoring infrastructure, we conduct influence line analysis, which compares bridge responses collected by the sensors with analytically computed response using the FE model. In this analysis, we utilize sensor data collected from a field test for identification of vehicle-bridge interaction (Hou *et al*. 2015). In the dynamic loading test, a single test truck instrumented with GPS sensor crosses the Telegraph Road Bridge without other traffics. The test truck passes the middle lane of the bridge at approximately 60 mph. The specification of the test truck can be found in Fig. 15 and Table 1 (Hou *et al*. 2015). When the test truck crosses the bridge, strain gauges (installed as described in Fig. 16) measure the dynamic strain response of the bridge (Hou *et al*. 2015), and the collected data sets are stored in the Apache Cassandra in the main server. On the other hand, the corresponding vehicle load and vehicle lane are defined in the FE model of the Telegraph Road Bridge (as shown in Fig. 17) for the simulation. The FE model is then sent to the Apache Cassandra in the main server.

Table 1 Test truck load description (unit: pound) (Hou et al. 2015)

Steer Axle	Drive Axle	Trailer Lead Axle	Trailer Rear Axle	Total
9,460	17,620	17,820	17,600	62,500



(a) 1st modal frequency along with temperature measurement (SSI module)



(b) Prediction of 1st modal frequency according to change of temperature (GPML module)

Fig. 14 Data analysis result computed by using SSI module and GPML module in local computer

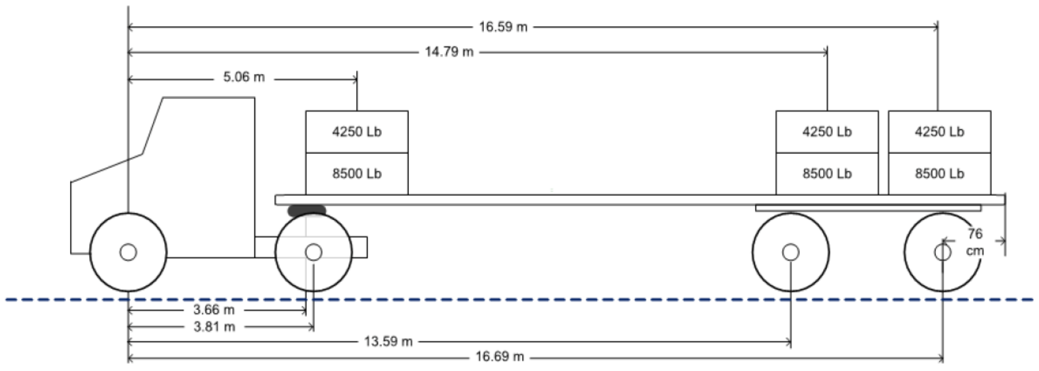
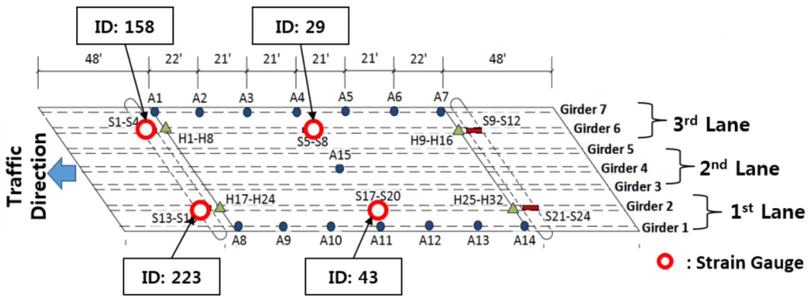
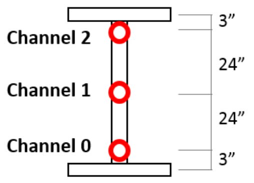


Fig. 15 Test truck dimension (Hou *et al.* 2015)

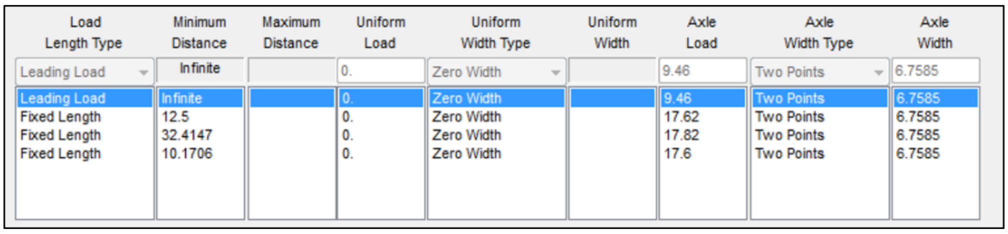


(a) Plan view



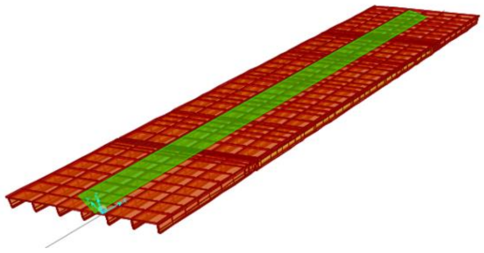
(b) Section view

Fig. 16 Location of strain gauges

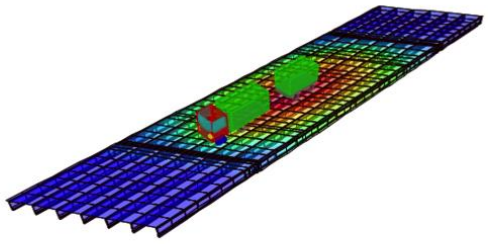


Load Length Type	Minimum Distance	Maximum Distance	Uniform Load	Uniform Width Type	Uniform Width	Axle Load	Axle Width Type	Axle Width
Leading Load	Infinite		0.	Zero Width		9.46	Two Points	6.7585
Leading Load	Infinite		0.	Zero Width		9.46	Two Points	6.7585
Fixed Length	12.5		0.	Zero Width		17.62	Two Points	6.7585
Fixed Length	32.4147		0.	Zero Width		17.82	Two Points	6.7585
Fixed Length	10.1706		0.	Zero Width		17.6	Two Points	6.7585

(a) Vehicle load configuration



(b) Defined vehicle lane



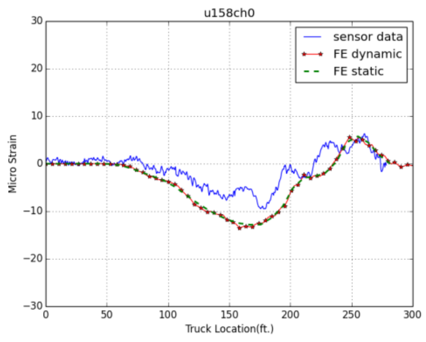
(c) Visualized test truck model

Fig. 17 Test truck defined in FE model (CSI Bridge (2015))

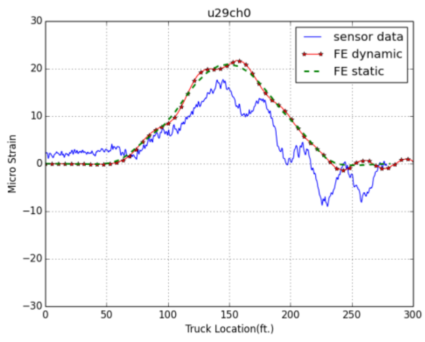
Once all the necessary data are stored in the main server, we plot the influence lines for sensor data by retrieving the collected sensor data and plotting the strain response along with the truck location. Next, we download the FE model from the server and conduct static and dynamic FE analysis to compute the influence line for the sensor locations. Regarding the FE analysis, direct integration method (Hilber-Hughes-Taylor method) without damping is employed, and 0.03 second is selected for the time step for the integration. Furthermore, since strain cannot be directly obtained from analysis results of CSI Bridge, we calculate the strain indirectly using the stress response obtained from the analysis. Finally, we compare the measured response and analytical response of the bridge by overlaying the obtained influence lines. Figs. 18(a), 18(b), 18(c), and 18(d) show the overlays of the influence lines at a sensor location (channel 0), respectively. The results show that the measured response is very similar with the analytical response, although the analytical response shows slightly higher maximum response than the measured response.

5. Conclusions

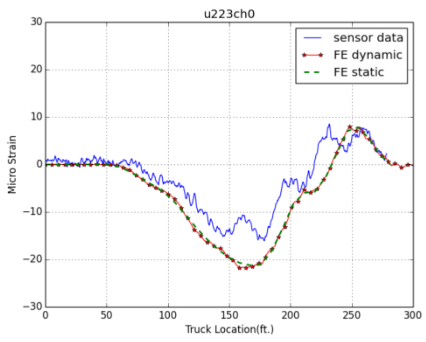
In this study, a cyber infrastructure for bridge monitoring applications based on state-of-the-art data management technologies is developed. First, we investigate the bridge monitoring framework and define the data requirements including flexibility, scalability and query performance. Based on the data requirements, Apache Cassandra and MongoDB are selected as the backend database systems of the cyber bridge monitoring framework. Apache Cassandra is a column family database that is suitable for large-scale distributed database, while MongoDB is a document oriented database that has advantages on the schema-less data structure and fast performance. In addition, standardized modelling languages such as OpenBrIM and SensorML are employed to handle unstructured data and to support interoperability.



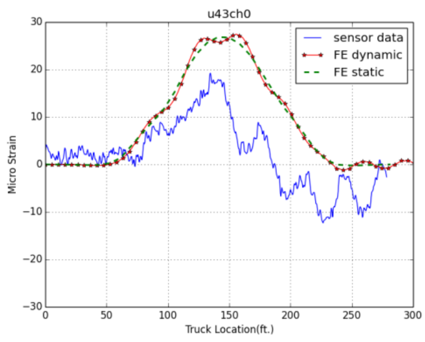
(a) strain gauge at end-span of girder 6



(b) strain gauge at mid-span of girder 6



(c) strain gauge at end-span of girder 2



(d) strain gauge at mid-span of girder 2

Fig. 18 Influence line analysis result

In the current proposed data management system, the monitoring system architecture consists of onsite computer, main server, local computer, and mobile interface. Data schemas for sensor data, sensor information and bridge information model are designed to facilitate system automation and to improve data management performance. The data schema for sensor data of MongoDB is defined using hierarchical data structure for ease of data access. On the other hand, the data schema for sensor data of Apache Cassandra is defined using time-series data modelling scheme to guarantee desirable performance for time-series data. The data schemas for bridge information model and sensor information are defined based on OpenBrIM and SensorML standards, respectively. In addition, we have developed interface programs to better support the automation and integration of the system.

The data management system is validated using the sensor data collected from the Telegraph Road Bridge along with the bridge model and sensor information of the bridge. Results show that the proposed cyber bridge monitoring infrastructure can handle storage and retrieval of the data efficiently and stably. To demonstrate the utilization of the cyber infrastructure, modal analysis module and machine learning modules are employed. Furthermore, we compare the influence lines obtained from the sensor data and bridge information with the one obtained by analysis. The results show that the proposed system can help users easily query and utilize complex and large data for bridge monitoring applications.

Acknowledgments

This research is supported by a Grant No. 13SCIPA01 from Smart Civil Infrastructure Research Program funded by Ministry of Land, Infrastructure and Transport (MOLIT) of Korea government and Korea Agency for Infrastructure Technology Advancement (KAIA). The research is also partially supported by the US National Science Foundation (NSF), Grant No. ECCS-1446330 to Stanford University and Grant No. CMMI-1362513 and ECCS-1446521 to the University of Michigan. The authors thank the Michigan Department of Transportation (MDOT) for access to the Telegraph Road Bridge and for offering support during installation of the wireless monitoring system. Any opinions, findings, conclusions or recommendations expressed in this paper are solely those of the authors and do not necessarily reflect the views of NSF, MOLIT, KAIA or any other organizations and collaborators.

References

Ali, N., Chen, S.S., Srikonda, R. and Hu, H. (2014), "Development of concrete bridge data schema for interoperability", *Transportation Research Record*, **2406**(1), 87-97.

Bernstein, H.M., Jones, S.A. and Russo, M.A. (2012), "The business value of BIM in North America: multi-year trend analysis and user ratings (2007-2012)", SmartMarket Report, McGraw-Hill Construction, Bedford, MA.

Branson, R. (2014), "Facebook's Instagram: Making the switch to Cassandra from Redis, a 75% 'Insta' savings," [Online article], Retrieved from: http://planetcassandra.org/blog/interview/facebooks-instagram-making-the-switch-to-cassandra-from-redis-a-75-insta-savings/ (accessed on 16 Dec 2015).

Chen S.S. (2013), "Bridge Data Protocols for Interoperability Local Failure Bridge Data Protocols for Interoperability and Life Cycle Management", [Online article], Retrieved from: http://iug.buildingsmart.org/resources/itm-and-iug-meetings-2013-munich/infra-room/bridge-data-protocols-for-interoperability-and-life-cycle-management (accessed on 16 Dec 2015).

Chen, S.S. and Shirolé, A.M. (2006), "Integration of information and automation technologies in bridge engineering and management: Extending the state of the art", *Transportation Research Record*, **1976**(1), 3-12.

Cheng, J.C.P. and Das, M. (2013), "A cloud computing approach to partial exchange of BIM models", *Proceedings of the 30th CIB W78 International Conference*.

Cheng, J.C.P., Law, K.H., Bjornsson, H., Jones, A. and Sriram, R. (2010), "A service oriented framework for construction supply chain integration", *Autom. Constr.*, **19**(2), 245-260.

Chodorow, K. (2013), "MongoDB: the definitive guide", O'Reilly Media, Inc., [Online book], Retrieved from: http://proquest.safaribooksonline.com/9781449381578?uicode=stanford (accessed on 16 Dec 2015).

Cross, E.J., Koo, K.Y., Brownjohn, J.M.W. and Worden, K. (2013), "Long-term monitoring and data analysis of the Tamar Bridge", *Mech. Syst. Signal Pr.*, **35**(1-2), 16-34.

Datastax (2011), "Netflix gives users exactly what they want – every time," [Online article], Retrieved from: http://www.datastax.com/wp-content/uploads/2011/09/CS-Netflix.pdf (accessed on 16 Dec 2015).

Datastax (2012), "eBay engages customers with personalized recommendations," [Online article], Retrieved from: http://www.datastax.com/wp-content/uploads/2012/12/DataStax-CS-eBay.pdf (accessed on 16 Dec 2015).

Eastman, C., Teicholz, P., Sacks, R. and Liston, K. (2011), *BIM handbook: a guide to building information modeling for owners, managers, designers, engineers, and contractors*, John Wiley & Sons, Inc., Hoboken, NJ.

Grolinger, K., Higashino, W.A., Tiwari, A. and Capretz, M.A. (2013), "Data management in cloud environments: NoSQL and NewSQL data stores", *J. Cloud Comput.: Adv., Syst. Appl.*, **2**(1), 22.

Han, J., Haihong, E., Le, G. and Du, J. (2011), "Survey on NoSQL database", *Proceedings of the ICPCA 2011*.

Hecht, R. and Jablonski, S. (2011), "NoSQL Evaluation: A use case oriented survey", *Proceedings of CSC 2011*.

Hewitt, E. (2010), "Cassandra: the definitive guide," O'Reilly Media, Inc., [Safari Book Online], Retrieved from: http://proquest.safaribooksonline.com/9781449399764 (accessed on 16 Dec 2015).

Hou, R., Zhang, Y., O'Connor, S., Hong, Y. and Lynch, J.P. (2015), "Monitoring and identification of vehicle-bridge interaction using mobile truck-based wireless sensors", *Proceedings of the 11th International Workshop on Advanced Smart Materials and Smart Structures Technology*, August 1-2, 2015, University of Illinois, Urbana-Champaign, United States.

Hows, D., Membrey, P. and Plugge, E. (2014). "MongoDB Basics," Apress. [Online book], Retrieved from: http://link.springer.com/book/10.1007%2F978-1-4842-0895-3 (accessed on 20 Dec 2015)

Jang, S., Jo, H., Cho, S., Mechitov, K., Rice, J.A., Sim, S., Jung, H., Yun, C., Spencer, Jr., B.F. and Agha, G. (2010), "Structural health monitoring of a cable-stayed bridge using smart sensor technology: deployment and evaluation", *Smart Struct. Syst.*, **6**(5-6), 439-459.

Jeong, S., Byun, J., Kim, D., Sohn, H., Bae, I.H. and Law, K.H. (2015a), "A data management infrastructure for bridge monitoring", *Proceedings of the SPIE Smart Structures/NDE Conference*, art no. 94350P.

Jeong, S., Zhang, Y., Lynch, J., Sohn, H. and LAW, K.H. (2015b), "A NoSQL-based data management infrastructure for bridge monitoring database", *Proceedings of the Structural Health Monitoring 2015 (IWSHM 2015)*, Stanford University, Stanford, CA, USA, September 1-3, 2015.

Karaman, S.G., Chen, S.S. and Ratnagaran, B.J. (2013), "Three-dimensional parametric data exchange for curved steel bridges", *Transportation Research Record*, **2331**(1), 27-34.

Koh, H., Park, W. and Kim, H. (2013). "Recent activities on operational monitoring of long-span bridges in Korea," *Proceedings of SHMII 2013*, pp. 66-82.

Law, K.H., Cheng, J.C.P., Fruchter, R. and Sriram, R. (2016), "Cloud applications in engineering", In Encyclopedia of Cloud Computing, (Eds., Murugesan, S. and Bojanova, I.), Wiley. (in press).

Law, K.H., Smarsly, K. and Wang, Y. (2014), "Sensor data management technologies for infrastructure asset management", In Sensor Technologies for Civil Infrastructures: Applications in Structural Health Monitoring, (Eds., Wang, M.L., Lynch, J.P. and Sohn, H.), Woodhead Publishing, Cambridge, UK, **2**(1), 3-32.

Le, T.D., Kim, S.H., Nguyen, M.H., Kim, D., Shin, S.Y., Lee, K.E. and da Rosa Righi, R. (2014), "EPC information services with No-SQL datastore for the internet of things", *Proceedings of IEEE RFID 2014*.

Li, H., Ou, J., Zhao, X., Zhou, W., Li, H., Zhou, Z. and Yang, Y. (2006), "Structural health monitoring system for the Shandong Binzhou Yellow River Highway Bridge", *Comput.- Aided Civil. Infrastruct. Eng.*, **21**(4), 306-317.

Li, T., Liu, Y., Tian, Y., Shen, S. and Mao, W. (2012), "A storage solution for massive IoT data based on NoSQL", *Proceedings of the 2012 IEEE International Conference on Green Computing and Communications*.

Lynch, J.P., Kamat, V., Li, V.C., Flynn, M., Sylvester, D., Najafi, K., Gordon, T., Lepech, M., Emami-Naeini, A., Krimotat, A., Ettouney, M., Alampalli, S. and Ozdemir, T. (2009), "Overview of a cyber-enabled wireless monitoring system for the protection and management of critical infrastructure systems", *Proceedings of the SPIE - The International Society for Optical Engineering,* 7294, art no. 72940L.

Lynch, J.P. and Loh, K.J. (2006), "A summary review of wireless sensors and sensor networks for structural health monitoring", *Shock Vib. Digest*, **38**(2), 91-130.

Marzouk, M.M. and Hisham, M. (2011), "Bridge information modeling in sustainable bridge management", *Proceedings of the ICSDC 2011*.

McFadin, P. (2015), "Getting Started with Time Series Data Modeling," [Online article], Retrieved from: https://academy.datastax.com/demos/getting-started-time-series-data-modeling (accessed on 16 Dec 2015).

McNeill, D.K. (2009), “Data management and signal processing for structural health monitoring of civil infrastructure systems”, In Structural Health Monitoring of Civil Infrastructure Systems, (Eds., Karbhari, V. M. and Ansari, F.), CRC Press, Boca Raton, FL.

O'Connor, S.M., Zhang, Y., Lynch, J., Ettouney, M. and van der Linden, G. (2014), “Automated analysis of long-term bridge behavior and health using a cyber-enabled wireless monitoring system”, *Proceedings of SPIE - The International Society for Optical Engineering*, 9063, art. no. 90630Y.

O'Connor, S.M., Zhang, Y. and Lynch, J.P. (2015), “Automated outlier detection framework for identifying damage states in multi-girder steel bridges using long-term wireless monitoring data , *Proceedings of the SPIE Smart Structures/NDE Conference*, art no. 94370N.

Open Geospatial Consortium (2014), “OGC® SensorML: Model and XML Encoding Standard” [online article], Retrieved from: http://www.opengeospatial.org/standards/sensorml (accessed on 16 Dec 2015).

Overschee, P.V. (2002), “Subspace Identification for Linear Systems,” [Matlab package]. Retrieved from: http://www.mathworks.com/matlabcentral/fileexchange/2290-subspace-identification-for-linear-systems (accessed on 16 Dec 2015).

Padhy, R.P., Patra, M.R. and Satapathy, S.C. (2011), “RDBMS to NoSQL: Reviewing some next-generation non-relational databases”, *Int. J. Adv. Eng. Sci. Technol.*, **11**(1), 15-30.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V. and Vanderplas, J. (2011), “Scikit-learn: Machine learning in Python”, *J. Machine Learning Res.*, **12**, 2825-2830.

Ray, S. (2002), “Interoperability standards in the semantic web”, *J. Comput. Inform. Sci. Eng. - ASME*, **2**, 65-69.

Samec, V., Stamper, J., Sorsky, H. and Gilmore, T.W. (2014), “Long span suspension bridges – bridge information modeling”, *Proceedings of the IABMAS 2014*.

Shirolé, A.M., Chen, S.S. and Puckett, J.A. (2008), “Bridge information modeling for the life cycle: progress and challenges”, *Proceedings of the IBSMC08-025*.

Smarsly, K., Law, K.H. and Hartmann, D. (2013), “A cyberinfrastructure for integrated monitoring and life-cycle management of wind turbines”, *Proceedings of EG-ICE 2013*.

Sohn, H., Lim, H.J. and Yang, S. (2015), “A fatigue crack detection methodology”, Smart Sensors for Health and Environment Monitoring - Springer, KAIST, 233-253.

Thantriwatte, T.A.M.C. and Keppetiyagama, C.I. (2011), “NoSQL query processing system for wireless ad-hoc and sensor networks”, *Proceedings of the International Conference on Advances in ICT for Emerging Regions, ICTer 2011*, art. no. 6075030.

Zhang, Y., Kurata, M., Lynch, J.P., Van der Linden, G., Sederat, H. and Prakash, A. (2012), “Distributed cyberinfrastructure tools for automated data processing of structural monitoring data”, *Proceedings of the SPIE - The International Society for Optical Engineering*, 8347, art no. 83471Y.

Zhang, Y., O’Connor, S., van der Linden, G., Prakash, A. and Lynch, J. (2016), “SenStore: A scalable cyberinfrastructure platform for implementation of data-to-decision frameworks for infrastructure health management”, *J. Comput. Civil Eng.*, 10.1061/(ASCE)CP.1943-5487.0000560 , 04016012.

Zhou, G.D. and Yi, T.H. (2013), “Recent developments on wireless sensor networks technology for bridge health monitoring”, *Mathematical Problems in Engineering*, 947867.