A Data Management Infrastructure for Bridge Monitoring

Seongwoon Jeong*[a], Jaewook Byun[b], Daeyoung Kim[b], Hoon Sohn[c], In Hwan Bae[d], Kincho H. Law[a]
[a]Dept. of Civil & Environmental Engineering, Stanford University, Stanford, CA, USA 94305;
[b]Dept. of Computer Science, KAIST, Daejeon 305-701, Republic of Korea; [c]Dept. of Civil & Environmental Engineering, KAIST, Daejeon 305-701, Republic of Korea; [d]New Airport Hiway, Incheon 414-16, Republic of Korea

ABSTRACT

This paper discusses a data management infrastructure framework for bridge monitoring applications. As sensor technologies mature and become economically affordable, their deployment for bridge monitoring will continue to grow. Data management becomes a critical issue not only for storing the sensor data but also for integrating with the bridge model to support other functions, such as management, maintenance and inspection. The focus of this study is on the effective data management of bridge information and sensor data, which is crucial to structural health monitoring and life cycle management of bridge structures. We review the state-of-the-art of bridge information modeling and sensor data management, and propose a data management framework for bridge monitoring based on NoSQL database technologies that have been shown useful in handling high volume, time-series data and to flexibly deal with unstructured data schema. Specifically, Apache Cassandra and Mongo DB are deployed for the prototype implementation of the framework. This paper describes the database design for an XML-based Bridge Information Modeling (BrIM) schema, and the representation of sensor data using Sensor Model Language (SensorML). The proposed prototype data management framework is validated using data collected from the Yeongjong Bridge in Incheon, Korea.

Keywords: Structural health monitoring, data management, NoSQL database, bridge information modeling

1. INTRODUCTION

As sensor technologies mature and become economically affordable, the use of sensors for the health monitoring of infrastructures, such as bridges, will continue to grow [1, 2]. Installations of sensor networks on long-span bridges have been deployed [3, 4, 5]. The trend for bridge monitoring will involve installation of hundreds and thousands of sensors to collect valuable information about the state of the bridge structure [6]. Together with routine maintenance and inspection reports, the collected sensor data will enable the diagnosis of potential structural problems and the prognosis for the need of structural strengthening and repairs. Developments of advanced sensing materials and devices, e.g. piezo lead zirconate titanate (PZT) and fiber Bragg grating (FBG) sensors, can provide measurements for detailed and precise structural evaluation but also entail enormous amount of data because of high sampling rate [7, 8, 9]. Structural monitoring systems will be inundated with data that need to be processed, interpreted and brought forth to support lifecycle bridge management. While current structural health monitoring research continues to develop and explore new sensor technologies, very little efforts have been spent to investigate proper data management tools to efficiently store, manage, and retrieve sensor data. The data issues are of fundamental importance that need to be dealt with before sensing technologies can truly find useful for bridge lifecycle assessment and management.

Information models and interoperability standards have been proposed for quite some time as a means to build semantic web services (i.e. network-enabled engineering services accessible on the web) in engineering [10, 11]. In the building and construction industry, building information modeling (BIM) applications have begun to emerge as a vehicle to

* e-mail: swjeong3@stanford.edu; phone 1 650 666-5556;

support Integrated Project Delivery process through exchange of information with open standards. By adhering to open standard data modeling and format, information exchange and interoperability are now widely supported by BIM application software. Building information modeling has emerged as a broadly adopted, pervasive technology in the building industry worldwide. With advances in BIM technologies, much research has been attempted to develop standard Bridge Information Modeling (BrIM) standards [12, 13]. The objective of BrIM is to facilitate bridge lifecycle management by creating and maintaining a centralized and sharable data model for bridge structure [13, 14]. BrIM includes parametric information (such as 3-dimensional geometry), structural information (such as material), and management data (such as inspection output) and is intended to support data exchange among applications and project participants [15, 16]. BrIM schemas use extensible markup language (XML) as the basic syntax to attain interoperability [17, 18]. While XML based model representation has been shown appropriate, the XML based schemas often involve complex data structures such as nested data hierarchy. Designing a BrIM database that can support semi-structured and unstructured data schema would be greatly beneficial to take full advantage of state-of-the-art BrIM technologies.

Selecting an appropriate database tool for specific application is important for successful deployment of data management system. Relational database management system (RDBMS) and structured query language (SQL) have been widely used across all industries because of its reliability, convenience, and extensive user base. However, recent studies have shown that RDBMS has some fundamental limitations, in terms of reading and writing speed, capacity, scalability, and flexibility, particularly for the management of unstructured information [19, 20]. To overcome some of the limitations of RDBMS, NoSQL (Not Only SQL) database systems have been proposed, developed and exploited [21]. NoSQL database systems enable faster performance and support more flexible data schema in comparison to RDBMS by relaxing some of the rigid consistency and strict data schema requirements [22]. Relational tables and data structures are also not suitable for handling large and unstructured data acquired from bridge monitoring. Furthermore, while complex query and high consistency are not necessarily the key issues, efficient access and fast retrieval are important for structural monitoring applications. NoSQL represents a preferred alternative for managing bridge monitoring data because of its abilities to deal with big data and flexible schema.

This paper discusses a data management framework for bridge monitoring that is designed to handle sensor data and BrIM model utilizing NoSQL database systems. We first review current efforts in sensor data management and BrIM. Open source NoSQL database tools such as Apache Cassandra and MongoDB are investigated and deployed for prototype implementation. The database management system is described and prototype testing is conducted using the monitoring data collected from the Yeongjong Bridge in Incheon, Korea.

2. SYSTEM CONFIGURATION

2.1 Sensor data management system

The data management issues are of fundamental importance that need to be dealt with before advanced sensor technologies can truly find useful for bridge monitoring and management applications. Few research has been conducted focusing on the data management issues. McNeill provided an overview of data management issue for SHM [23]. Law et al. further discussed the data issues for the deployment of wireless sensor and sensor networks and the need for persistent backend data storage, management and access [24]. Smarsly et al. discussed sensor data management technologies for a wind turbine monitoring system [25]. Zhang et al. proposed a cyber-infrastructure system for managing bridge monitoring data and implemented with the SHM system on the New Carquinez Bridge [26]. The overall system configuration to be discussed in this paper is depicted as shown in Figure 1. The sensor data management system consists of an on-site computer, a centralized server, users' computers, and a web interface. The on-site computer receives raw sensor data from the sensor network on a bridge structure, stores the data temporarily, pre-processes the raw data, and sends them to the server. As a permanent storage, the centralized server stores all the pre-processed sensor data as well as other pertinent information regarding bridge specification, geometry, sensor information, etc. The users or local computers serve as the compute nodes that periodically retrieve data from the server, analyze the data, and sends the analysis data back to the server. End users can then retrieve the evaluation results from the server via web interface and other mobile devices [24].

Current SHM data management systems are mostly utilize SQL-based RDBMS. Recent studies have suggested that NoSQL database systems can show better performance for sensor network application than SQL database in terms of read/write speed and scalability [27, 28, 29]. In this study, we employ NoSQL database systems, instead of traditional RDBMS, to facilitate schema definition and to enable efficient performance. The desirable features for the on-site

computer and local computer would be a database tool that supports query languages for data analysis, while the key issues for the server would be extensibility and scalability to support long-term data management and archiving. Section 3 will discuss in details the selection of NoSQL database systems for the applications.

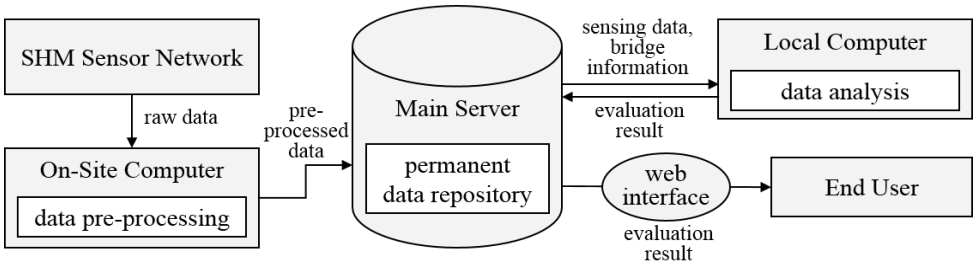


Figure 1. Sensor data management framework for structural health monitoring

2.2 Bridge information model repository

A bridge information model includes not only a 3-dimensional geometric model but also a variety of information related to the bridge that are shared among project participants from design to management [13, 30]. Ideally, the information model can be translated into different file formats for different applications on demand. With the success of implementation of building information modeling (BIM) in the building and construction industry, BrIM can potentially lead to similar benefits such as accurate model and consistent designs [26, 27]. The use of BrIM has the potential to reduce workloads and human errors by avoiding manual file translation [31, 32]. To achieve interoperability, many of current efforts focus on developing data exchange standards based on easily understandable marked up language such as XML. Karaman et al. proposed an interoperable data schema for curved steel bridges [17]. Ali et al. described a data schema for concrete-bridge [18]. BrIM is expected to play an important role for bridge monitoring and management by allowing users to retrieve inspection data or sensor data through 3-D bridge models. For instance, Samec et al. developed a web-based BrIM system where engineers can upload and view bridge inspection data including photo, video, and audio clip via 3-D visualization tool [14]. Marzouk and Hisham presented a BrIM framework integrated with bridge management system (BMS) [33].

For the data repository development, we have designed a data management infrastructure to store and share bridge information model and sensor data. OpenBrIM 2.0, an open source XML based BrIM schema, is selected to create a bridge information model for this study [34]. OpenBrIM allows users to describe a bridge with hierarchical *Obj* element sets. An *Obj* may compose of child(ren) elements and can define its type *ObjType*s as a template to describe frequently used child *Obj*s. The schema-free feature of NoSQL database is particularly desirable to store the complex XML schema that can be difficult to define in terms of relational database structure. Figure 2 shows BrIM document example using *Obj* and *ObjType*, which describe the skeleton of the BrIM schema. The details for the implementation of BrIM and sensor data management using NoSQL will be discussed in Section 4.

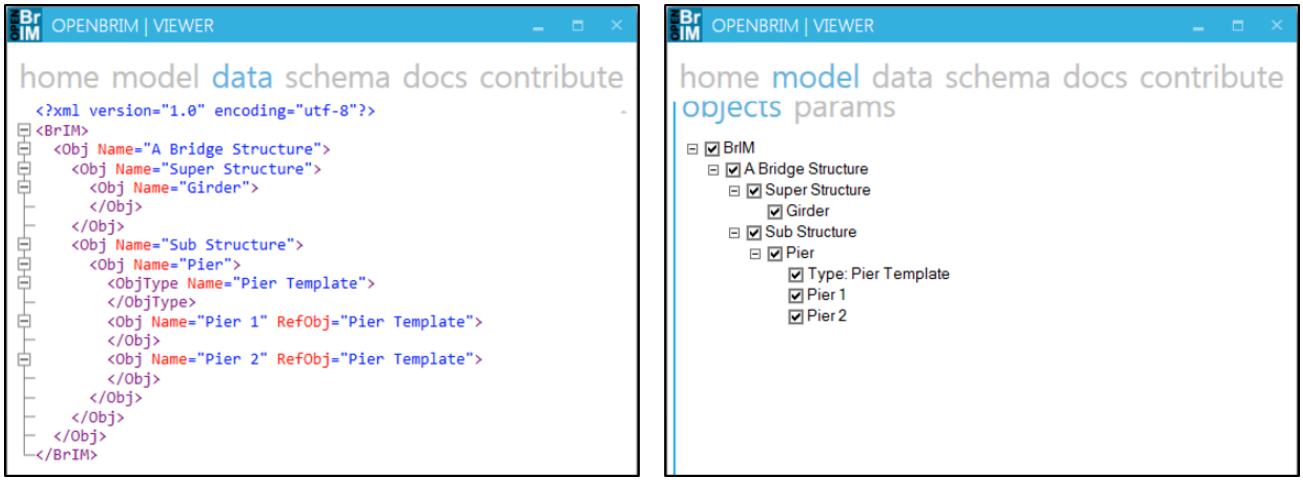


Figure 2. Simplified BrIM document (left) and corresponding Object hierarchy (right) on OpenBrIM Viewer

3. SELECTION OF NOSQL DATABASE TOOLS

There have been a number of NoSQL database systems with different features and properties. In general, NoSQL database systems can be categorized into key-value stores, document oriented stores, and column family stores according to the data models employed [19]. Each has its own strength and it is important to select the tool(s) for specific application. The key-value type database systems, such as Redis exhibit very fast performance because of the in-memory operations [35]; these systems however tend to have relatively small data capacity often limited to the main memory capacity of the computer. In this study, we examine the database tools that are based on document stores and column family stores.

3.1 Database system for supporting efficient querying

For efficient query purpose, we select MongoDB, a document oriented data storage that features powerful query capability. MongoDB schema consists of the key-value sets of binary JSON (BSON) schema-less documents. MongoDB enables relationships between elements by providing reference and embedded document. Furthermore, the database system supports complex data structure such as nested data or tree-structured data [36]. The strong query capability allows search by key or value, as well as indexing, range query, aggregation operation, etc. To achieve performance, MongoDB implements many measures such as data pre-allocation, memory-mapped storage engine, and dynamic query optimizer [37]. In addition, MongoDB can be scaled by "sharding" – a collection of data on multiple machines by splitting the data into ranges and distributing the data across multiple shards (involving master and slave machines) [36].

Because of its flexible schema (or schema-less), rich aggregation tools and capability of handle very large data sets, MongoDB has been widely used in many fields including Internet of Things (IoT) applications and real-time analysis [38, 39, 40, 41]. The data management infrastructure proposed will take advantage of MongoDB's efficient querying performance to support data processing on on-site computers as well as the local (user) computers.

3.2 Database system for supporting persistent archiving

Another NoSQL database system examined in this study is Apache Cassandra, which supports column-oriented storage. The basic data structure in Cassandra consists of key space, column family, row, and key-value pairs. The Cassandra's schema definition is not as flexible as schema-less document oriented database system because the column family needs to be predefined. But the schema definition is much more flexible than a relational database by allowing different number of attributes in each row [19]. Cassandra database systems show significant advantages using distributed data storage over multiple machines [42]. In particular, Cassandra uses decentralized and distributed system often with replications to guarantee that failure at single point would not cause total system failure whereas MongoDB uses master-slave replication that failure at the master node would lead to failure of the entire database [43]. For huge data sets, Cassandra's efficient partitioning based on consistent hashing also has performance advantages over document-oriented stores [19].

On the other hand, because of the decentralized and distributed data partitioning nature, the query performance of Cassandra is not as efficient as the range-based query strategy of MongoDB. The column oriented data type is not as flexible for describing complex data types and complex queries. However, Cassandra is particularly useful for dealing with high volume of data transaction and storage, especially with write operations [43, 44, 45]. 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 [46, 47].

For bridge monitoring application, database systems such as Cassandra that allow easy expansion and schema change, and increasing volume of sampling data are particularly suitable for persistent data stores. The database server is likely to have high read and write transaction rates as it continuously acquire data from the sensor network, but data retrieval from the server for analysis will be performed selectively and less frequently. To take advantage of its scalability and write efficiency, Cassandra database system is implemented on the main server for the prototyped data management infrastructure. Figure 3 shows the overall framework of the data management infrastructure which includes MongoDB, a document oriented data storage system for the onsite and local computers, and Apache Cassandra, a column oriented data storage system for the main server.

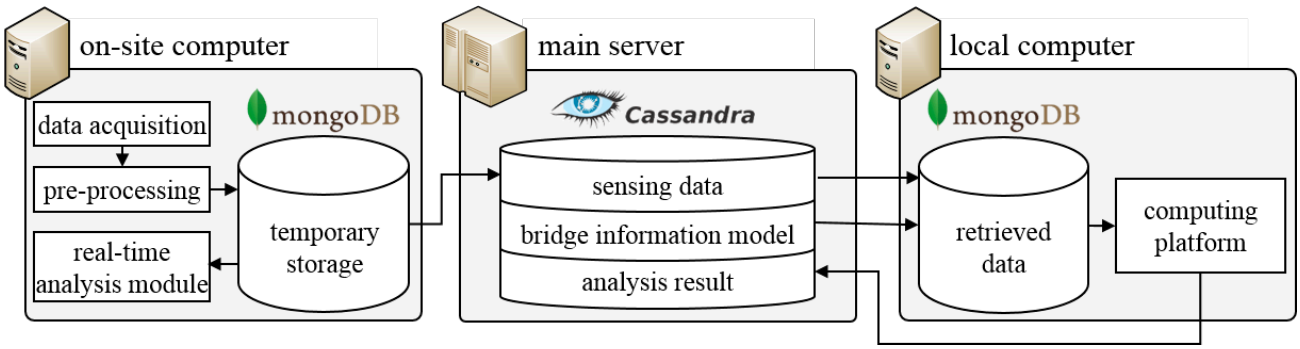


Figure 3. NoSQL based data management framework for bridge monitoring

4. DATA SCHEMA DESCRIPTION

Although the schema-free feature typifies NoSQL database system, data schema definition can significantly facilitate data retrieval and system automation. The pre-defined data schema does not imply that the system is restrained by a rigid data structure. The data schema can be easily changed and scaled. For example, when a new type of sensor is added to the bridge monitoring system, it is not necessary to edit the whole data structure for the NoSQL based data management infrastructure. The new sensor information can be added to the existing data schema.

For on-site computer storage, MongoDB is used to store only the sampling data acquired from the sensor network. On the other hand, for local computer's database, MongoDB stores not only the sampling data, but also the bridge information for analysis. Cassandra, as the main data storage system, will handle sensor information and bridge information as well as sensing data. A unified data schema for the sampling data, the sensor information, and the bridge information model are defined for both of the MongoDB and the Cassandra database systems.

4.1 Sensing data

Figure 4 shows the basic data hierarchy defined for the MongoDB database. The uppermost namespace is called database, which contains a set of collections. A collection stores a set of documents where each document is a fundamental data unit in the MongoDB database. The document is represented in the BSON structure that includes a set of field-value pairs where a value can be either a single data or a list of data. In the current prototype implementation, a database is defined for a project or a bridge structure, thus the database is named after the name of the bridge. To store all documents of sensing data, a collection named *sensingdata* is defined.

Figure 5 shows how a document stores the sampling data. Each Document stores the pre-processed data from an input file along with the date and the time that the data was acquired and the name of the sensor. Currently, each document collects a list of measured data over the period of one second. For example, if the sampling rate is 5Hz, the sampled data is divided into buckets; each bucket has five consecutive data and is stored in a single document. Since a document in MongoDB can have up to 16MB of storage [36], the strategy to partition the data according to sampling rate and the data storage is necessary to prevent data overflow which can be caused by sensors that have high sampling rate.

Apache Cassandra also has a hierarchical data structure as described in Figure 6. The top-level keyspace (similar to the namespace of database in MongoDB) is defined for a specific project or a bridge. The column family consists an array of rows where each row consists of a set of columns. Each column represents a basic element in the Cassandra database and is assigned a name and value pair. Although the column family is similar to a table in RDBMS and a row is similar to a tuple in RDBMS, the basic difference is that for each row a column family of Cassandra database can have different number of columns, while each tuple in RDBMS has the same number of attributes defined for the table.

When managing sampling data using Cassandra, even with consistent hashing, distributing consecutive data to different nodes could weaken query performance for continuous, time series data. To overcome this shortcoming, we employed Oliot-EPCIS, a Cassandra based data repository platform [27, 48]. Oliot-EPCIS deals with the query efficiency issue for the time series data by allocating partitioning key according to the sensor ID and event time to make sure that consecutive data is placed in the same node [48]. Figure 7 shows how the Oliot-EPCIS stores the sampling data. First, the Oliot-EPCIS platform captures the information from an XML document that contains the sensor ID, the event time, the sampling data, and other information. The system then converts the data into Cassandra input format and sends the

converted data to the column family named *objectevent*. Each row has a key in the form of "*sensorID*|*yyyymm: event time*" where "*sensorID*|*yyyymm*" acts as a partitioning key [27]. In addition, every row is set up to store the data measured for a period of one second from a sensor.

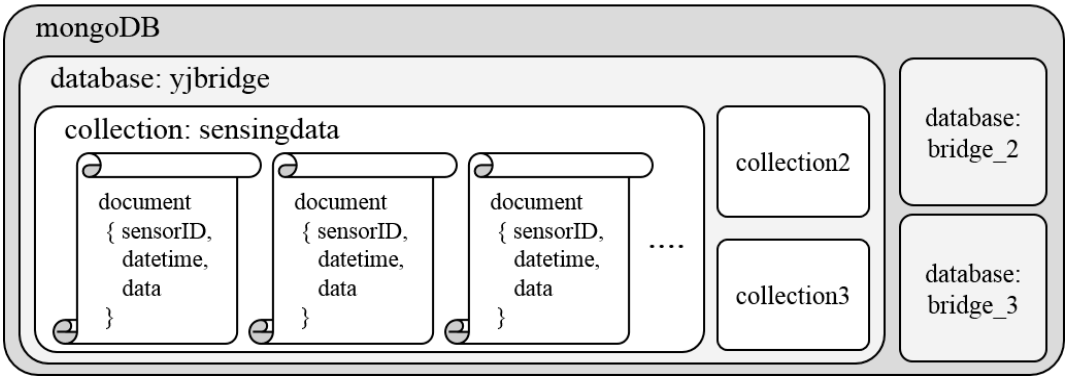


Figure 4. Data hierarchy in MongoDB

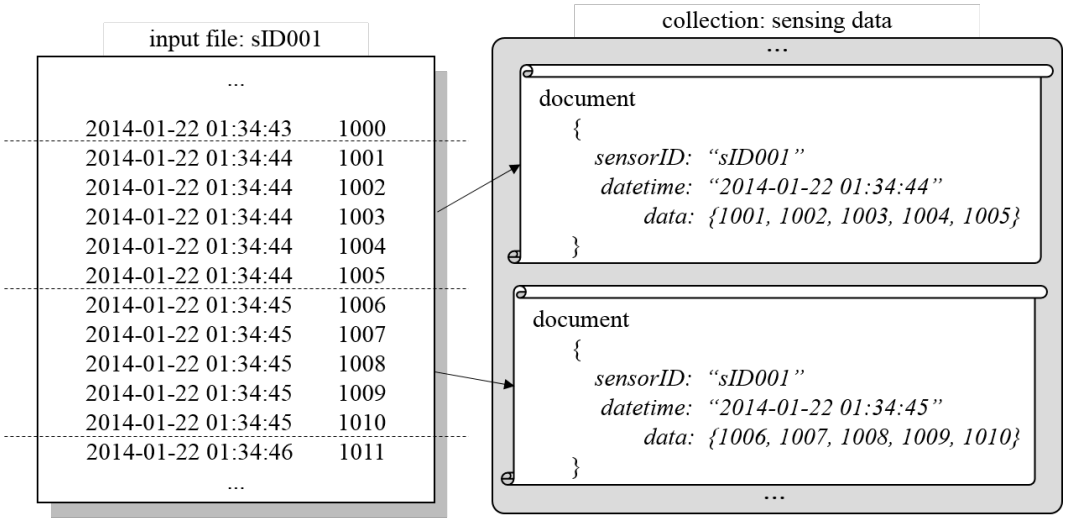


Figure 5. Data schema example of sensing data on MongoDB

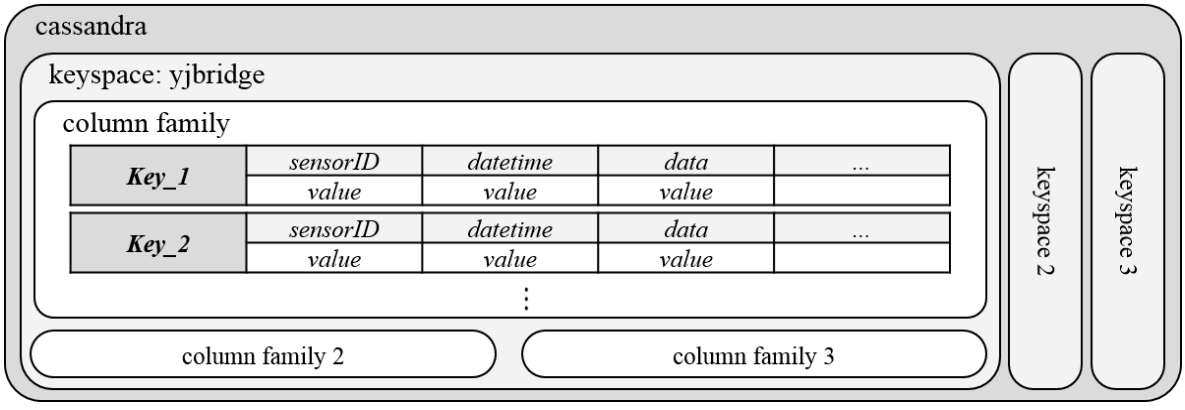


Figure 6. Data hierarchy in Cassandra

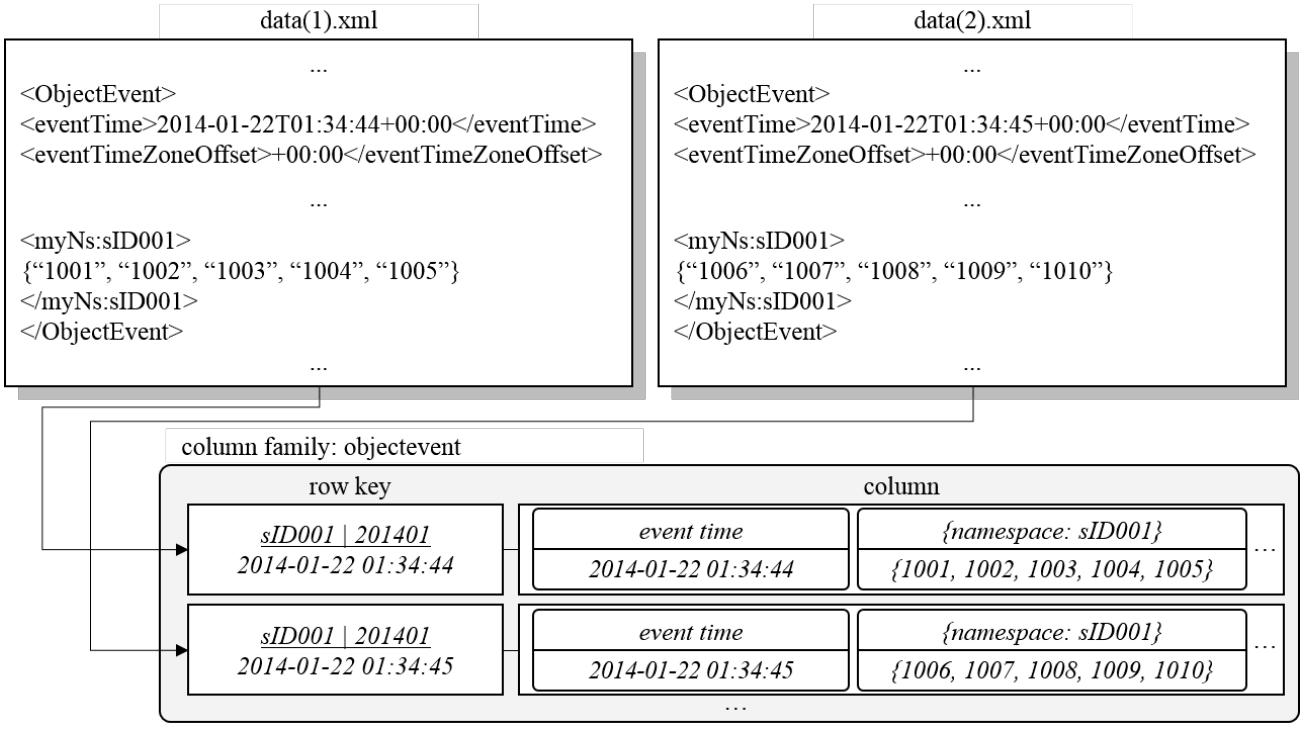


Figure 7. Data schema example of sensing data on Cassandra (Oliot-EPCIS)

4.2 Sensor information

The main server manages the information about all the sensors installed on a structure, and allows users to search about the sensors. For interoperability purpose, we adopt Sensor Model Language (SensorML), a standard for defining measurement and post-measurement process proposed by the Open Geospatial Consortium (OGC) [49]. Using the data schema described in SensorML, the sensor information are categorized as shown in Table 1. Although not every sensor has the information for all the categories and attributes, the unstructured information can be easily handled elegantly with the schema-free feature of Cassandra.

The column family named “*sensorinformation*” is designed for storing the sensor information. Each row is assigned to a sensor and each column stores an attribute of the sensor. To manage the unstructured data, each row can have a different set of attributes and omitted information can be ignored (instead of having identical set of attributes with null values for omitted attributes). Figure 8 illustrates an instance of the column family “*sensorinformation*”.

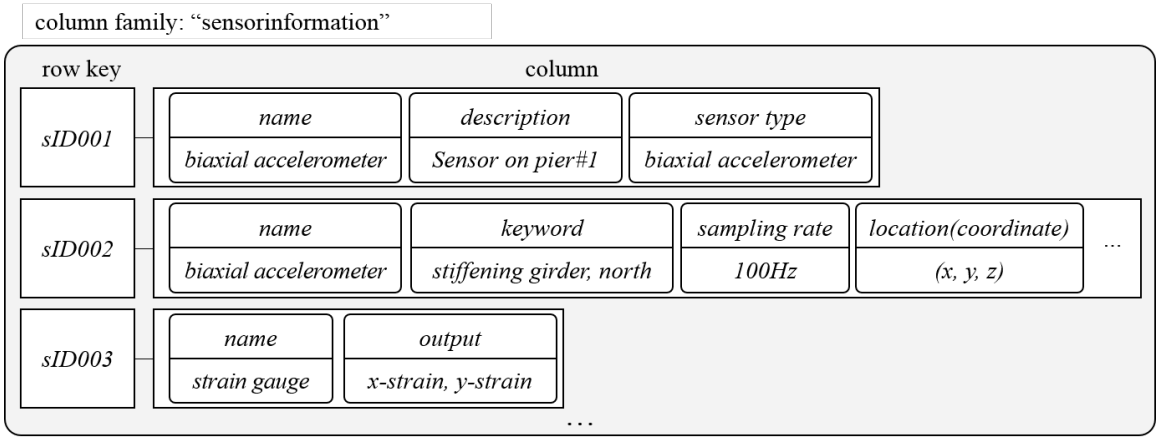


Figure 8. Data schema example of sensor information on Cassandra

Table 1. Category of sensor information [50]

Category	Attributes
System description	sensor ID, sensor name, description, keywords, group
Identifiers	long name, short name, model number, manufacturer
Classifiers	intended application, sensor type
Constraints	document valid time, security constraints, legal constraints
Input & output	input, output, unit of measurement, quantity definition
Location	location(text), location(section), location(coordinate)
Parameter	sampling rate
Physical properties	weight, weight unit, length, length unit, width, width unit, height, height, unit, casing material
Electrical requirements	voltage, current type, amp range, sensing range
Capabilities	sensing range, sensitivity, sample period, measurement output time
Contacts	responsible party, telephone, address
Documentation	manual
Data	data link

4.3 Bridge information model

In this study, a database schema is designed according to the bridge information modeling (BrIM) schema by Chen et al. [51]. First, a BrIM document defines an Objtype for each structural element type. We then arrange the Objs in the workspace corresponding to the actual structural elements. To store a BrIM document in Cassandra database, the column family "*structuretype*" and "*structurearrangement*" are created as illustrated in Figure 9. For the repository of *Objtype*, "*structuretype*" contains a set of rows where each row is allocated to each *Objtype*. The columns of a row store the attributes of the *Objtype* such as start point, end point, and surface definition. The column family "*structurearrangement*" corresponds to *Obj*s; each row represents a part of the structure and a column within a row stores an element of the structure in the form of XML excerpts. The *Obj*s can inherit the data of an *Objtype* by assigning a reference object *RefObj*. For instance, in Figure 9, the *Side Truss Vertical* column of the *Side Truss* row in the "*structurearrangement*" column family inherits the attributes of *Steel Box Girder* in the "*structuretype*" column family.

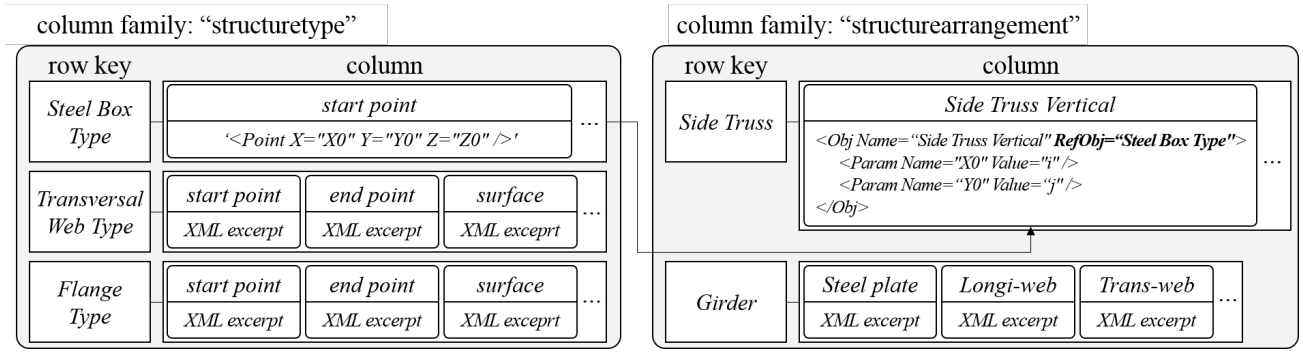


Figure 9. Bridge information model data schema on Cassandra

MongoDB stores BrIM document in the similar way as in the Cassandra. As shown in Figure 10, two collections named "*brim.objtype*" and "*brim.superobj*" are defined for *Objtype* and *Obj*, respectively. Each document in "*brim.objtype*" stores the information of each *Objtype*, and each document in "*brim.superobj*" contains the data of each *Obj*.

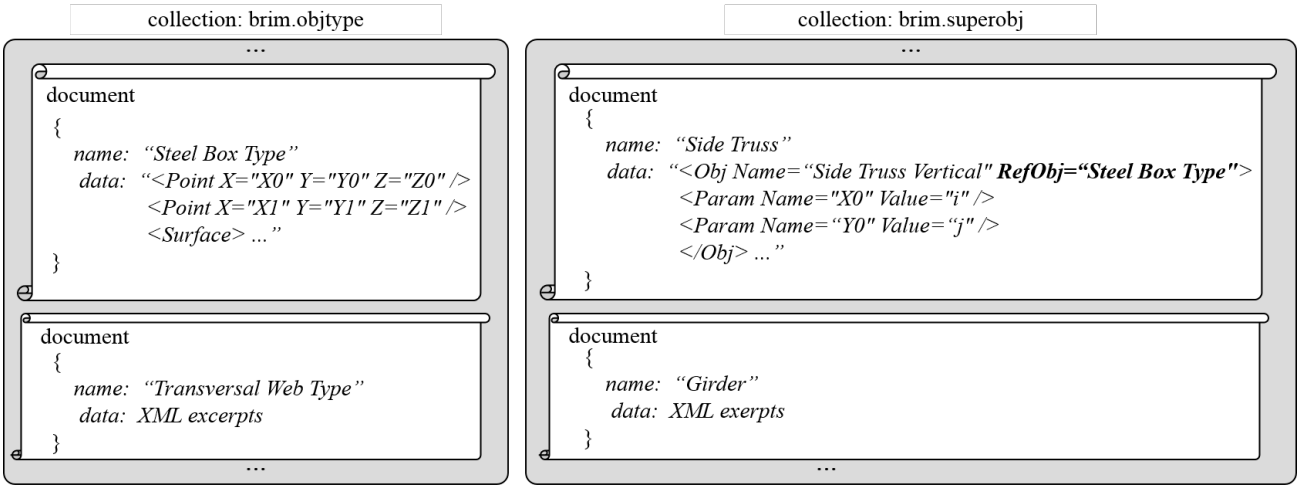


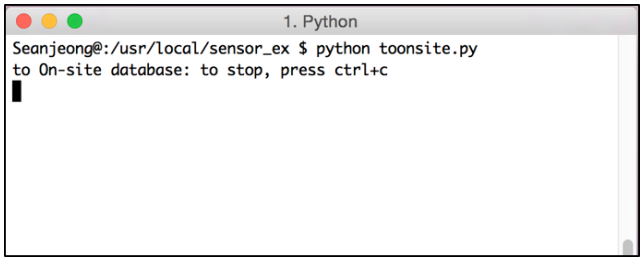
Figure 10. Bridge information model data schema on MongoDB

5. IMPLEMENTATION DETAIL

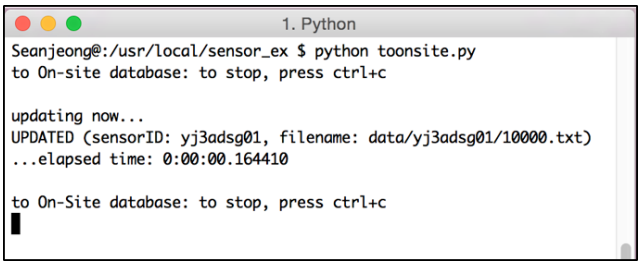
To test the proposed data schema and database design, we use the model of the stiffening truss girder of Yeongjong Bridge in Incheon, Korea and the collected sensor data as an example. Table 2 shows the description of the sampling data from the dynamic strain gauge sensors. The sensors collected the measurements at the sampling rate of 100Hz; thus a set of a hundred consecutive data is stored in a collection of MongoDB and in a row of Cassandra. The input file from the sensors contains 405 seconds of sensing data for about 1.3Mbytes. For prototyping, three programs written in python are developed to store the data to the on-site computer, to store data to the main server, and to retrieve data from the main server to a user's computer, respectively. Figure 11 shows a screenshot where the program receives a new data file and sends the data to MongoDB. Figure 11 (a) shows the initial status of the program, and Figure 11 (b) shows the status after the input file *10000.txt* is created in the folder named *yj3adsg01*. The program console shown in figure 11 (b) indicates that the input file named *10000.txt* is updated for the sensor *yj3adsg01*. Once the program reads all the input files, the collection *sensingdata* holds the 405 documents per each sensor (Figure 12 (a)), and each document stores a list of a hundred data values (Figure 12 (b)).

Table 2. Detailed description of sampling data

Sensor ID	**yj3adsg01**	**yj3adsg02**	**yj3adsg03**	**yj3adsg04**	**yj3adsg05**	**yj3adsg06**
Name	Dynamic strain gauge					
Sampling period	2014-01-22 01:34:44 ~ 2014-01-22 01:41:28 (405 seconds)					
Sampling rate	100 Hz					
Input file	10000.txt	20000.txt	30000.txt	40000.txt	50000.txt	60000.txt
Description	Dynamic strain gauge on stiffening truss girder at west pylon					
Keyword	Dynamic Strain Gauge, Stiffening Truss Girder, West Pylon					
Group	Suspension Bridge:Stiffening truss girder:West pylon:Dynamic strain					
Output	dynamic strain					
Unit	microstrain					
Location (text)	Stiffening truss girder at west pylon of Yeongjong Bridge					
Location (section)	3A-3A					

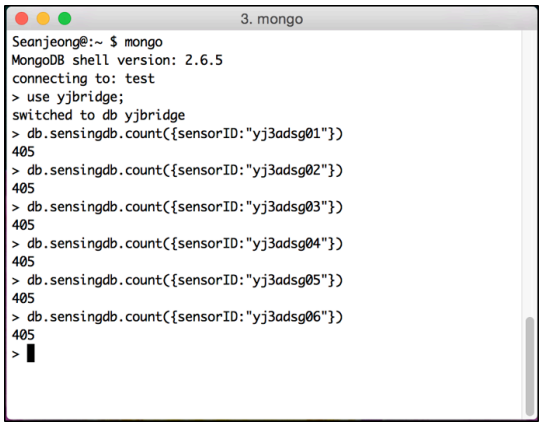


(a) Initial screen

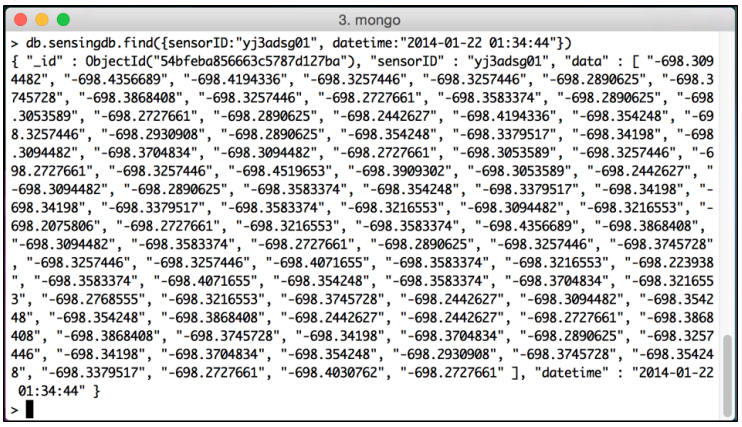


(b) Screen after storing 10000.txt

Figure 11. toonsite.py: Program storing data to MongoDB in on-site computer

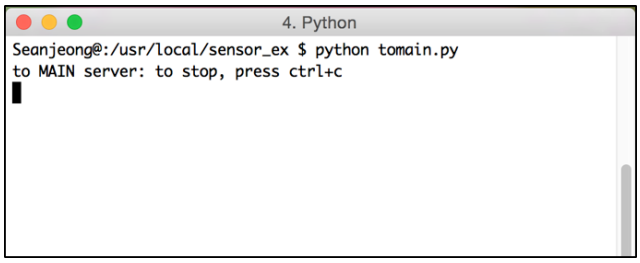


(a) The number of document of each sensor

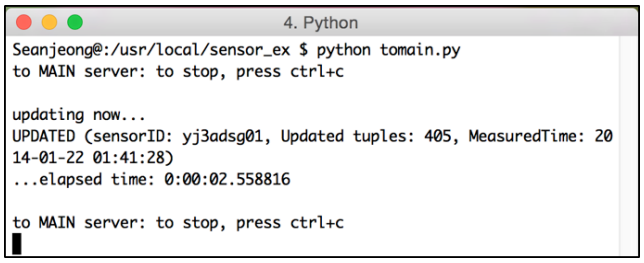


(b) Retrieved document

Figure 12. Results of running toonsite.py



(a) Initial screen



(b) Screen after catching 405 updates in on-site computer

Figure 13. tomain.py: Program storing data to Cassandra in main server

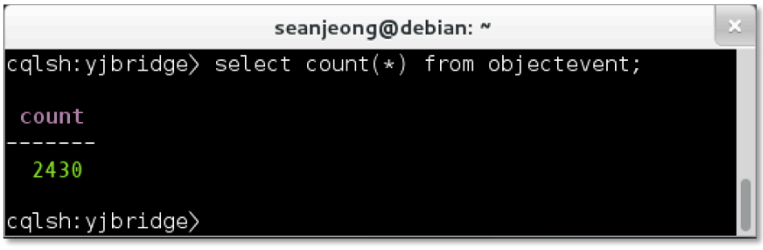
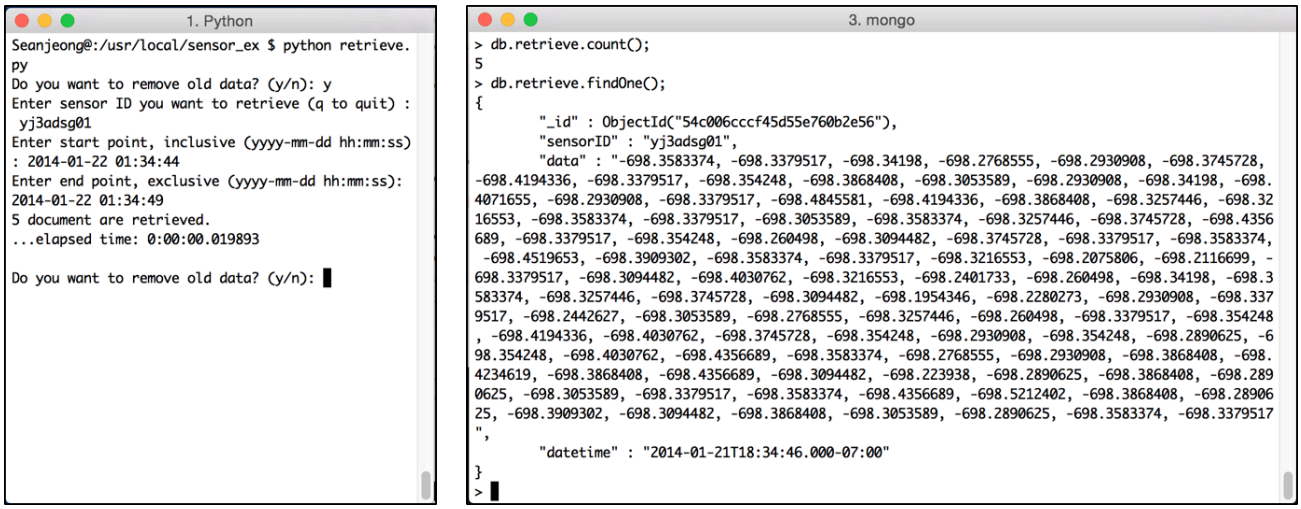


Figure 14. Result of running tomain.py

Once the update on the MongoDB in the on-site computer is completed, the MongoDB documents are converted into XML documents, and the documents are sent to Cassandra database. Figure 13 shows the process in transferring the MongoDB document to the Cassandra database. Figure 13 (b) shows that 405 rows are updated for the sensor *yj3adsg01* that is measured until 2014-01-22 01:41:28. As shown in Figure 14, after all the data from the on-site computer are transferred, the column family *objectevent* of Cassandra has a total of 2,430 rows which is identical to the number of sensors multiplied by number of data entry for each sensor.



(a) Searching on retireve.py (b) Retrieved document in MongoDB

Figure 15. Data retrieval using retrieve.py

To retrieve the data from the main server, the user enters the search conditions including information such as sensor ID and time period. The query is then sent to the Cassandra database (via web querying platform provided by Oliot-EPCIS), and retrieves the data to the MongoDB database resided in the local computer. The retrieved data can be found in the collection named *retrieve* in MongoDB. As shown in Figure 15 (a), the user search the sensing data collected by sensor yj3adsg01 from 2014-01-22 01:34:44 inclusive to 2014-01-22 01:34:49 exclusive. As a result, five documents are retrieved as shown in figure 15 (a), and the retrieved data are shown in MongoDB console (see figure 15 (b)).

The sensor shown in Table 2 is stored in the *sensorinformation* column family in the Cassandra database. An Excel spreadsheet is created to convert the table of sensor information into the inputs for Cassandra. Once converted, users can then store the sensor via the Cassandra console. Figure 16 shows the example sensor information for sensor *yj3adsg01* retrieved directly from the Cassandra console.

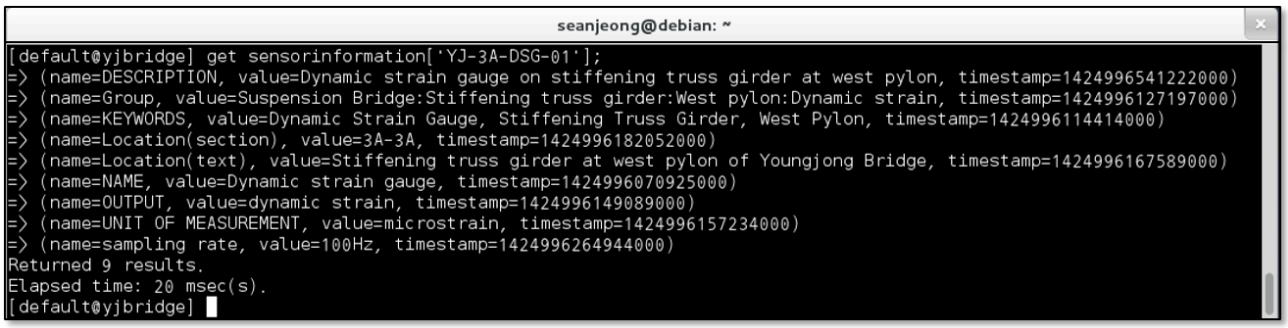
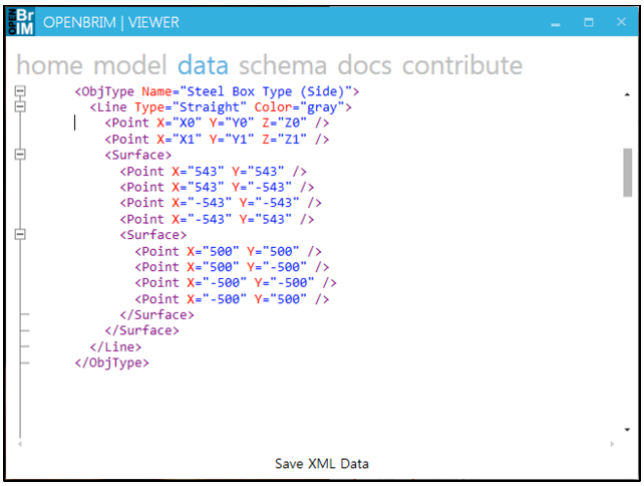


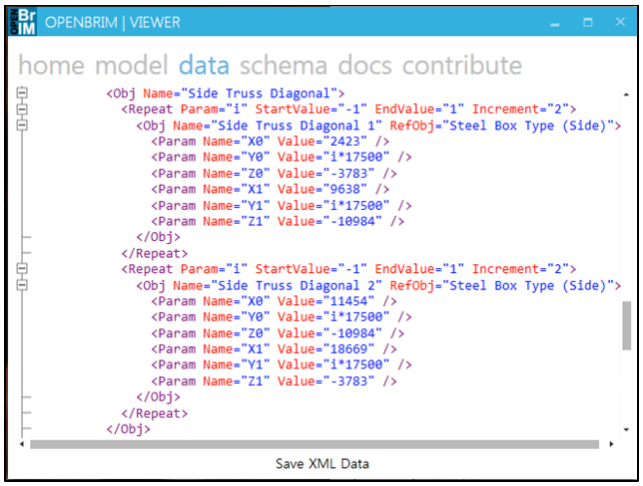
Figure 16. Sensor information for yj3adsg01 retrieved from Cassandra

In order to store the bridge information model, we first made an OpenBrIM document describing the stiffening truss girder of the Yeongjong Bridge. The OpenBrIM document consists of two parts – *ObjType* and *Obj*. Figure 17 shows the examples of an *ObjType* named *Steel Box Girder (Side)* and an *Obj* named *Side Truss Diagonal* that inherit the features of *Steel Box Girder (Side)* via the referencing object *RefObj*. The 3-dimensional model displayed using the OpenBrIM

Viewer is shown in Figure 18 (a). Figure 18 (b) shows the detail view of the sensor information attached to the bridge information model.

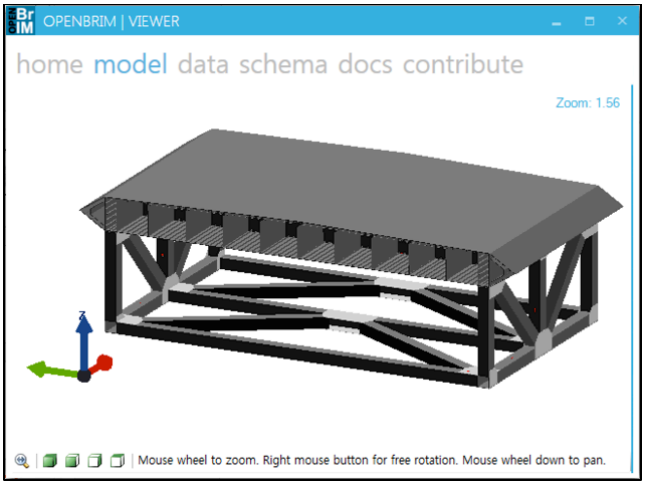


(a) objtype: steel box type (side)

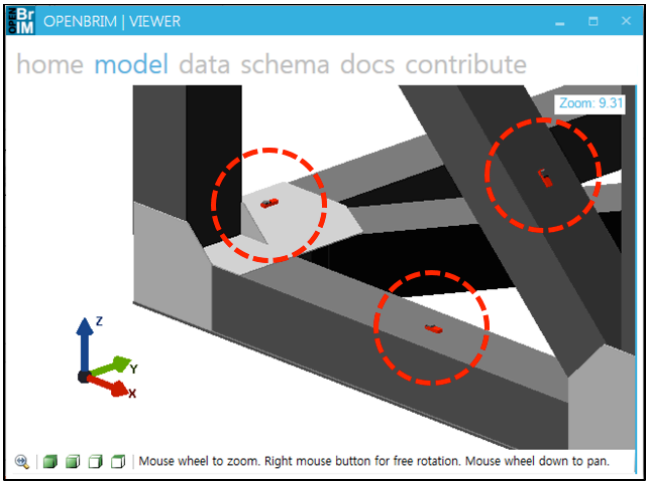


(b) obj: side truss diagonal

Figure 17. Excerpts of BrIM document of the Yeongjong Bridge's stiffening truss girder



(a) 3-D view of stiffening truss girder



(b) Detail view of sensor geometry

Figure 18. 3-D view of Yeongjong Bridge's stiffening truss girder (OpenBrIM Viewer)

Currently the BrIM document for Yeongjong Bridge is manually converted into MongoDB inputs and stored in Cassandra database. This converted input file follows the schema described in Figure 9 and Figure 10. For example, the object type *Steel Box Girder (Side)* occupies one document in the collection *brim.object* and the object *Side Truss Diagonal* is stored in a document named Side Truss, which belongs to *brim.superobj* as an element group, together with the sibling objects. Similarly, Cassandra stores the *Steel Box Girder (Side)* in a row of *structuretype* column family and *Side Truss Diagonal* in a column of row named *Side Truss* that belongs to *structurearrangement* column family. Figure 19 and Figure 20 show the retrieved results of *Steel Box Girder (Side)* and *Side Truss Diagonal* from the MongoDB and the Cassandra database, respectively.

(a) ObjType: Steel Box Type (Side)	(b) Obj: Side Truss Diagonal
> db.brim.objtype.find({"_id":3}) *{ "_id" : 3, "data" : "<Line Type=\"Straight\" Color=\"gray\">\n<Point X=\"X0\" Y=\"Y0\" Z=\"Z0\" />\n<Point X=\"X1\" Y=\"Y1\" Z=\"Z1\" />\n<Surface>\n<Point X=\"543\" Y=\"543\" />\n<Point X=\"543\" Y=\"-543\" />\n<Point X=\"-543\" Y=\"-543\" />\n<Point X=\"-543\" Y=\"543\" />\n<Surface>\n<Point X=\"500\" Y=\"500\" />\n<Point X=\"500\" Y=\"-500\" />\n<Point X=\"-500\" Y=\"-500\" />\n<Point X=\"-500\" Y=\"500\" />\n</Surface>\n</Surface>\n</Line>\n", "name" : "Steel Box Type (Side)" }* >	> db.brim.obj.find({"_id":3}) *{ "_id" : 3, "data" : " ... <Obj Name=\"Side Truss Diagonal\">\n<Repeat Param=\"i\" StartValue=\"-1\" EndValue=\"1\" Increment=\"2\">\n<Obj Name=\"Side Truss Diagonal 1\" RefObj=\"Steel Box Type (Side)\">\n<Param Name=\"X0\" Value=\"2423\" />\n<Param Name=\"Y0\" Value=\"i*17500\" />\n<Param Name=\"Z0\" Value=\"-3783\" />\n<Param Name=\"X1\" Value=\"9638\" />\n<Param Name=\"Y1\" Value=\"i*17500\" />\n<Param Name=\"Z1\" Value=\"-10984\" />\n</Obj>\n</Repeat>\n<Repeat Param=\"i\" StartValue=\"-1\" EndValue=\"1\" Increment=\"2\">\n<Obj Name=\"Side Truss Diagonal 2\" RefObj=\"Steel Box Type (Side)\">\n<Param Name=\"X0\" Value=\"11454\" />\n<Param Name=\"Y0\" Value=\"i*17500\" />\n<Param Name=\"Z0\" Value=\"-10984\" />\n<Param Name=\"X1\" Value=\"18669\" />\n<Param Name=\"Y1\" Value=\"i*17500\" />\n<Param Name=\"Z1\" Value=\"-3783\" />\n</Obj>\n</Repeat>\n</Obj> ... ", "name" : "Side Truss" }* >

Figure 19. Retrieved bridge information model from MongoDB

(a) ObjType: Steel Box Type (Side)	(b) Obj: Side Truss Diagonal
[default@yj] get structuretype['Steel Box Type (Side)']; *=> (name=End Point 1, value=<Point X="X1" Y="Y1" Z="Z1" />, timestamp=1417730293329000)* *=> (name=Line Type 1, value=Straight, gray, timestamp=1417730293323000)* *=> (name=Start Point 1, value=<Point X="X0" Y="Y0" Z="Z0" />, timestamp=1417730293326000)* *=> (name=Surface 1, value=<Point X="512.5" Y="512.5" /><Point X="512.5" Y="-512.5" /><Point X="-512.5" Y="-512.5" /><Point X="-512.5" Y="512.5" />, timestamp=1417730293351000)* *=> (name=Void 1, value=<Point X="500" Y="500" /><Point X="500" Y="-500" /><Point X="-500" Y="-500" /><Point X="-500" Y="500" />, timestamp=1417730293338000)* *Returned 5 results.* *Elapsed time: 5.01 msec(s).*	[default@yj] get structurearrangement['Side Truss']['Side Truss Diagonal']; *=> (name=Side Truss Diagonal, value=<Obj Name="Side Truss Diagonal"><Repeat Param="i" StartValue="-1" EndValue="1" Increment="2"><Obj Name="Side Truss Diagonal 1" RefObj="Steel Box Type (Side)"><Param Name="X0" Value="2423" /><Param Name="Y0" Value="i*17500" /><Param Name="Z0" Value="-3783" /><Param Name="X1" Value="9638" /><Param Name="Y1" Value="i*17500" /><Param Name="Z1" Value="-10984" />* *... (omitted) ...* *<Param Name="Z1" Value="-3783" /></Obj></Repeat></Obj>, timestamp=1417730308562000)* *Elapsed time: 23 msec(s).*

Figure 20. Retrieved bridge information model from Cassandra

6. CONCLUSION

In this study, a data management framework for bridge monitoring is proposed. We review current developments in data management system for sensor network and bridge information modeling (BrIM). We deploy MongoDB and Apache Cassandra, two state-of-the-art NoSQL database systems. MongoDB is particularly suitable for fast and diverse query performance required for on-site and local (user) computers, while Cassandra is optimized for handling persistent store in the main server. Data schemas are developed to store the sensing data, the sensor information, and the bridge information model. Oliot-EPCIS is adopted for the Cassandra platform to improve performance for storing and retrieving time series data. The schemas for sensor information and BrIM are designed based on standard models, namely SensorML and OpenBrIM, respectively. The proposed data management framework and the data schema are validated by populating the Yeongjong Bridge's information including sampling data, sensor information and building information model. The results show that the proposed schema and system can handle the bridge and monitoring data and allow efficient storage and retrieval by users.

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), Award No. ECCS-1446330. The research team at Stanford University would like to thank Prof. Stuart S. Chen of SUNY at Buffalo for providing the BrIM schema. 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

[1] Lynch, J. P., and Loh, K. J., "A summary review of wireless sensors and sensor networks for structural health monitoring," *Shock and Vibration Digest*, 38(2), 91-130 (2006).
[2] 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., "Overview of a cyber-enabled wireless monitoring system for the protection and management of critical infrastructure systems," Proc. SPIE 7294, 72940L (2009).
[3] Jang, S., Jo, H., Cho, S., Mechitov, K., Rice, J. A., Sim, S., Jung, H., Yun, C., Spencer, Jr., B. F., and Agha, G., "Structural health monitoring of a cable-stayed bridge using smart sensor technology: deployment and evaluation," *Smart Structures and Systems*, 6(5-6), 439-459 (2010).
[4] Koh, H., Park, W., and Kim, H., "Recent activities on operational monitoring of long-span bridges in Korea," Proc. SHMII 2013, 66-82 (2013).
[5] Knobbe, A., Blockeel, H., Koopman, A., Calders, T., Obladen, B., Bosma, C., Galenkamp, H., Koenders, E., and Kok, J., "InfraWatch: Data management of large systems for monitoring infrastructural performance," Proc. IDA 2010, 91-102 (2010).
[6] Zhou, G. D. and Yi, T. H., "Recent developments on wireless sensor networks technology for bridge health monitoring," *Mathematical Problems in Engineering*, 2013, 947867 (2013).
[7] Lim, H. J., Sohn, H., and Liu, P., "Binding conditions for nonlinear ultrasonic generation unifying wave propagation and vibration," *Applied Physics Letters*, 104(21), 214103 (2014).
[8] Lieske, U., Dietrich, A., Schubert, L., and Frankenstein, B., "Wireless system for structural health monitoring based on Lamb waves," Proc. SPIE 8343, 83430B (2012).
[9] Lin, B., & Giurgiutiu, V., "Development of optical equipment for ultrasonic guided wave structural health monitoring," Proc. SPIE 9062, 90620R (2014).
[10] Ray, S. R., "Interoperability standards in the semantic web," *Journal of Computing and Information Science in Engineering*, 2(1), 65-69 (2002)
[11] Cheng, J.C.P., Law, K.H., Bjornsson, H., Jones, A. and Sriram, R., "A service oriented framework for construction supply chain integration," *Automation in Construction*, 19(2), 245-260 (2010).
[12] Lebeque, E., Fies, B., Gual, J., Arthaud, G., Liebich, T., and Yabuki, N., "IFC-BRIDGE V3 Data Model – IFC4," Unpublished Draft Document, BuildingSMART.
[13] Chen, S. S., and Shirolé, A. M., "Integration of information and automation technologies in bridge engineering and management: Extending the state of the art," *Transportation Research Record*, 1976(1), 3-12 (2006).
[14] Samec, V., Stamper, J., Sorsky, H., and Gilmore, T. W., "Long span suspension bridges – bridge information modeling," Proc. IABMAS 2014, 1005-1010 (2014).
[15] Shirole, A. M., Chen, S. S., and Puckett, J. A., "Bridge Information Modeling for the Life Cycle: Progress and Challenges," Proc. IBSMC08-025, 313-323 (2008).
[16] Chen, S. S., Shirole, A. M., Puckett, J. A. Riordan, T. J., and Gao, Q., "BrIM: An Innovative Approach to Managing Bridge Project Delivery," In *1st International Conference on Transportation Construction Management*, Transportation Research Board, Orlando, FL (2009).
[17] Karaman, S. G., Chen, S. S., & Ratnagaran, B. J., "Three-Dimensional Parametric Data Exchange for Curved Steel Bridges," *Transportation Research Record*, 2331(1), 27-34 (2013).
[18] Ali, N., Chen, S. S., Srikonda, R., and Hu, H., "Development of Concrete Bridge Data Schema for Interoperability," *Transportation Research Record*, 2406(1), 87-97 (2014).
[19] Hecht, R., and Jablonski, S., "NoSQL Evaluation: A use case oriented survey," Proc. CSC 2011, 336-341 (2011).
[20] Han, J., Haihong, E., Le, G., and Du, J., "Survey on NoSQL database," Proc. ICPCA 2011, 363-366 (2011).
[21] Strauch, C., and Kriha, W., "NoSQL databases," Lecture Notes, Stuttgart Media University (2011).

[22] Moniruzzaman, A. B. M., & Hossain, S. A., "Nosql database: New era of databases for big data analytics-classification, characteristics and comparison," *International Journal of Database Theory and Application*, 6(4), 1-13 (2013).
[23] McNeill, D.K., "Data management and signal processing for structural health monitoring of civil infrastructure systems," In *Structural Health Monitoring of Civil Infrastructure Systems*, Karbhari, V. M. and Ansari, F. (Eds.), CRC Press, Boca Raton, FL, 283-304 (2009).
[24] Law, K. H., Smarsly, K., and Wang, Y., "Sensor data management technologies for infrastructure asset management," In *Sensor Technologies for Civil Infrastructures: Applications in Structural Health Monitoring*, Wang, M. L., Lynch, J. P., and Sohn, H. (Eds.), Woodhead Publishing, Cambridge, UK, 2(1), 3-32 (2014).
[25] Smarsly, K., Law, K. H., & Hartmann, D., "A cyberinfrastructure for integrated monitoring and life-cycle management of wind turbines," Proc. EG-ICE 2013, 6(30) (2013).
[26] Zhang, Y., Kurata, M., Lynch, J. P., Van der Linden, G., Sederat, H., and Prakash, A., "Distributed cyberinfrastructure tools for automated data processing of structural monitoring data," Proc. SPIE 8347, 83471Y (2012).
[27] Le, T. D., Kim, S. H., Nguyen, M. H., Kim, D., Shin, S. Y., Lee, K. E., and da Rosa Righi, R., "EPC information services with No-SQL datastore for the Internet of Things," Proc. IEEE RFID 2014, 47-54 (2014).
[28] Thantriwatte, T. A. M. C., and Keppetiyagama, C. I., "NoSQL query processing system for wireless ad-hoc and sensor networks," Proc. ICTer 2011, 78-82 (2011).
[29] Scheidgen, M., Zubow, A., and Sombrutzki, R., "HWL—A high performance wireless sensor research network," Proc. INSS 2012, 6240552 (2012).
[30] Eastman, C., Teicholz, P., Sacks, R., and Liston, K., *BIM handbook: a guide to building information modeling for owners, managers, designers, engineers, and contractors*, John Wiley & Sons, Inc., Hoboken, NJ (2011).
[31] Bernstein, H. M., Jones, S. A., and Russo, M. A., "The business value of BIM in North America: multi-year trend analysis and user ratings (2007-2012)", *SmartMarket Report*, McGraw-Hill Construction, Bedford, MA, 9-14 (2012).
[32] Kivimäki, T., and Heikkilä, R., "Bridge information modeling (BrIM) and model utilization at worksites in Finland," Proc. ISARC 2010, 505-513 (2010).
[33] Marzouk, M. M., and Hisham, M., "Bridge information modeling in sustainable bridge management," Proc. ICSDC 2011, 457-466 (2011).
[34] "OpenBrIM," [Online]. Available: http://openbrim.org/ [Accessed: Jan. 2015].
[35] Macedo, T., and Oliveira, F., "Redis cookbook," O'Reilly Media, Inc. (2011). *Safari Book Online*. [Accessed: Dec. 2014].
[36] "MongoDB," [Online]. Available: http://www.mongodb.org/ [Accessed: Dec. 2014].
[37] Chodorow, K., "MongoDB: the definitive guide", O'Reilly Media, Inc. (2013). *Safari Book Online*. [Accessed: Dec. 2014].
[38] "Application never before possible," [Online]. Available: http://www.mongodb.com/use-cases [Accessed: Jan, 2015].
[39] "How a database can make your organization faster, better, leaner: examples and guidelines for the enterprise decision maker," *A MongoDB White Paper* (2013) [Online]. Available: http://dws.la/wp-content/uploads/2014/12/MongoDB_Better_Faster_Leaner.pdf [Accessed: Jan, 2015].
[40] "Chicago's WindyGrid: using Mongodb to create a smarter and safer city," [Online]. Available: http://www.mongodb.com/customers/city-of-chicago [Accessed: Jan, 2015].
[41] "IoT and Big Data: A Joint Whitepaper by Bosch Software Innovations and MongoDB," *A Bosch Software Innovations and MongoDB White Paper* (2015) [Online]. Available: http://info.mongodb.com/rs/mongodb/images/MongoDB_BoschSI_IoT_BigData.pdf [Accessed: Feb, 2015].
[42] "Cassandra," [Online]. Available: http://planetcassandra.org/ [Accessed: Dec. 2014].
[43] Hewitt, E., "Cassandra: the definitive guide," O'Reilly Media, Inc. (2010). *Safari Book Online*. [Accessed: Dec. 2014].
[44] Branson, R., "Facebook's Instagram: Making the switch to Cassandra from Redis, a 75% 'Insta' savings," [Online]. Available: http://planetcassandra.org/blog/interview/facebooks-instagram-making-the-switch-to-cassandra-from-redis-a-75-insta-savings/ [Accessed: Jan, 2015].
[45] Branson, R., "Cassandra at Instagram 2014," In *Cassandra Summit 2014*, San Francisco, CA (2014).
[46] "eBay engages customers with personalized recommendations," [Online]. Available: http://www.datastax.com/wp-content/uploads/2012/12/DataStax-CS-eBay.pdf [Accessed: Jan, 2015].

[47] "Netflix gives users exactly what they want – every time," [Online]. Available: http://www.datastax.com/wp-content/uploads/2011/09/CS-Netflix.pdf [Accessed: Jan, 2015].
[48] "Oliot-EPCIS," [Online]. Available: http://oliot.org/ [Accessed: Dec. 2014].
[49] "SensorML," [Online]. Available: http://www.opengeospatial.org/standards/sensorml [Accessed: Dec. 2014].
[50] Botts, M., Robin, A., "OGC SensorML: Model and XML Encoding Standard," Open Geospatial Consortium (2013).
[51] Chen, S. S., Hu, H., Ali, N., Srikonda, R., Shirole, A. M., "From BIM to BrIM: Plus, Minus, Delta," In *NIBS/TRB Workshop* (2014).