Bridge Sensor Mart: A Flexible and Scalable Data Storage and Analysis Framework for Structural Health Monitoring

Nazif C. Tas & Ciprian Raileanu & Mathaeus Dejori & Claus Neubauer
Siemens Corporate Research, Princeton, NJ

ABSTRACT: This paper reports on the new sensor data collection, analysis and storage framework, Bridge Sensor Mart (BSM), for bridge health monitoring data. BSM defines a distributed data storage and data analytics infrastructure in order to collect, store, analyze and manage sensor data for Structural Health Monitoring applications.

1 INRODUCTION

Continuously monitoring or even real-time forecasting the performance of civil infrastructures based on gathered sensing information offers a tremendous opportunity to increase safety by detecting and localizing damage before it reaches a critical level. Such monitoring and analysis help reduce costs by shifting from current "run-to-failure" or preventive maintenance practices towards predictive maintenance strategies. Additionally, structure engineers can use the collected data for retrospective analyses in order to better understand structural behaviors and their interrelationship with external factors such as traffic, load or environmental influences.

Federal and industrial interest in Structural Health Monitoring (SHM) projects have been significantly increased in the past decades, also fueled by the progress made in the area of sensor networks and information technology. Smart sensor networks, distributed over disparate structures and communicating wirelessly with each other have been proven to be effective in many studies, and consequently have been deployed on signature structures. Kim et. al (2007) for example designed and implemented a wireless sensor network to monitor the 4200 ft long main span and the south tower of the Golden Gate Bridge. See Ansari (2005) for many more examples of SHM applications.

Albeit the tremendous progress made in new sensing and networking technologies, one of the biggest challenges remains: How to efficiently collect, store and provide the accumulated data to the end users and applications for further analysis? SHM applications are gathering large amounts of data that are impossible to manage with the current systems, and hence making the dissemination and sharing of such data very cumbersome or even impractical. In this work, we present a new data storage and analysis framework, Bridge Sensor Mart (BSM), designed specifically for long term SHM.

The key features of this framework are:

1. Distributed Data Analytics: The post-processing of incoming data is usually performed on a single computer. The BSM framework allows distributing data analysis and data processing procedures across multiple machines allowing users to handle large amounts of data.
2. Distributed Data Storage: BSM also supports the distributed storage of accumulated data. This distributed architecture allows the system to be extended to handle larger scales of data by simply adding additional hardware components without changing the design or codebase.
3. Flexible Sensor Support: Current data management systems are designed to support specific sensing hardware or certain vendors. The BSM architecture is sensor- and provider agnostic. The data import layer provides a generic interface that can be extended through a plug-in mechanism to support new sensor hardware.
4. Flexible Data Analytics Support: BSM provides a generic interface for users to plug-in their own data analysis algorithms, and to extend the analytical capabilities without requiring any changes in the system.

BSM is based on a very efficient hybrid database structure which combines and utilizes two kinds of databases: a scientific database for storing large

amounts of historical sensor data and a regular row-oriented databases for storing metadata. Simulations performed showed that the BSM database structure considerably outperform traditional databases for sensor-related data tasks.

Overall BSM allows to robustly manage large amounts of data, by implementing a fully distributed architecture for data gathering, analysis and dissemination. Thus, a single point of failure is avoided and the workload is distributed among several computers. This architecture promises high scalability, as it is straightforwarded to extend the system by simply deploying new hardware (e.g. new server nodes) in order to lessen the pressure on highly loaded computers.

2 BACKGROUND

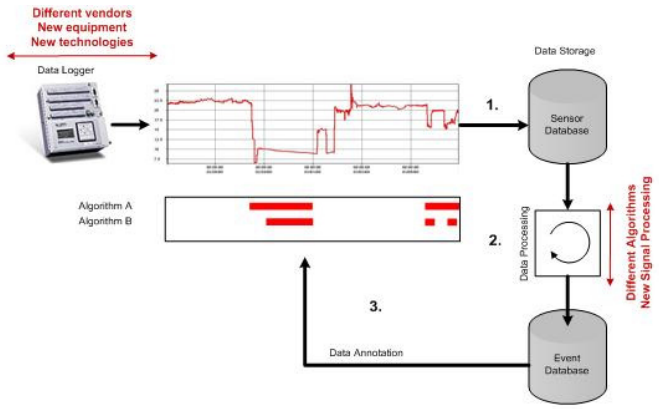

Figure 1 Data flow of a typical long-term monitoring scenario.

As shown in Figure 1, a possible workflow for long-time monitoring can be decomposed into three major steps[1]. First, the data are collected from a dedicated hardware device (such as a data logger) and stored into a database. Secondly, the stored data are constantly analyzed through different analysis methods, such as outlier detection or state estimation models. Based on the results of the analysis methods the data are labeled accordingly and events are generated. For instance, to mark intervals where a certain error has been observed or to trigger events when a certain threshold has been exceeded. Such labels are finally stored as events and aligned to the corresponding sensor signal through unique ids and a time interval.

The architecture shown in Figure 1 is sparsely scalable since all the data are gathered, processed and stored by single machines making it impossible to scale up with growing data. Especially when deployed as part of long-term SHM settings, IT frameworks have to provide a scalable architecture in order to cope with the growing amount of data. This paper describes a data collection and data analysis framework that overcomes the limitations of existing IT solutions by providing an architecture that enables users to handle large amounts of data by distributing algorithmic components as well as the data across different machines (shown in Figure 2)

Similar existing frameworks are vendor-specific and therefore not opened for collecting data from the sensing hardware of different providers, and most notably not flexible enough to connect to sensing technologies not seen so far. Furthermore, existing frameworks do nott allow the creation and modification of novel analysis algorithms.

BSM implements a plug-in interface that allows for the integration of a variety of different sensor types, technologies and data analysis methods. This approach enables users to extend the system based on their individual requirements with customized data acquisition and data analysis methods.

In the rest of this document, the BSM architecture components and the underlying principles are discussed and various performance results are provided.

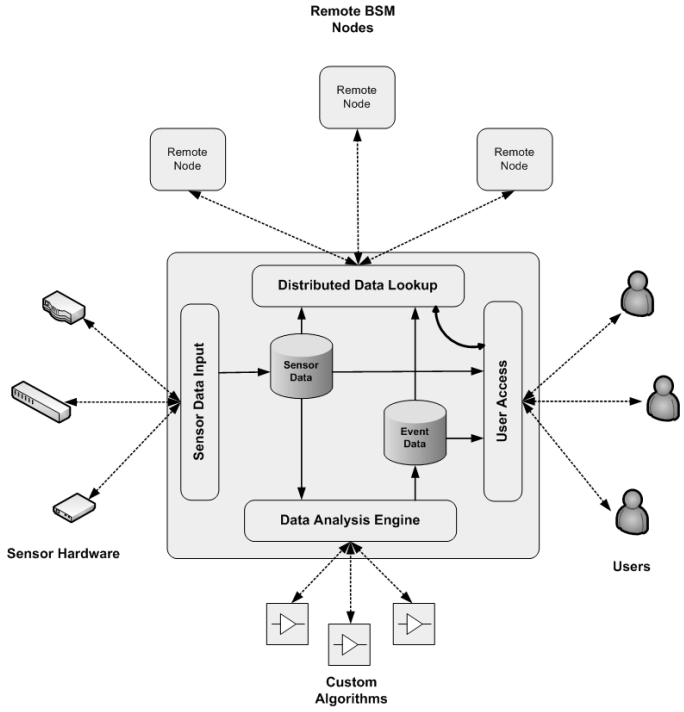

Figure 2 Bridge Sensor Mart system architecture.

3 DISTRIBUTED ARCHITECTURE

Just as a common typical SHM application, the BSM system is expected to collect, store and process vast amounts of data from sensing devices located in diverse physical locations with possibly high sampling frequencies. As the number of data sources and the number of users increase, a centralized IT solution poses serious scalability problems. The

[1] Note that Figure 1 depicts one among many different workflows and does not cover all aspects of SHM.

BSM system defined in this document overcomes these kinds of issues by utilizing a distributed peer-to-peer infrastructure, as shown in the top part of Figure 2, and thus allowing convenient extensibility of the entire system through addition of new computers, so called *BSM nodes*.

A BSM node is a logical entity which supports whole or partial functionalities of data collection, storage, processing and data access. BSM nodes are organized in a peer-to-peer fashion, enabling fast data discovery and access. In this system, each BSM node is responsible for collecting the data locally from one or more sensing devices. The collected data are stored and processed locally and the metadata defining the nature of the fetched data together with the outcomes of the analysis engine is dissipated among the other BSM nodes through the peer-to-peer infrastructure for data lookup and remote access.

When the users are interested in a particular type of information and issue a query to a target node, firstly the query is performed on the local database searching for immediate availability. If it is possible to respond to the query entirely, the information is fetched from the local database and made available to the user. If the information needed to respond to the query is distributed among other BSM nodes, the target node first discovers the locations of the other BSM nodes needed to complete the query response, through the peer-to-peer infrastructure. Once the locations of the BSM nodes are discovered, the target node communicates with them separately by transferring the initial user query and collecting the associated responses. Finally, the target node merges all the responses, including its own, together and replies back to the user as a single response.

Notice that in this structure all the details of the distributed data lookup and access are handled by the BSM system and the user interacts with the system in a transparent way, as if s/he is interacting with a single centralized system. In this fashion, as the platform storage capacity is extended by adding new hardware, the system automatically registers the newly accessible components as BSM nodes enabling them to be discoverable among the other BSM nodes. Hence, the newly registered data is made available immediately to the users with no effort required on the user side. Using the BSM peer-to-peer structure, the users can access a diversely located distributed system through a single point of entry and query the entire system as if they are querying a single database.

4 DATA STORAGE ARCHITECTURE

Storing and processing the sensor data is a challenging database task, as the amount of data stored is usually extremely large (sometimes millions of sensor readings per day) with a large variable update frequency range (from "once a day" update to 100Hz readings). In addition to the sensor readings, the BSM system stores several other types of data structured in variety of ways such as the sensor inventory information stored for hardware management, the user information stored for secure system access and authorization, and metadata information for storing the steps of data transformation during the data import/analysis processes. Hence, BSM database structure faces the challenges of storing heterogeneous data with diverse set of requirements.

In order to tackle this problem, BSM system deploys an efficient, fast hybrid database structure which promises the fast querying speed of scientific databases without the insertion bottleneck common in such systems. BSM utilizes two databases at the same time and optimizes the queries issued for the maximum performance.

4.1 *Column-oriented databases vs. Row-oriented Databases*

Traditional databases implement a record oriented database schema where the records are stored continuously. Through this row-oriented storage mechanism, new records (data insertion) can be handled very efficiently by pushing all the record information into the storage system. Hence this kind of database systems is usually referred as write-optimized or row-oriented databases (Stonebreaker et al. 2005).

On the contrary, where arbitrary operations requiring multiple records to be accessed are common, databases should be read-optimized. In such databases, data is usually updated periodically (and relatively infrequently) in bulks and most of the time accessed through read operations. Thus, for these kinds of databases, in contrast with the row-oriented schema, a column-oriented schema, where each column is stored contiguously, is shown to be more efficient (Stonebreaker et al. 2005). This kind of schema is used widely for fast, real-time analysis on vast amounts of data See KX (2009) for an example on handling financial data from stock exchange.

BSM system deploys a traditional Oracle database for the inventory data given the widely accepted industrial usage and reputation of this database system. For our sensor data, we considered two well-known scientific databases available: LucidDB and MonetDB.

LucidDB is an open-source, column-oriented database whose goal is to provide flexibility in high-performance data integration and efficiency in sophisticated query processing. LucidDB promises these functionalities through the column-oriented structure, bitmap indexing, automatic hash join/aggregation and page-level multiversioning. LucidDB automatically identifies the index needed

for the tables and creates the necessary indices on the fly yielding optimal data compression, reduced I/O, and fast evaluation of Boolean expressions, without the need for database administrator to choose the index type to use.

MonetDB is a free column-oriented storage system with memory-based internal data representation and CPU-tuned vectorized query execution. MonetDB also uses automatic and self-tuning indices for run-time query optimization

In order to compare the performance of these two column-oriented databases under realistic conditions, we performed several experiments that we report about in the next sections. Our findings suggest that MonetDB performance is superior to LucidDB for the BSM data access and storage needs.

The BSM system deploys a hybrid database structure where sensor data is stored using the scientific MonetDB database, while the rest of the data is stored using the traditional Oracle database. The motivations for these choices are reported through the rest of this section.

4.2 *BSM Database Performance Benchmarks for Sensor Data*

In order to see which scientific database to utilize for BSM sensor data storage, we compiled the following experiment: we installed both databases together with the traditional Oracle database on a PC with 2.33GHz Intel Core 2 CPU and 2GBs of RAM. We created five test files with different number of records. The size of the test files and the number of records in each test file is shown in Table 1.

Table 1. Test size for scientific database experiment.

Test cases (i.e. x-axis)	Tuple Size	File Size(B)
2	$41*10^2$	211K
3	$41*10^3$	2M
4	$41*10^4$	19M
5	$41*10^5$	220M
6	$41*10^6$	2.3G

For each of the databases, we loaded each of the test cases separately and issued three SQL statements in order to quantify their performance in terms of data access. The three test cases we performed are summarized in Table 2. Each test case is repeated 10 times and the average of the benchmark values are reported.

Table 2. Test statements for the scientific database experiment.

Name	SQL Statement
Data Retrieval	select * from sensors where sensorid = 'RPM' and (readingdate < '2008-01-01' and readingdate > '2004-01-01')
Data Aggregation	select avg(readingvalue) from sensors where sensorid='RPM'
Data Range	select sensorid from sensors where readingvalue > 0.1

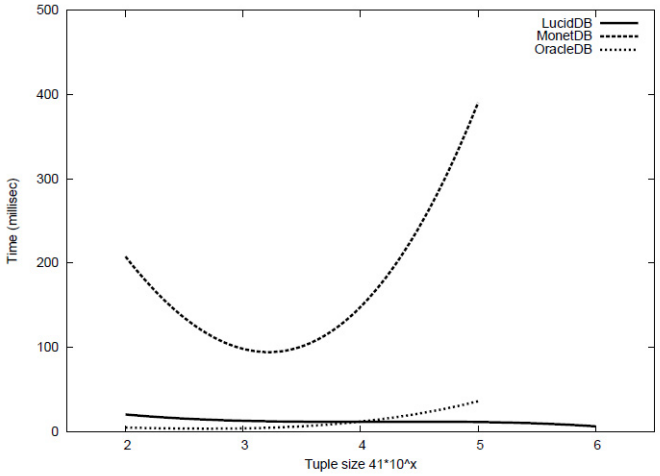

Figure 4. Performance benchmarks for data retrieval query.

Figure 4 illustrates the results for the first test of data retrieval, in which Oracle and LucidDB perform similarly, while MonetDB performs the worst for getting the sensor information in a specific time interval. Notice that in this statement, all the sensor information is gathered and there is no additional operation done on the result set.

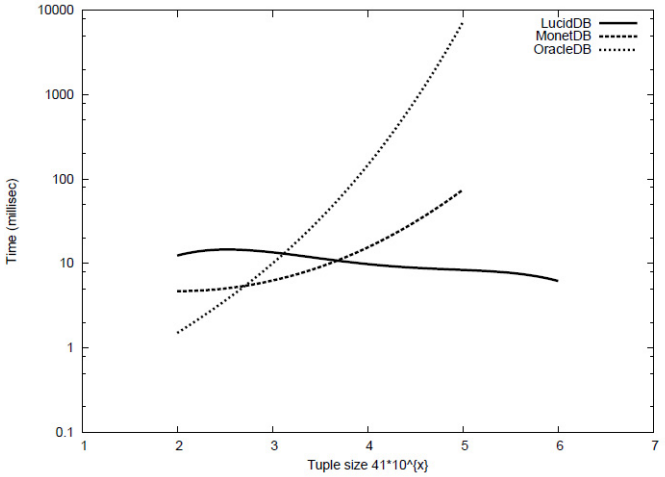

Figure 5. Performance benchmarks for data aggregation query.

Figure 5 illustrates the results for the next test of data aggregation which issues a more common operation of getting the average values of a specific sensor type. Notice that the y-axis on the figure is logarithmic. LucidDB and MonetDB, with their column-oriented structure optimized for such a column operation, perform much better than the row-based Oracle database, as it can be seen from the figure. As the size of the test increase, Oracle starts to perform worse and worse. In fact, for the 6th test size, Oracle did not stop for several hours and the query had to be manually stopped without completing.

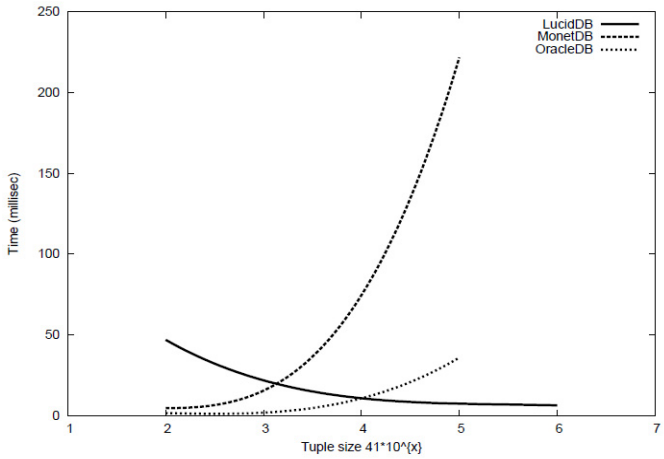

Figure 6. Performance benchmarks for data range query.

Figure 6 illustrates the third and the last test case of data range query in which a single column (the sensorid) is returned and the results are filtered on the reading value of the sensor. The results are very similar to the data retrieval and LucidDB performs the best, Oracle second and MonetDB the worst.

These results suggest that the scientific databases LucidDB and MonetDB both perform very well for column specific operations such as taking the average of a column of a database table with their quick data access promise through column-oriented storage. Even though Oracle is comparable in performance for the data aggregation and data range tests, in the BSM system, scientific databases will be preferred for the sensor data storage since the column-wise operations are very common in analyzing the sensor data as in the case of data aggregation query.

When the results of the tests performed are compared, LucidDB outperforms other databases considerably. However, it should be noted that the tests performed so far focused on the data query performance only and did not consider the cost of adding new data to the system, i.e. data insertion. In our next experiment, we measured the time of loading the dataset #5 which is, we believe, a reasonable upper bound for daily upload in order to see how the databases perform for the task of data insertion. MonetDB took 2 minutes to load the 220MBs of data whereas, because of its more comprehensive indexing mechanism, LucidDB took more than 30 minutes to load.

Because of the impressive benchmark results for queries containing column-oriented operations and reasonable insertion statistics for data upload, MonetDB is chosen as the database to deploy in the sensor data storage architecture.

In order to perceive how realistic the insertion performance of MonetDB was, we executed a final experiment comparing the row-oriented Oracle database and MonetDB for insertion operation. Our experiment consisted of pushing insertion SQL statements into the database one by one and auto-commit after each of the insertion operation. Notice that this experiment is slightly different than the data load experiment where the data is committed after all the data is loaded into the system. This experiment was conducted on a rather more powerful PC with Intel Xeon 2GHz CPU with 8 cores and 16GBs of RAM.

Figure 7 illustrates the results for the insertion experiment. As can be seen, MonetDB performs better than the row-oriented Oracle database, suggesting a good match for insertion operation as well.

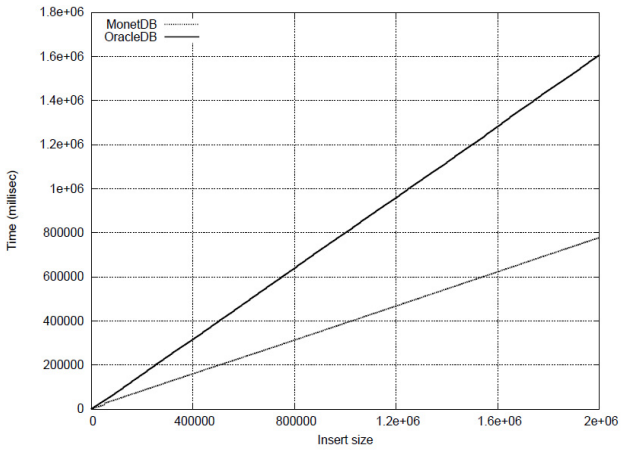

Figure 7 Performance benchmarks for the insertion test.

Following the outcomes of the experiments reported above, BSM deploys a hybrid database structure consisting of two different types of databases: First, a scientific database is utilized in order to store the sensor readings gathered directly from the sensing devices in order to support fast aggregation queries which are the common use cases for this type of data; and second, a traditional database is utilized for storing the data that is commonly accessed through regular data retrieval or range operations, e.g. sensor inventory and historical data. The BSM database structure utilizes both of these database types at the same time and the queries submitted by

the users are automatically transferred to the related database components in real-time.

5 USER ACCESS METHODS

As the BSM system gathers the data registered from different data sources, the users interested in the collected sensor data can communicate with the system directly using the BSM data access methods as shown on the right side of Figure 2. The BSM system supports three methods for processing and accessing the sensor data, namely browsing, monitoring and processing.

5.1 *Browsing*

The users can browse and fetch the available data stored through the associated data access tools. Notice that the browsing functionality targets the data that are already stored in the database and does not require any state maintenance for the system, i.e. no state of the user is kept and all the requests are assumed to be independent. Example queries for this method are getting the sensor data gathered in a specified time interval, getting the sensors instrumented on a specific physical item, getting the average reading of a specified sensor, getting vendor information on a specific sensor, etc.

5.2 *Monitoring*

In addition to browsing in historical data users can request constant monitoring on specific sensors to be constantly notified on newly available data. The users initiate interest on a specific sensor by calling the associated methods which insert their subscription information into the system. Each subscriber is given a subscription ID for future use and easy access. As the data is fetched from the data sources, it is cross-analyzed with the subscription information and if subscribers interested in new data are found, they are automatically notified. The subscriber notification is designed as a pull mechanism rather than a push mechanism where the latter requires open connections all the time. Using the former mechanisms, the subscribers can poll for newly available data for their subscribed sensors as frequently as they desire. The pull mechanism lessens the load on the server by giving more responsibility to the clients. An example query for subscription method is getting the new data for a specific sensor as it is being fetched.

5.3 *Processing*

The users have the freedom of defining their own data processing algorithms according to their needs and capabilities. These data processing algorithms search for specific data patterns in the incoming data and register the patterns into the system as they have discovered as so-called *events*. In this way, the raw sensor data is transformed into semantically meaningful information. As the data is coming in, similar to the monitoring method, it is being analyzed through the user-defined data processing algorithms (i.e. event descriptions) as can be seen in the bottom section of Figure 2 and if matches are found, events are stored into the database. Using this method, the users only gets notified in the case of an event, triggered by a custom algorithm, rather than the raw sensor data themselves. An example query for the events method is the following: a user is monitoring a specific temperature sensor, wants to get notified if the sensor reading values exceeds a threshold of 85 degrees F indicating a potentially hazardous condition. In that case, the user is not interested in the values that are below the defined threshold. Instead, based on the defined algorithm the user only gets notified in the event of interest. This approach also minimizes the data transfer as all the computations and analytics is handled on the server side. Using the BSM plug-in architecture defined above, the users can create any arbitrary algorithm and register it to the system conveniently. The BSM system defines a very generic plug-in architecture through a standard producer/consumer model which supports an open registration and integration. This generic plug-in mechanism enables users to create complex analytical pipelines by connecting several algorithmic workflows.

.

Overall BSM provides users with three basic access methodologies use according to their needs and capabilities. The users interested in historical data might choose to browse the already available data whereas the users who are interested in the most recent data might subscribe to the sensor of interest through monitoring and fetch the data as it becomes available. Finally, if the users are interested in more refined results gathered from sophisticated algorithms and functions, they might create and register their own data processing algorithms and subscribe for their events of interest.

6 SOFTWARE ARCHITECTURE

The BSM framework provides a generic framework for fetching, storing and analyzing various kinds of sensor data. The distributed databases discussed in the previous sections are queried and populated by a corresponding distributed software layer consisting of several object-oriented components. One of the core platform components is the *Scheduler* component. The Scheduler allows for scheduling and execution of various jobs at predefined time intervals. The job-specific logic can be implemented as object-

oriented software components (classes), and the scheduling interval can be as granular as needed, for example a job can be executed every other minute, three times a month, or twice every month from May to September.

Such a scheduling support is essential for acquiring the raw data from different sensor data loggers as seen on the left section of Figure 2. In a typical scenario the data is fetched (pulled) from an array of sensors at regular time intervals, and then distributed (pushed) throughout the BSM system using a producer-consumer model. A producer-consumer model consists of several processes (consumers and producers) that pass data among them either directly (piping), or via a broker as outlined in the coming paragraphs. The BSM platform allows for a large number of consumers and producers to be wired together transparently, allowing for a great flexibility and modularity, as well as an increased data throughput. Furthermore, chains of consumers and producers can be created, allowing for complex workflows in which a component can act concurrently as consumer to its predecessor and producer to its successor.

A producer will typically access a data source using a Scheduler and pull the latest sensor data. Once the data is acquired from the sensor loggers, it is distributed to a set of Consumers interested in a particular class of data. The Producers are pushing the data via a Message Broker, which allows for the producers and consumers to be loosely coupled, therefore increasing the system scalability and modifiability. A Message Broker is a specialized software component that acts as an intermediate layer between producers and consumers, handling the message sources, destinations, and payloads.

A simple Consumer may for example just receive the data from the Producer via the Message Broker and then save it into a database table for further analysis in value-timestamp pairs. In a second scenario, a Consumer may, for example, analyze the data for particular events (e.g. the sensor readings being above or below a predefined threshold), or for a sequence of events (e.g. the sensor readings are slowly decreasing over time). Finally, a third Consumer may distribute the data into fact and dimensions tables, and notify its successor Consumer about the creation of the tables. Upon receiving the notification, the successor Consumer can create an OLAP hypercube for fast multidimensional queries, time series analysis, data slicing and visualization.

The BSM platform allows one to focus on the sensor and event-specific business logic, while providing a robust framework for scheduling, data flow, storing and data visualization. Furthermore, the core framework is built around the distributed computing paradigm; all the core components (Scheduler, Message Broker, Consumer and Producer) can be distributed across a cluster of machines, allowing for large amounts of data to be processed efficiently. - Finally, all the configuration of the software components, like job schedules and producer-consumer workflows, can be conveniently specified and easily modified using an XML file.

7 PLUG-IN ARCHITECTURE

The architecture proposed herein overcomes the limitations of current SHM frameworks by providing two plug-in interfaces shown.

The plug-in interface on the data input side (on the left side of Figure 2) provides a flexible and easily extendible data gathering mechanism which supports fetching the data stored in vendor-specific locations and devices, such as the dedicated data logging devices referred as the data-loggers for obtaining in-field sensor data or proprietary database systems, etc. BSM data gathering mechanism supports periodic data fetching by automatically getting the data within the predefined time intervals. Using the BSM data gathering mechanism, adding a new data source to the system is as easy as defining the necessary components and registering it with the rest of the structure. Hence, different types of sensors from different vendors can coexist and run efficiently in the BSM infrastructure.

A second plug-in interface depicted at the bottom of Figure 2 enables the users to plug-in customized data analysis tools into the system conveniently, thus extending the algorithmic power of the system without changing the underlying framework or code base. Moreover, using this infrastructure, each of the data analysis components defined by the users are notified only by the data they are interested in, thus the size of the data to be analyzed is reduced significantly. With this infrastructure, the users can easily register the components they implement into the system and observe the results provided through the events framework.

8 CONCLUSION

This document describes the architecture of a data framework specifically designed for long term SHM. The so-called Bridge Sensor Mart (BSM) infrastructure is intended for efficient collection, storage, analysis and management of data gathered from sensing devices. The proposed architecture is particularly aimed at solving the data challenges faced in Structural Health Monitoring applications. With its modular and extendible design, the BSM system promises the users the ability to plug-in their own sensors as data sources as well as their own data analysis algorithms as data processing workflow steps.

BSM system envisions a fully distributed architecture where the gathered data can be stored and accessed in an extremely fast and transparent way. With its flexible yet resourceful design paradigm, we believe that the BSM framework can be conveniently used in other potential applications such as traffic and environmental monitoring.

9 REFERENCES

Ansari F.. Sensing issues in civil structural health monitoring. Springer, 2005.

Kim S., Pakzad S., Culler D., Demmel J., Fenves G., Glaser S., and Turon M. Health Monitoring of Civil Infrastructures Using Wireless Sensor Networks. *In the Proceedings of the 6th International Conference on Information Processing in Sensor Networks (IPSN '07)*, Cambridge, MA, April 2007, ACM Press, pp. 254-263.

LucidDB webpage http://www.luciddb.org/.

MonetDB webpage. http://monetdb.cwi.nl/.

Stonebreaker, M. et al. C-Store: A column-oriented DBMS, Proceedings of the 31st VLDB Conference, 2005.

The kdb+ Database. A unified database for streaming and historical data, KX Whitepaper. 2009.