I.J. Modern Education and Computer Science, **2017, 9, 32-41**
Published Online September 2017 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijmecs.2017.09.04

A Real-time DBMS System for the Immigration Processing of Large Hajj Crowd

Amir A. Khwaja
King Faisal University, Al-Ahsa, 31982, Kingdom of Saudi Arabia
Email: akhwaja@kfu.edu.sa

Received: 20 March 2017; Accepted: 22 August 2017; Published: 08 September 2017

***Abstract*—**Hajj is an important Islamic ritual and one of the five pillars of Islam. The Hajj event occurs in the twelfth month of the Islamic lunar calendar and requires anywhere from two to three millions of Muslims from all over the world to make pilgrimage for 10-15 days to the Holy city of Makkah in the Kingdom of Saudi Arabia. Providing quality Hajj services to such large number of pilgrims has been a significant challenge for the Saudi Arabian authorities. Among other services, immigration processing of a large Hajj pilgrim crowd arriving simultaneously at various Saudi Arabian ports during the specific Hajj days has resulted in significant delays at these ports. Unique and technology based solutions must be explored to alleviate the various Hajj related pilgrim service problems and to improve overall quality of these services. This paper reports experience with the design and development of a prototype backend DBMS system to automate the immigration processing of the large Hajj crowd. A real-time DBMS is considered for meeting the processing requirements of such a large Hajj pilgrim crowd arriving simultaneously at various ports. The purpose of this prototype was to understand the challenges and the feasibility of implementation of the backend system using a real-time DBMS.

***Index Terms*—**Large crowd management, Hajj, Real-time DBMS, Sensei Database.

I. Introduction

Hajj is one of the five pillars of Islam. It has a set of acts of worship to be performed in and around the city of Makkah in the Kingdom of Saudi Arabia at least once in a lifetime by every Muslim who satisfies certain conditions. The nature of today's Hajj requires substantial planning and effort to support and to facilitate these religious rites. It is one of the world's largest annual events. More than two million pilgrims from approximately 200 countries gathered for Hajj in 2015 [1]. The 2030 Vision of Saudi Arabia [2] confirms more than three million pilgrims for Hajj by the year 2030.

An important challenge facing Saudi Arabian authorities during this event is to facilitate the information processing of the large number of pilgrims. With such large number of pilgrims arriving simultaneously at multiple airports and shipping ports in Saudi Arabia for performing Hajj every year, these embarkation ports need mechanisms for efficient traveler management and to speed up pilgrims' information processing to avoid significant delays at the airports which usually takes several hours for immigration and other formalities [3]. These delays consist of several components and while not all components may be completely avoidable, an attempt can be made to eliminate or significantly reduce some of these delay components. One such delay component is manual immigration processing of each Hajj traveler at the airport. Since no specific data is available from the Ministry of Hajj Web site [4] on the details of these delays, interviewing some of the immigration personnel at the Riyadh International Airport, Capital of Saudi Arabia, revealed that such immigration processing per person during Hajj season may take anywhere from 35-40 minutes.

Improving the quality of Hajj service is one of the primary concerns of the local authorities and need special kind of planning, analysis, design, and technology [5]. Studies have indicated that the quality of services provided to travelers at King Abdulaziz International Airport, Jeddah, Saudi Arabia, one of the largest and most crowded airport during the Hajj season due to its closeness to the city of Makkah, shows that 50% of travelers were not satisfied about the services at the airport and they found that there is a gap between perceptions and expectation [5]. Different technologies facilitating management of this kind of large crowd event should be considered such as using smart cards and radio-frequency identification (RFID) based identification cards, high-end client and server machines, and special kind of backend databases such as real-time or distributed databases to reduce such delays and to improve quality of service.

The use of smart card technology may be explored for making the Hajj immigration process more efficient. Smart card based Hajj identity cards may be issued to the pilgrims with all the relevant information stored on the smart card. Card readers at the airports and shipping ports can read pilgrim data from the smart card and verify by accessing and comparing with golden data stored in the backend Database Management System (DBMS). These transactions must be performed in a very short time, perhaps in milliseconds, so as to allow efficient

processing between when a pilgrim swipes his/her card to the notification of data confirmation from the backend DBMS to the immigration authorities at the port. Use of a smart card based Hajj automatic permit verification will eliminate manual immigration processing and may significantly reduce such processing delays.

This paper reports experience with the design and implementation of a prototype backend DBMS system for meeting the processing requirements of a large crowd of Hajj pilgrims arriving simultaneously at various ports. The purpose of this prototype was to understand the challenges and the feasibility of implementation of the backend DBMS using a technology that can satisfy the project requirements.

The rest of the paper is organized as follows: Section II presents and evaluates some related research work. Section III provides high level Hajj backend DBMS requirements. Section IV evaluates various DBMS technologies to best meet the Hajj large crowd management requirements. Section V goes over the prototype implementation details. Section VI highlights some issues and challenges during the prototype implementation and how these issues were addressed. Section VII concludes the paper.

II. Related Work

This section provides a summary of similar or related research work in the area of Hajj crowd management.

Mitchell et al. suggest crowd management using an RFID and mobile device/network based solution [6]. The main focus of their work is pilgrim detection in a crowd for the purposes of understanding and controlling overcrowding, identifying the pilgrims' location in the case of emergencies, tracking lost family and friends, and authorities sending emergency and other alerts to the pilgrims. Mitchell el al.'s solution requires strategic placement of RFID readers at key Hajj locations with defined zones. When a pilgrim comes into that zone, the reader detects and transmits the location to the central controller. However, the authors reported several issues and challenges in using RFID in a large crowd settings such as limited range of RFID readers, pilgrim self-blocking RFID signal, incorrect angle of RFID device for the reader, and the most challenging to determine the optimal placement of readers in a terrain type of landscape with mountains and other obstacles. Most of the work reported by this research is conceptual with limited prototyping and testing. No apparent solutions to the issues and challenges reported were presented and discussed with any technical details. The backend storage solution suggested is an Oracle database. However, no details are provided and the reported work also did not address issues of efficient storage and accessing of pilgrim data, especially in the case of time sensitive emergency events, from a relational database that may present challenges for a database consisting of millions of records.

Mohandes et al. propose an RFID based wristband solution for pilgrims [7]. Their primary use cases consist of crowd control in specific Hajj locations by placing readers that can monitor flow of crowd and help the security personnel in controlling the entry of pilgrims in high risk areas. Some of the other use cases consist of identification of pilgrims by storing basic personal and official documentation information especially in case of deaths and people getting lost, medical conditions stored on the tag for emergency situations, and detection of official Hajj permission stored on the tag from a distance to determine if a group of pilgrims have all permission to perform Hajj. However, as discussed by Mitchell et al., the wristband based RFID is not quite effective due to pilgrim potentially self-blocking the signals [6]. In addition, placement of card readers at optimal locations for crowd control is a significant challenge as well as the challenge of limitation in signal detection ranges [6] that may not allow a possible practical solution. A limited prototype Visual Basic based system is implemented to demonstrate pilgrim identification use case where the reader is connected to the PC running the VB based user interface application via a serial port. The prototype implementation has only unique identification (UID) stored on the tag and the pilgrim details are stored in a local database. Mohandes et al. suggest that the actual implementation will have all pilgrim information stored on the tag and not retrieved from a database. However, there are challenges in determining type and format of information to be stored on the tag due the limitations in RFID storage as well as security concerns for various stored critical information about the pilgrim. The paper does not address and discuss any of these challenges.

Alsaggaf et al. [8] build their work on earlier proposals on Hajj management using RFID technology with security features [7] and extended it by introducing the fingerprint biometric approach that is unobtrusive and difficult to steal and forge. Their main focus is improving and increasing the pilgrim identification authentication process. They suggest integrating the fingerprint biometric with the public key infrastructure (PKI) algorithms to provide a robust identity authentication. Their solution suggests RFID based wristbands for pilgrims. However, as was pointed out by Mitchell et al. [6] that the experimentations demonstrated that wristband based RFID are not effective as the wearer of the wristband may block the signals to be detected by the RFID reader. Alsaggaf et al.'s solution depends on the use of an enrolled biometric template, which is held on a locally or centrally formed database with the authentication of person usually undertaken through one-to-one comparisons of his/her identity to the stored templates. There is no indication from the paper about any real or a prototype system implementation and it appears from the paper that this is a conceptual model.

The Pilgrim Smart Identification (PSI) system is also developed using the RFID technology [9]. Similar to the other systems, the intent of this system is to store personal pilgrim information and medical records in case of emergencies, coma, and other critical conditions. Geabel et al.'s main intent is to overcome language barriers for pilgrims and Hajj workers by storing the

necessary information in an RFID used in a bracelet worn by the pilgrims. The system uses two applications one on a personal computer for the administrators and workers and the other for the card reader to write tag ID and to read necessary pilgrim information from the tag. The system also uses two corresponding DBMS, one for each corresponding application, based on the SQL (Structured Query Language) Server 2005 and SQL Compact version. No security features are used to protect critical pilgrim information and no communication mechanisms are included other than the card readers. No discussion is provided for the performance of the relational backend DBMS in the case of large Hajj crowd processing.

Based on the related work review above, the bulk of the Hajj research is related to large crowd management which consists of using RFID based technology for monitoring, controlling, and reporting large crowds. However, as can be seen from the above related work review, this task is quite challenging as there are several factors related to the logistics, physical limitations, and the current RFID technology limitations that may severely impede the practical application and deployment of these solutions. As such, most of the above research is conceptual solutions with none to minimal actual implementation. Moreover, none of the above work has addressed the backend DBMS processing challenges for such large crowds. The reported research work in this paper, in contrast, is suggesting smart card based automated point solutions for various pilgrim related use cases. These point solutions are practical from both physical and technology perspective as each of these solution requires the smart card to be in close vicinity of the card reader such as at airport immigration. The reported work in this paper addresses a critical area of backend data processing for such large crowds.

III. Hajj Backend System Requirements

The pilgrims arrive to various airports terminals and shipping ports and they have to complete immigration process at each specific terminal. The pilgrims are expected to either pass through a scanning portal or are expected to wave their smart travelers' card in front of a contactless card reader. The card reader scans the relevant data for immigration processing, forms specific data packets, and provides the data to an application running on a host machine. The scanned traveler information from the smart card is then expected to be verified with the golden information stored in the immigration backend database. In order to process large number of pilgrims, the enquiry must happen within a predetermined time limit. The host application forms a DBMS query, sets timing constraint, and submits the query to the backend DBMS that contains the golden data for all pilgrims. If the pilgrim data is found, a beep is sounded at the terminal and a success notification is displayed on the host machine for the immigration officer. If the pilgrim data is not found, does not match, or the timing violation occurs, an exception handler is triggered. The exception handler will send a notification to the immigration officer to either have the pilgrim rescan the card or to perform the immigration process manually. Fig. 1 shows the Hajj system flow.

Some of the key requirements identified for the Hajj Traveler Information System are summarized as follows:

- Definition of specific data format after scanning pilgrim data from smart card and feeding it to the Traveler Information System application which then processes this data, forms a search query, and sends the query to the backend DBMS for verification of pilgrim data;
- Retrieval of pilgrim information from the DBMS for verification within a specified time limit (in milliseconds) under large data conditions which is expected to be anywhere from two to three millions;
- Definition and handling of all exceptional conditions:
  - Pilgrim is not found
  - Some or all pilgrim data does not match
  - Data is not retrieved in specified time limit
- Normal and exceptional conditions notification modes for immigration officials
  - Use some hardware mechanism such as different beeps for both passing and failing cases (will be simulated in the proposed project by printing messages on the screen).

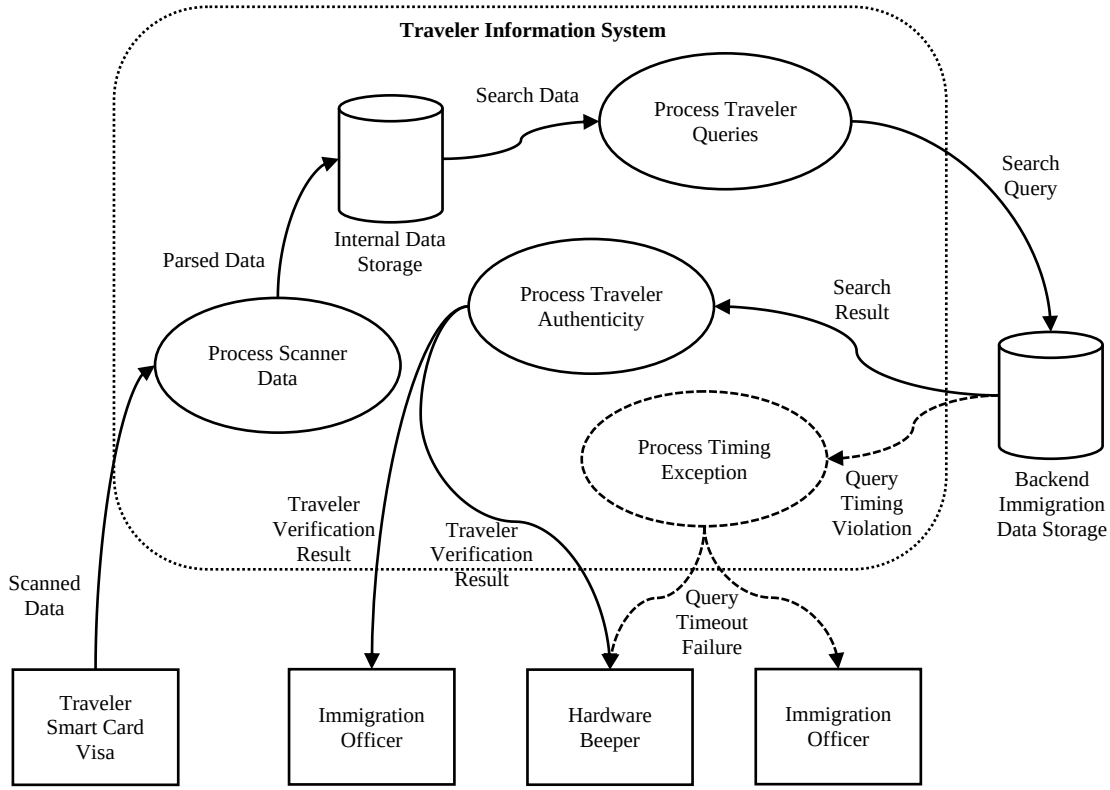

Fig. 1. Hajj Immigration Process Flow.

IV. DBMS Technologies For Hajj System

Multiple database technologies may be considered to process pilgrims' immigration information in a timely manner and to avoid any processing delays. Relational database may not be the ideal choice for such applications [10]. Relational DBMS are typically designed for storing transactional data and are optimized for analyzing complex relationships between data [10]. Relational databases can focus on these relationships because the data itself is updated relatively infrequently and does not require guarantying timeliness for each transaction.

Real-time Database Management System (RTDBMS) is a database system in which transactions have explicit timing constraints. RTDBMS does not depend only on logical result of computation but also the occurrence time of logical result. RTDBMS allows users to explicitly add desired timing constraints with various actions that may be met through system processing [11]. Some of the characteristics of RTDBMS are:

- Frequent changing of data stored in an RTDBMS within given timing constraint with correctness of such data;
- Handling of temporal data;
- Controlling concurrency access of data; and
- Defining and imposing timing constraints on data accesses.

Data management in real-time systems has specific timing requirements in the accessing and processing of data. Such time sensitive data management cannot be handled with traditional DBMS. Traditional databases often miss their transactions deadlines and suffer from overload management problem. Real-time databases have the ability to handle large number of queries with specific timing constraints. Another core feature of an RTDBMS is in-memory database. The traditional database is a disk-based database. Its processing time is nondeterministic because it involves the disk access i.e. data transfer between internal and external memory, buffer management, waiting list, and lock management. This property makes traditional database system unable to achieve the requirements of real-time transactions that must meet high efficiency and deterministic time [12].

A distributed database is a collection of multiple, logically interrelated databases distributed over a computer network [13]. Distributed databases could improve reliability and availability through distributed transactions and improve overall database performance. However, such database systems do not provide capability to define or to guarantee timing requirements as in the case of a real-time database system [13]. A real-time database system has the ability to handle large number of queries with specific timing constraints. On the other hand, distributed databases attempt to achieve integration without centralization, which is able to handle large amount of data but without timing constraints [13].

"Not Only SQL" or NoSQL databases have recently emerged with the increased number of internet applications and the development of cloud computing generating mass unstructured data that the traditional relational databases have started to struggle with [14]. Some of the advantages of NoSQL databases are that data can be quickly read and written, support of mass data, extensibility, and low cost [14]. A NoSQL database system offers an approach to data management and database design for large sets of distributed data and real-time web applications [14]. Studies have shown that NoSQL databases generally compare well in runtime performance with SQL based relational databases especially for inserts, updates, and simple queries but not when querying non-key attributes and aggregate queries [15]. Hence, NoSQL databases would be better for large data with constantly changing schema or with less complex queries. Such databases may also work comparably for certain operations on structured data as well. However, NoSQL DBMS may not work well on structured data with complex queries [15].

The main advantage of real-time database over distributed or NoSQL database is the ability of defining and enforcing timing constraints to each process that others do not support. Table 1 shows comparison of these database technologies for various DBMS features. Column two in Table 1 also highlights which of these features are considered a requirement for the Hajj pilgrim immigration processing application domain.

Since the smart card based automatic immigration processing of large number of pilgrims require response time from the backend DBMS in a very short time as well as mechanisms for handling exceptions in the case of timing violation, this project selected real-time DBMS technology to meet its primary requirement.

Table 1. Database Technology Comparison

Comparison	Feature Priority for the Proposed Project	Distributed Database	Real-time Database	NoSQL Database
Scheduling	Required	Yes	Yes	Yes
Timing Constraint	Required	No	Yes	No
Transactions Priority	Required	Yes	Yes	Yes
Dynamic Data	Not Required	No	Yes	Yes

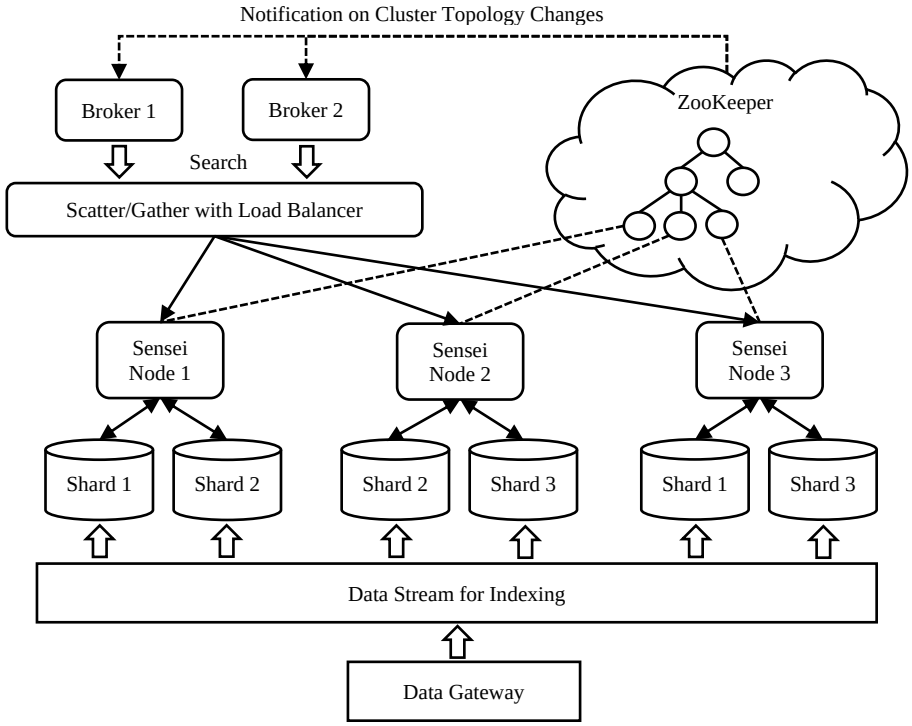

Fig. 2. SenseiDB Architecture [12].

Sensei Database: Sensei DBMS is a freeware distributed real-time database system that was built to support many product initiatives at LinkedIn.com and is the foundation to the LinkedIn's search and data infrastructure [16]. SensieDB was selected for the prototype implementation of the Hajj smart card based immigration processing project. SenseiDB has several characteristics: atomicity and isolation, durability, consistency, elasticity, and low query latency.

The relevance support functionality in SenseiDB is implemented to help users create and tune relevance models in an easy way. The basic pipeline is that users

send a JavaScript Object Notation (JSON)/Browsing Query Language (BQL) request containing a simple relevance model expression to the server. The server will compile the code and generate the corresponding relevance model. Relevance models may have a name so that next time the user only needs to refer to the model with its name. Also, models are cached inside each node and hence there is no extra performance cost [16]. SenseiDB also supports shards that help in organizing data into horizontal partitions. In designing large data systems by separating the data into smaller shards, query-time computation cost can be split into smaller shards and work can be done in parallel [16]. For example, the issue of querying data with 100 million documents can be reduced into ten parallel queries with ten shards, each holding 10 million documents with the final result is compiled by joining the individual results. Fig. 2 shows the architectural diagram of SenseiDB [16]. The Zookeeper component in Figure 2 manages Sensei's network topology or cluster.

V. Hajj Prototype System Implementation

In this project, multiple modules have been installed, setup, or implemented in order to get the real-time database system environment up and running. This section provides the details of the various modules and components necessary for the prototype Hajj backend system.

Sensei Schema: Sensei schema consists of one flat Extensible Markup Language (XML) table instead of separate tables for each entity to avoid using JOIN command that will cause unknown delays. In addition, there are no foreign keys in the sensei schema. Fig. 3 shows a sample partial schema for the Hajj pilgrim information.

```
-<table uid="id">
   <column name="TFName" type="string"/>
   <column name="TMName" type="string"/>
   <column name="TLName" type="string"/>
   <column name="TAge" type="int"/>
   <column name="TPassportNo" type="string"/>
   <column name="TContactNo" type="int"/>
   <column name="TCountry" type="string"/>
   <column name="TCity" type="string"/>
   <column name="TDistrict" type="string"/>
   <column name="TBuildingNo" type="string"/>
   <column name="ArrivalDate" type="date"/>
   <column name="AirportName" type="string"/>
   <column name="TVisaNo" type="string"/>
   <column name="TVisa_SDate" type="date"/>
   <column name="TVisa_EDate" type="date"/>
```

Fig. 3. Sample XML Schema.

Development Environment: SenseiDB is built using Java and Python languages. Hence, Java Development Kit (JDK) environment was used to develop the application software for this project. This application software is expected to receive and parse the scanned data from pilgrim smart cards and then compose and perform the SenseiDB queries. The JDK virtual machine allows the Sensei classes to understand and to perform several tasks related to the database and the BQL operations that are written using the Java programming language.

Python pyparser is a module that is an alternative approach to creating and executing python code. SenseiDB provides the client with the capability to develop applications using either Java or Python. Some modules of the Sensei database are built using the Python language.

Sensei Database Engine and Zookeeper Apache Server: SenseiDB consists of several files that comprise of the database engine and the Zookeeper Apache server. Zookeeper should be running before starting Sensei engine in order to get the database running. The reason behind using Zookeeper server is that it is capable to handle real-time aspects and has the ability to create several cluster nodes based on the data as each table represents one cluster.

Generation and Insertion of Hajj Data: Sensei database uses JSON objects as data format. In case of the Hajj prototype system, thousands of dummy records had to be produced in order to simulate the performance of the actual Hajj scenario. In order to generate dummy data, an online JSON website generator www.yandataellan.com [17] has been used as shown in Fig. 4.

After generating the JSON code as dummy data, it has been inserted into the Sensei JSON data file in order to allow the user to find searchable data. As mentioned above, this step is used to simulate the actual performance of Sensei database installed on server with millions of records during the Hajj season.

```
{"Hajj":[
    {"TravelerfirstName":"Abdulaziz",
     "TravelerlastName":"AlQasem"},
]}
```

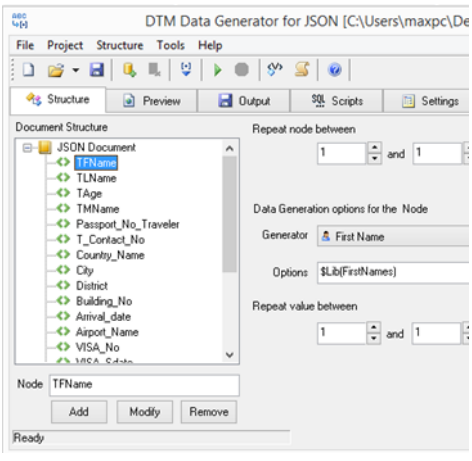

Fig.4. JSON Data Generator.

SenseiDB uses XML format to identify tables and facets that will use JSON file as source of data as shown above.

Identification of Sensei Database Configuration: SenseiDB comes with multiple configurations that need to be setup as Sensei properties before the database could be used. These properties help define some key

parameters for the database such as timing constraints for the queries, schema, and data location. Fig. 5 shows a sample of Sensei properties.

```
# sensei node parameters
sensei.node.id=1
sensei.node.partitions=0,1

# sensei network server parameters
sensei.server.port=1234
sensei.server.requestThreadCorePoolSize=20
sensei.server.requestThreadMaxPoolSize=70
sensei.server.requestThreadKeepAliveTimeSecs=
300

# sensei cluster parameters
sensei.cluster.name=sensei
sensei.cluster.url=localhost:2181
sensei.cluster.timeout=30000

# sensei indexing parameters
Sensei.index.director=index/hajjdata
```

Fig.5. Sensei Configuration.

Installation and Configuration of Zookeeper Server: Zookeeper is a centralized Apache server that provides services for maintaining configuration information, naming, providing distributed synchronization, and providing group services. Zookeeper Apache server comes with configurations such as setting port, cluster, and localhost, that are needed to run server clusters. These configurations are related with the port number and time needed for ticking "ping". Fig. 6 shows setting of Zookeeper configuration for this project.

```
tickTime=2000
dataDir=/var/lib/zookeeper
clientPort=2181
```

Fig. 6. Zookeeper Configuration.

Execution of BQL Queries: Browsing Query Language (BQL) is an SQL-like language that provides Sensei users easy access to the database if they have experience using traditional relational databases. A BQL example query is as follows:

```
SELECT tags, price from cars
WHERE tags CONTAINS ALL ("cool",”hybrid")
EXCEPT("favorite") LIMIT 5;
```

These BQL queries may directly be run in the Sensei shell or can be composed and submitted using the Java application connected with the Sensei DBMS.

Java Application: A Java desktop application has been developed to read and parse pilgrim data from a smart card, to form and to execute a Sensei query, to handle exceptions, and to define additional finer granularity timing constraints on top of the existing timing constraints provided by Sensei. Sensei provides timing constraints at the "minutes" granularity level. In order to meet the project requirements, finer timing constraints at the milliseconds level were needed which were defined and enforced at the Java application level.

The Java desktop application consists of several classes. Table 2 lists the Hajj Java application classes and their purpose. Some of these classes are used to store specific pilgrim information such as personal information, country information, passport and visa information, flight information, and Mualeem information. These classes provide the user the ability to manipulate data separately and retrieve them again, e.g. specifying age range to be retrieved from traveler class to retrieve just the accepted range of age. Mualeem is a local Hajj group responsible for logistics of the pilgrims such as accommodation, local transport, food, etc. Each pilgrim coming for Hajj has to register with some local Mualeem group as part of the Hajj application process.

Voice Alert class is to provide beep from reading .wav file containing several messages. These messages are for various notifications from the backend DBMS such as pilgrim information positive verification, pilgrim information missing or mismatch notification, and timing constraint violations while accessing the backend DBMS.

In the timing module, synchronized method has been used to delegate the Web Service invocation to another thread, and joining on it using a specified timeout. The timing module creates tunnel that carries out request with timing constraint. Sensei query is loaded within this request using synchronized method to be performed within the specified timing constraints. The application module allows the ability to enhance and to increase the timing constraints granularity from minutes as supported by Sensei to milliseconds as required by the project.

Exception Handling: In the Hajj Java application, the main class contains Sensei query class that has the responsibility to query the database based on data gathered from JSON file. There are following possibilities of running a query:

- If the query finds and retrieves the needed data within specified timing constraints, the application will display message indicating that the pilgrim information exists and it will result in a specific beep at the terminal.
- If the Sensei query could not find matched information or if there are mismatches in some fields within specified timing constraints, it will display message indicating that there is no record for this traveler with a specific beep at the terminal. In this case, the system will prompt the immigration officer for a manual verification.
- If the Sensei query exceeded the query search timing constraints, the application will make two more attempts. The application will then send a request to Sensei and Zookeeper server to kill the query after the third attempt also fails. Message will be displayed indicating that the timing constraints exceeded and a unique beep will be produced at the terminal. The application will then prompt the immigration officer for manual verification of the pilgrim.

Table 2. Hajj Java Application Classes

Class	Description
Traveler info class	Contains traveler information and related methods such as: first name, middle name, last name, contact number and age.
Country info class	Contains country information and related methods such as: country name, city, district, and building number.
Passport info class	Contain passport information and related methods such as: passport number, passport issue date, and passport expire date.
Visa info class	Contain visa information and related methods such as: visa number, visa issue date, and visa expire date.
Mualeem info class	Contains Mualeem information and related methods such as: Mualeem id, first name, last name, contact number, email, and office location.
Arrival info class	Contains information about arrival information and related methods such as: flight information, airport name, arrival date and time, leaving date and time.
Main class	Contains four sub-classes: timing class, query class, JSON parser class, and voice alert class. In addition, it serves as the glue class for the various information classes listed above.

Query Performance and Optimization: SenseiDB improves query search performance by automatically indexing data loaded in the random access memory (RAM) for the first run to enhance the performance of query. It also offers manual indexing for machines with not enough RAM to allow Sensei to load manually indexed data as partial dataset into RAM. Both indexing helps Sensei to easily find the needed data previously queried. In this project, the prototype is developed on a client machine with limited RAM. Hence, manual indexing has been used for a small subset of data as test.

Sample Runs Using a Large Dataset: Dataset has been populated for this prototype implementation with around million records to simulate the performance of real case of Hajj. As Sensei database was designed to run in a server environment, 3.8 GB of RAM was not enough to load the entire dataset on the client machine, so this prototype loaded partial datasets to get the sample runs. In all of the selected datasets, the query was performed within the specified timing constraints. In order to ensure the two exceptional cases can be correctly handled by Sensei and the Java application, forced error conditions were correctly simulated by the prototype system. The three test scenarios were as follows:

- The first case contains correct data stored in JSON file and runs within given timing constraint.
- Second case contains incorrect data stored in JSON file and runs within given timing constraint.
- Third case contains correct data performed with smaller duration of timing constraint to demonstrate timing violation.

VI. Implementation Issues

There were some challenges encountered during the prototype Hajj system implementation and there were some key learnings as well.

1. Issue: Linux operating system is a highly customized, open source operating system and it requires high-end machine to perform well. It was challenging to develop real-time project in a Linux environment.
   Learning Outcome: Fedora version is the most suitable version for a client personal computer. On the other hand, Red Hat is the best-customized version for the server environment.
2. Issue: Zookeeper Hadoop Apache server is appropriate to deal with real-time databases because it is a high-performance coordination service for distributed applications with different clusters. Configurations of Zookeeper server were challenging due to rarity of resources available on the Web.
   Learning Outcome: Using general configuration is enough; Sensei database comes with its configuration that is needed to manipulate real time aspects.
3. Issue: BQL language is the query language for Sensei with unique syntax. This language is like SQL but the difference is that BQL has the ability to deal with facets that allow users to configure the way they want to retrieve the data.
   Learning Outcome: BQL has two syntaxes. The first syntax works with Linux shell as an SQL like query language. The second has a special syntax used by Java application based on Sensei library.
4. Issue: Sensei database uses Zookeeper instance as the default setting. It was a difficult issue to deal with the included version instead of using a new separate version available on the Web.
   Learning Outcome: Included version is fully compatible with the Sensei database but had to be properly configured. After some debug, it was determined that the port number and tick time had to be properly specified as they vary from system to system.
5. Issue: Zookeeper server allows the user to define up to eight clusters, which will increase performance by replicating the data. Due to lack of any documentation, it was quite challenging to enable and implement multiple clusters.
   Learning Outcome: Zookeeper Apache server is designed to represent one cluster for each table of the database. As this project deals with only one entity “traveler”, one table was enough to represent the needed information that was held in one cluster for this prototype implementation. An extension of the project is going on currently enhancing the performance by enabling multiple clusters.

6. Issue: Database model used in the SenseiDB is a flat XML file combined with JSON files. This fact had to be discovered during the preliminary setting and implementation because Sensei Web site does not provide details about the database model.
   Learning Outcome: XML file used along with JSON file to represent the facet which is being used to indicate how Sensei will query the data. Facets can be specified as a range to inform the query that there are data only between the specified range. Facets is one factor that helps Sensei to speed up query performance.

VII. Conclusion

A prototype system for improving immigration processing efficiency of Islamic Hajj event large crowd management was developed using Sensei real-time DBMS. The prototype system was developed as a feasibility study for the concept and was successfully used to demonstrate that Hajj large crowd management can be handled with the proposed approach. At present, several efforts are currently going on to extend this work at the College of Computer Sciences and Information Technology, King Faisal University, Kingdom of Saudi Arabia. An extension of the prototype backend DBMS system is under development to move the system from client environment to a server environment that will allow running on datasets of from three to five millions to simulate actual Hajj load and overload. In addition, sharding or horizontal partitioning is being enabled to evaluate and to further improve performance of pilgrim information verification. Finally, performance analysis of Sensei versus Mongo NoSQL database is also being worked on to understand if there are any performance advantages of one versus the other and to accurately capture and define timing constraints under various application scenarios.

References

[1] Hajj. https://en.wikipedia.org/wiki/Hajj [20 Mar 2017].
[2] Vision 2030. http://vision2030.gov.sa/en [20 Mar 2017].
[3] M. Yamin, "A Framework for Improved Hajj Management and Research," *ENTIC Bulletin*, 2008.
[4] Ministry of Hajj. http://haj.gov.sa/english/pages/default.aspx [20 Mar 2017].
[5] E. Riyad, "Towards a High-Quality Religious Tourism Marketing: The Case of Hajj Service in Saudi Arabia," *Tourism Analysis*, Vol. 17, No. 4, pp. 509-522.
[6] R. O. Mitchell, H. Rashid, F. Dawood, and A. AlKhalidi, "Hajj Crowd Management and Navigation System : People Tracking and Location Based Services via Integrated Mobile and RFID Systems," *Proceedings of the 2013 International Conference on Computer Applications Technology (ICCAT)*, Jan 20-22, 2013, Sousse, Tunisia, pp. 1-7.
[7] M. Mohandes, "An RFID-Based Pilgrim Identification System (A Pilot Study*)," Proceedings of the 11th International Conference on Optimization of Electrical and Electronic Equipment (OPTIM 2008)*, May 22-24, 2008, Brasov, Romania, pp. 107-112.
[8] E. Alsaggaf, O. Batarfi, N. Aljojo, and C. Adams, "Secure Hajj Permission Based on Identifiable Pilgrim's Information," *International Journal of Information Technology and Computer Science (IJITCS)*, Vol. 7, No. 5, April 2015, pp. 67-76.
[9] A. Geabel, K. Jastaniah, R. Abu Hassan, R. Aljehani, M. Babadr, and M. Abulkhair, "Pilgrim Smart Identification Using RFID Technology (PSI)," in *Design, User Experience, and Usability: User Experience Design for Everyday Life Applications and Services*, Lecture Notes in Computer Science, Springer, 2014, pp. 273-280.
[10] B. Kao and H. Garcia-Molina, "An Overview of Real-Time Database Systems," in *Real Time Computing*, W. A. Halang and A. D. Stoyenko (editors), NATO ASI Series F: Computer and Systems Sciences, Vol. 127, Springer: Berlin, Heidelberg, 1994, pp. 261-282.
[11] Y. Lu, Y. Liu, Q. Han, G. Hu, and Z. Li, "The Architecture and Execution Model of an Active Real-time Database Management System," *Proceedings of the 1997 International Conference on Information, Communications and Signal Processing (ICICS)*, Sep. 9-12, 1997, Singapore, pp. 815-819.
[12] J. Wu, Y. Cheng, and N. N. Schulz, "Overview of Real-Time Database Management System Design for Power System SCADA System," *Proceedings of the IEEE SoutheastCon 2006*, Mar. 31 – Apr. 2, 2006, Memphis, Tennessee, USA, pp. 62-66.
[13] M. T. Ozsu and P. Valduriez, *Principles of Distributed Database Systems*, 3rd ed., Springer, 2011.
[14] J. Han, E. Haihong, G. Le, and J. Du, "Survey on NoSQL database," *Proceedings of the 6th International Conference on Pervasive Computing and Applications (ICPCA)*, Oct. 26-28, 2011, Port Elizabeth, South Africa, pp. 363–366.
[15] Z. Parker, S. Poe, and S. V. Vrbsky, "Comparing NoSQL MongoDB to an SQL DB," *Proceedings on the 51st ACM Southeast Conference (ACMSE '13)*, Apr. 4-6, 2013, Savannah, Georgia, USA, pp. 5:1 – 5:6.
[16] SenseiDB. http://senseidb.github.io/sensei/overview.html [30 Jan 2017].
[17] Data Generator. www.yandataellan.com [1 Mar 2017]

Authors' Profiles

Amir A. Khwaja received his MS and PhD degrees in Computer Science from Arizona State University, Arizona, USA. He received his Bachelor's degree in Computer Engineering from N.E.D. University of Engineering and Technology, Karachi, Pakistan.

He is an assistant professor at the College of Computer Sciences and Information Technology, King Faisal University, Al-Ahsa, Kingdom of Saudi Arabia since 2014. Prior to that he had 21 years of semiconductor industry experience. He worked for 20 years in Intel Corporation, USA, in various capacities: CAD software developer, XScale and Atom mobile system-on-a-chip (SoC) validation architect, Validation Program Manager, and Sr. Engineering Manager. He worked for one year in Qualcomm, San Diego, California, USA, as a Principal Engineer and Sr. Manager, leading the successful completion of the validation of Qualcomm's

Femtocell SoC product. He has also worked as an adjunct faculty for Arizona State University and University of Phoenix.

How to cite this paper: Amir A. Khwaja," A Real-time DBMS System for the Immigration Processing of Large Hajj Crowd", International Journal of Modern Education and Computer Science(IJMECS), Vol.9, No.9, pp. 32-41, 2017.DOI: 10.5815/ijmecs.2017.09.04

