A Real-Time Algorithm for Tracking Astray Pilgrim based on in-Memory Data Structures

Mohammad A.R. Abdeen[1], Ahmad Taleb[2]
Department of Computer Science
Faculty of Computer and Information Systems
Islamic University of Madinah
Madinah, Saudi Arabia

***Abstract*—Large crowd management presents a significant challenge to organizers and for the success of the event and to achieve the set objectives. One of the biggest events and with largest crowd in the world is the Muslim pilgrimage to Mecca that happens every year and lasts for five years. The event hosts over two million people from over 80 countries across the world with men, women, and children of various age groups and many languages. One of the challenges that faces the authorities in Saudi Arabia is that many of the pilgrims become astray during the event due to the relative complexity of the rituals mainly mountainous landscape and the language barrier. This result in them being unable to perform the required rituals on the prescribed time(s) with the possibility to invalidate the whole pilgrimage and jeopardize their once-in-a-life journey. Last year over 20,000 pilgrims went astray during the pilgrimage season. In this paper we present a tracking algorithm to help track, alarm, and report astray pilgrims. The algorithm is implemented on a server that contains pilgrims' data such as geolocations, time stamp and personal information such as name, age, gender, and nationality. Each pilgrim is equipped with a wearable device to report the geolocations and the timestamp to the centralized server. Pilgrims are organized in groups of 20 persons at maximum. By identifying the distance of the pilgrim to its group's centroid and whether or not the pilgrim's geolocation is where it is supposed to be according to the pilgrimage schedule, the algorithm determines if the pilgrim is astray or on a verge of becoming astray. Algorithm complexity analysis is performed. For better performance and shorter real-time time to determine the pilgrim's status, the algorithm employs an in-memory data structure. The analysis showed that the time complexity is O(n). The algorithm has also been tested using simulation runs based on synthesized data that is randomly generated within a specified geographical zone and according to the pilgrimage plan. The simulation results showed good agreement with the analytical performance analysis.**

Keywords*—In-Memory structure; real-time; tracking algorithm for astray pilgrim; large crowd management

I. Introduction

Managing of large crowds represent significant challenges to in numerous large-crowd events across the globe. One of the largest, most noticeable and frequent of these events is the Muslim pilgrimage to Mecca (Hajj). Every year around three million pilgrim travel to perform their life duty of pilgrimage. Several challenges face those millions of pilgrims including the fact that they probably have never been to those places before in addition to the lack of knowledge of the Arabic language, the language of the land. Pilgrims also need to visit numerous places as part of their Hajj rituals. The landscape of the majority of those places are mountainous (Such as Mount Arafat and Muzdalifah) or do not have street names but rather a tremendous camp site of tens of thousands of tents (the region on Mena). Those destinations have a relatively small area of a few square kilometre which constitutes very high-density population with many of elderly men, women and children.

A significant challenge that faces the authority is Saudi Arabia is that many of these pilgrims go astray during their once-in-a-lifetime journey which could mean they miss the time window of the rituals thereby invalidating the whole Pilgrimage and waste their lifetime saving. In fact, a published study showed that around 70% of the male pilgrims are over 60 years old and 30% of them are illiterate [1]. In a previous year, around 20,000 pilgrims were went astray during the trip [2].

Existing systems that help guide the lost pilgrims rely on the pull model. In other words, if the authority locates an astray person then they read his/her information that is barcoded on wearable device, such as a bracelet. An authority personal then guides that pilgrim to their destination/group/tent. There could be much time wasted prior to locating an astray pilgrim which increases the possibility that pilgrimage is invalid. In many cases pilgrims are unable to use mobile phones due to poor coverage given the large number of subscribers or due to an empty battery and the unavailability of nearby charging facilities.

A. Previous Work

Previous systems used in the kingdom of Saudi Arabia to track and guide the astray pilgrims are mainly human based. The kingdom has hired thousands of boy scouts during Haj season for the purpose of guiding the lost Haj. But these systems in place are poll-type systems, i.e. the pilgrim has to actually reach to the center of lost Haji's or they have to come across one of the guides. It might be such time before the pilgrim is guided. The guide uses the name tag – that includes very limited information – to identify the pilgrim and their group and/or tent location in Mena [3]. Last year, the Saudi announce the launching of the "Electronic bracelet" project [4]. It includes information about the pilgrim but it is not equipped with a facility to report that the pilgrim is lost neither with the ability to send their location and information over a wireless network. In another work by Mohandes [5] RFID tags were used to store pilgrim information such as name, passport info, country. It was also suggested that the tags can be used to track the pilgrims by placing an RFID reader in the vicinity. Tracking of hundreds of thousands requires thousands of RFID readers while the lost ones are in thousands only. The system does not give the ability to the pilgrim to identify themselves as lost but rather it attempts to detect a lost pilgrim by tracking all of the pilgrim. This solution is an overkill and it also produces many false alarms. In [6], the authors spoke about a smart RFID system but the objectives of such a system was to store the personal information about the pilgrim as well as their medical information that will be of help in emergency situations. The system does not use a GPS module to locate the pilgrim and can only be used for location purposes in shorter ranges (100 meters of so). This is not suitable for the whole pilgrimage area which extends for several kilometers (10 km). In [7], however, the authors introduced a GPS based tracking system to locate each and every Haj and store their medical information. This system requires powerful servers and are implemented in discreet components. The proposed system does not consider the size, cost, and the power consumption. There are commercial products in the market that are designed for children tracking [8]. Examples of those products are AngelSense, hereO GPS, and AmbyGear. Those products however are pricy (from $120 - $170 USD) and the battery life is in around 40 hours. These devices are available as independent devices with no existing intercorrelation with no crowd management capabilities.

Traditional methods of creating databases in-disk suffer from long delays and does not satisfy the real-time performance requirements for many of today’s applications. As an example, trading companies need to detect sudden changes in trading processes and act upon this change “instantaneously” (within few milliseconds). Such a targeted response time is impossible to achieve using traditional disk-based storage/processing systems. The solution is to keep the data in the random-access memory (RAM) all the time.

In-memory database systems have been used in the past [10, 11] but those techniques have been challenged by the recent evolution in hardware [12]. Previous work on in-memory data management and processing have focused on several aspects such as indexing [13, 14], data layouts [15], parallelism [16, 17], concurrency control and transaction management [18, 19], query processing [20, 21, 22] and fault tolerance [23, 24]. In this work we present an in-memory database solution for the purpose of large-crowd tracking at real-time.

II. The System Architecture

In a previous work, we presented an overall architecture of an astray pilgrim tracking system [9]. The architecture is shown in Fig. 1. It consists of a client side and a server side. The client is in the form of wearable devices that have a GPS module to transmit the current geo-locations frequently (every hour in normal situations). The devices are also connected to the mobile network via a SIM card from sending the geo-coordinated and receiving commands from the central station. The server side consists of a database storing the pilgrim’s information including their ID and their group ID, current (and previous) GPS locations with the corresponding time stamps. The database design with the tables is discussed in the following section.

III. The Transactional Database Design

At the operational or transactional level, the main target of this work is to guide the astray pilgrim by sending them alert messages as well as to their group leader and the authorities. To achieve this objective the system design includes a normalized relational database that supports an improved real time operation. The main function of the central database is to record the pilgrims’ geolocations and other necessary information for possible future data analytics. Fig. 2 shows the relational data that supports the storage of pilgrims’ information as well as their geolocations and status during the Hajj period. The figure shows four tables; the Pilgrim, the Group, the PilgrimTracking, and the ResponsibleAuthority tables. The Pilgrim table, contains details about each pilgrim while the PilgrimTracking table contains the PilgrimID (GroupID and ID of pilgrim in the group), timestamp, GPSLocation, status of pilgrim and distance between the pilgrim and the centroid of the pilgrim's group. In addition, the information about the groups such as their IDs, leaders, phone numbers are stored in table Groups. Moreover, the table ResponsibleAuthority contains information about the authorities (managers, phone number and office location). Each table in the relational database has a primary key (single or compound). For instance, the primary key of table PilgrimTracking is compound of three attributes. The relationship between the tables are represented by using the concept of foreign keys. For instance, the attributes AuthorityID, PilgrimID, GroupID are foreign keys (FK) that are used to connect the tables of the database together.

IV. A server-side Real-Time Tracking Algorithm

As per the ministry of Haj in Saudi Arabia, a group leader is assigned for every 20 pilgrims. Therefore, we assume that the maximum number *m* of pilgrims in a group is 20. Each tracking device stores the personal information of the pilgrim. This includes the pilgrim ID which is composed of the original group ID and the number of the pilgrim within the group (maximum 20 pilgrims per group), Name, Date of Birth, phone, spoken language, nationality and gender. The original number of groups is calculated as follows:

Number of Original Groups = Total Number of Pilgrims / Maximum Number of Pilgrims in a group

As per the official document of the Ministry of Haj in Saudi Arabia, the total number of domestic and foreign pilgrims in 2017 was 2.4 Million. Therefore, the estimated number of groups is 2.4 M /20 = 120,000 groups.

A. *Tacking All Pilgrims using Two Dimensional Array as an in-Memory Structre*

Fig. 3 below depicts the processing steps for the server with the numbered circles showing the flow. It is at this stage that the in-memory structures are created and manipulated. Upon system start, each tracking device sends the geolocations and the timestamp to the centralized server via the RF interface. At the receiving end of the server is a dispatcher process. The dispatcher process receives the geolocation and the timestamp information and then stores this information into one of the two data structures, the PilgrimLocationRecord or the LostPilgrimRecord. The PilgrimLocationRecord is a two-dimensional array PilgrimLocationRecord[n, m] (where n is the number of groups (equation 1) and m is the maximum number of pilgrims in a group). Each element of this array is a structure that contains the current geolocation, the current timestamp, the status of the pilgrim (whether astray or not astray) and the pilgrim's distance from their group centroid. A group centroid is calculated as in Equation 1. The PilgrimLocationRecord array is created/updated every hour. The geolocations and timestamps are sent to the centralized server, however, the values of the status and distance from centroid are calculated and updated during processing of the received data. Algorithm 1 provides a concise overview of tracking all pilgrims every hour.

The process of determining lost pilgrims every hour is described as follows (Algorithm 1): Pilgrims move in groups. Each group consists of a maximum of 20 persons. For each group the group centroid is calculated using the GPS coordinates of each member of the group. A group centroid at any time is defined as the point in space where the sum of distances of this point to all group members is minimum. The centroid is given by Equation (1) below.

$$Cent = \min \sum_{ID=0}^{ID=m} LOC_{ID} - LOC_{cent} \quad (1)$$

The average distance Dav of each group is also calculated based on Equation (2) below

$$D_{av} = \frac{\sum_{ID=0}^{ID=m} D_{ID}}{m} \quad (2)$$

To determine if a pilgrim is astray it is hypothesized that if its distance to the centroid is three or more times the average distance Dav to his/her group, the an astray pilgrim status is declared. The status value of each pilgrim is either normal (N) if the pilgrim's distance to the centroid is less than three times Dav, or astray (L) if the distance is at least three times the Dav.

If a pilgrim's status is N, then an alert signal is sent to the wearable device to show a green color. If the status is L, however, a continuous red light is shown on the wearable device if L was persistent for three consecutive reads. Otherwise, a flashing red light is shown on the wearable device.

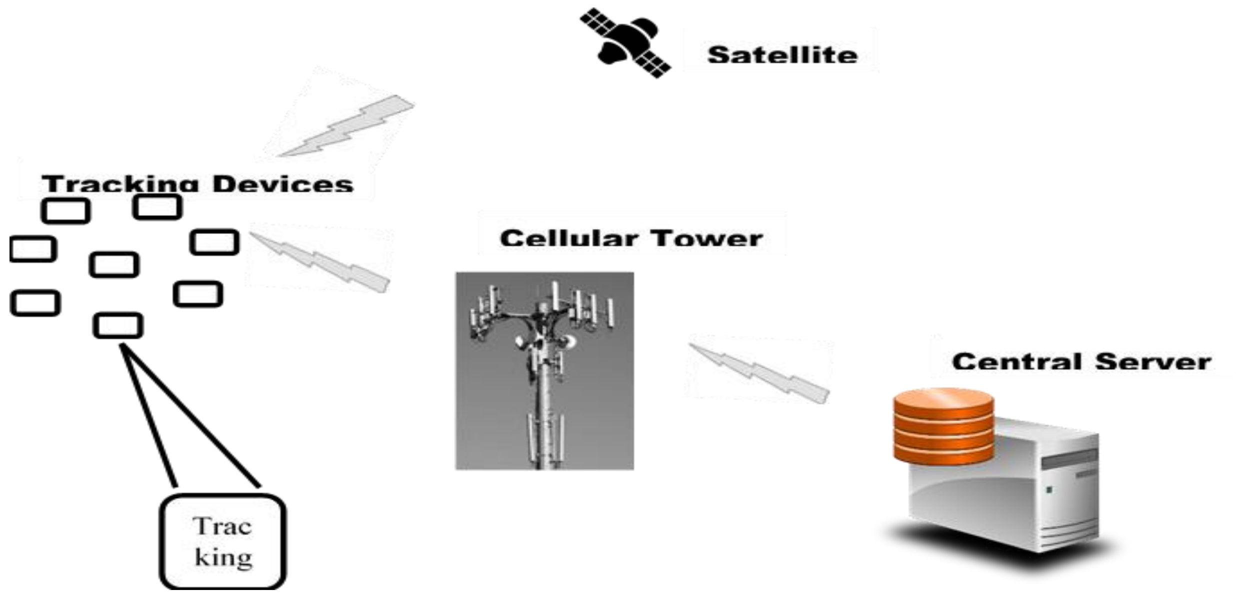

Fig. 1. The Astray Pilgrim Tracking System Architecture.

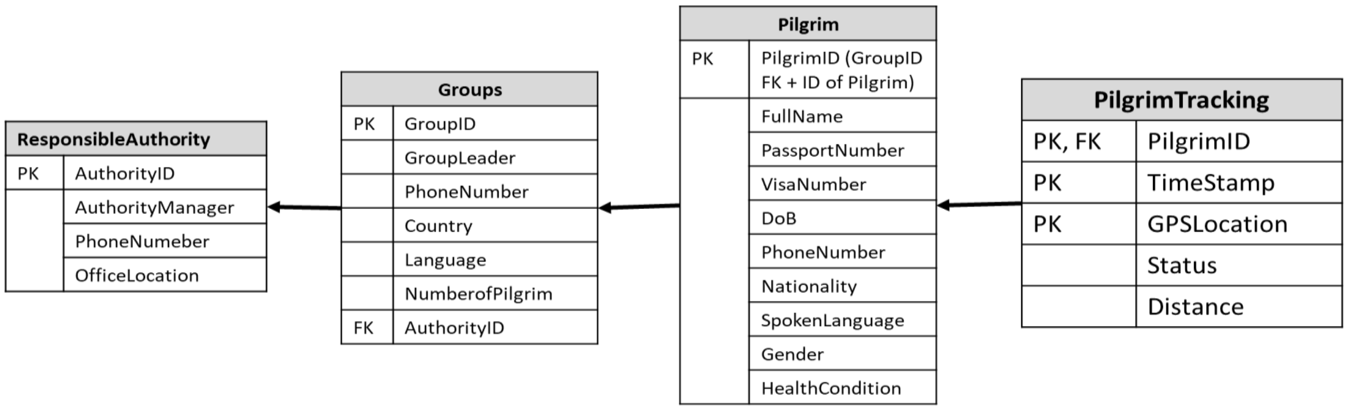

Fig. 2. The Centralized Database design.

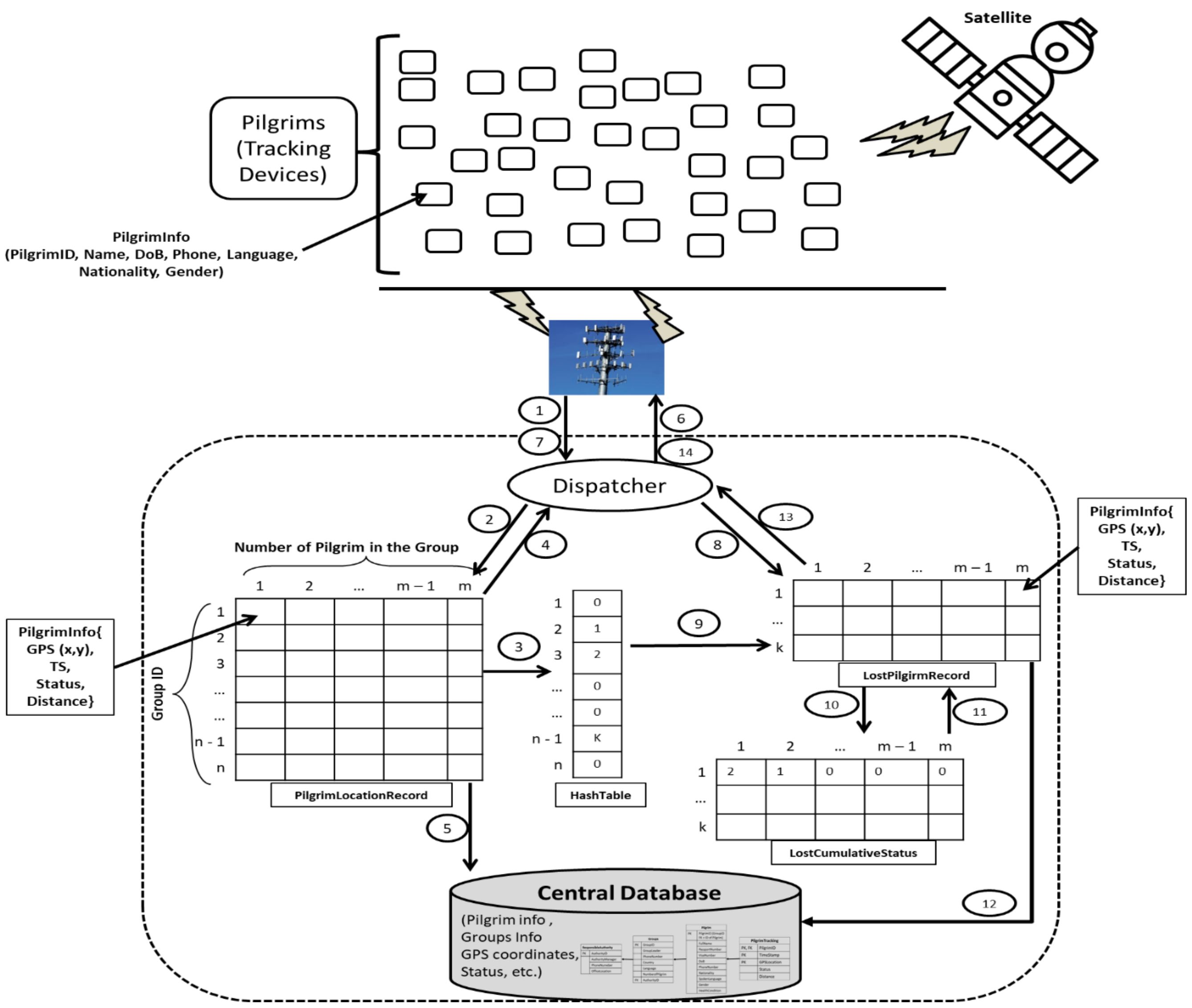
Satellite
Pilgrims (Tracking Devices)
PilgrimInfo (PilgrimID, Name, DoB, Phone, Language, Nationality, Gender)
Dispatcher
Number of Pilgrim in the Group
Group ID
PilgrimInfo{ GPS (x,y), TS, Status, Distance}
PilgrimLocationRecord
HashTable
LostPilgirmRecord
LostCumulativeStatus
Central Database
(Pilgrim info , Groups Info GPS coordinates, Status, etc.)

Fig. 3. The in-Memory Data Structures of the Server-Side Tracking Architecture.

The values of the status and the distance between the centroid and each pilgrim's GPS coordinate are updated in the PilgrimLocationRecord structure. If a pilgrim’s status indicated a lost or is about to be lost (if the status is L for less than three consecutive times) the values of the PilgrimLocationRecord structure are flushed into the PilgrimTracking table in the central database (Step 5 in Fig. 3). Tracking information for all pilgrim is updated every hour. If a group has one or more lost member, all pilgrims in that group are tracked more frequently (every 10 minutes) to be able to determine the actual status of the pilgrim within the hour. Algorithm 2 shows the key steps of tracking lost pilgrims every ten minutes.

B. Hashing Structure

A hashing mechanism is used to map only the groups with lost pilgrims using their original ID while utilizing sequential places in memory for direct access and minimal memory size. The hashing mechanism is implemented as a one-dimensional array of n integers and is called HashTable (Fig. 3). This array is used to store indexes of groups who have lost pilgrims. Fig. 3 shows an example of the HashTable structure with n integers (where n is the number of original groups). A value of “0” in a given hash location means that this specific group has no lost members. A value other than “0” means there are nonzero lost members and the specific value gives the cumulative sequence number of this group in the lost pilgrims’ groups. As an example, the HashTable[1] contains a value of zero and therefore group “1” has no lost members. On the other hand, the values of HashTable[2] and HashTable[3] mean that group 2 and 3 have lost pilgrims. The contents of the HashTable structure in Fig. 3 illustrates that there are k groups (group 2, 3, and n-1) with lost pilgrims. The HashTable structure ensures O(1) conversion of the original group ID to the appropriate index in the LostPilgrimRecord structure (A two dimensional array to track lost pilgrims more frequently (e.g. every 10 minutes)). For instance, the original group “3” is stored in the second row of the LostPLigrimRecord structure because HashTable[3] = 2 (meaning that the third group has lost pilgrim and will be stored in LostPilgrimRecord [2]).

Algorithm 1 Tracking All Pilgrims (every hour)

```
1. Create a two-dimensional array PilgrimLocationRecord[n, m]
2. Receive GPS coordinates and timestamps of all pilgrims' IDs
3. for each pilgrimID (ID) with GPS coordinate G and timestamp TS do
       3.1 Split the PilgrimID (ID) to get the group ID (x) and the ID (y) of the pilgrim within Group x
       3.2 Update PilgrimLocationRecord [x, y].GPS = G
       3.3 Update PilgrimLocationRecord [x, y].timestamp = TS
4. Create an array of integers HashTable [n], default value is 0
5. Create an integer variable Lost
6. Create a variable nbGroupLost and set it to 0    //number of groups with lost pilgrims
7. for each group $g_i$ , where i =1,2, …, n do
       7.1 calculate the centroid $c_i$ of $g_i$
       7.2 Lost = 0
       7.3 for each pilgrim j in group $g_i$ , where j =1,2, …, m do
               7.3.1 calculate the distance d between PilgrimLocationRecord[i, j].GPS and $c_i$
               7.3.2 PilgrimLocationRecord[i, j].distance = d
               7.3.3 if d > LostDistance then      // LostDistance = 1000 Meters
                       7.3.3.1 send flashing signal to Pilgrim (i,j)
                       7.3.3.2 PilgrimLocationRecord[i, j].status = 1 // 1 means lost, 0 means Not lost
                       7.3.3.3 Lost = 1
       7.4 If Lost = 1 then
               7.4.1 Send request to the RFID tower to track the pilgrims in $g_i$ every 10 minutes
               7.4.2 nbGroupLost = nbGroupLost + 1
               7.4.3 HashTable[i] = nbGroupLost
8. Insert all values of PilgrimLocationRecord structure into table PilgrimTracking (PilgrimID, TimeStamp,
   GPSLocation, Status, Distance)
9. Delete PilgrimLocationRecord structure from the main memory
```

C. *Tracking Lost Pilgrims using Two-Dimensional Array as an in-Memory Strcuture*

Once the groups with astray members are identified and the HashTable is constructed, an in-memory structure called LostPilgrimRecord is created (Fig. 3). This structure is a two-dimensional array with k rows and m columns (k represents the number of groups with astray pilgrims and m is the maximum number of pilgrims in a group). The LostPilgrimRecord array is used to store information about lost pilgrims every 10 minutes. The HashTable is used to get the appropriate index of each original group ID. For example, the information of pilgrim ID 21 (where original group ID is 2 and the ID of the ID of the pilgrim within group 2 is 1) is stored in LostPilgrimRecord [1, 1]. The centroid of each group is calculated and used to update the status and the distance between a pilgrim and his/her group centroid. A two-dimensional array LostCumulativeStatus[k, m] of integers is created to track the consecutive lost status of each pilgrim. Before sending any alert signal to the lost pilgrim, the LostCumulativeStatus (a structure to track the consecutive lost status of pilgrims) is checked to know how many consecutive times a pilgrim is reported lost. The process is as follows (1) If the status of pilgrim LostPilgrimRecord [i, j] is L (Lost), then the value LostCumulativeStatus[i, j] is incremented (2) else if the status of pilgrim LostPilgrimRecord [i, j] is N (Normal or Not Lost), then the value of LostCumulativeStatus [i, j] is reset to 0 and a green light is shown on the wearable device. There are three types of light indicators shown by the wearable device:

- A green light, which means everything is fine and the status is N.
- A flashing red light, which means that the pilgrim has been detected lost by the system for a number of consecutive times less than three.
- A continuous red light, which means that the pilgrim has been detected and confirmed lost.

For example, Fig. 3 shows that Pilgrim (21) has been detected to have been astray for two consecutive sampling times, whereas, Pilgrim (22) was detected lost for only one time. Finally, the values of the LostPilgrimRecord are flushed into PilgrimTracking table in the central database. Algorithm 2 shows the processing steps to track lost pilgrims every 10 minutes.

V. Theoretical Analysis (Size and Performance)

In this section, we discuss the main memory storage requirements and performance for the proposed in-memory structures and algorithms to track astray pilgrims during the haj period.

A. *Size Complexity*

The main memory storage requirements for the **PilgrimLocationRecord**, **HashTable, LostPilgirmRecord, LostCumulativeStatus** structure are quite impressive.

Algorithm 2 Tracking lost Pilgrims (every 10 minutes)

```
1. Create a two-dimensional array LostPilgirmRecord [nbGroupLost, m]
2. Create a two dimensional array of integers LostCumulativeStatus [nbGroupLost, m]
3. Receive GPS coordinates and timestamps of the pilgrims who belong to the groups with lost pilgrim(s)
4. for each pilgrimID (ID) with GPS coordinate G and timestamp TS do
       Split the PilgrimID (ID) to get the group ID (x) and the ID (y) of the pilgrim within Group x
       z = HashTable[x]  //convert the original group ID into the corresponding index
       Update LostPilgirmRecord [z, y].GPS = G
       Update PilgrimLocationRecord [z, y].timestamp = TS
5. Create an integer variable Lost
6. for each group $g_i$ , where i =1, 2, …, nbGroupLost do
       calculate the centroid $c_i$ of $g_i$
       Lost = 0
       for each pilgrim j in group $g_i$ , where j =1,2, …, m do
           calculate the distance d between LostPilgirmRecord [i, j].GPS and $c_i$
           LostPilgirmRecord [i, j].distance = d
           if d > LostDistance then                    // LostDistance = 1000 Meters
               LostPilgirmRecord [i, j].status = 1
               increment LostCumulativeStatus[i, j] by 1
               if LostCumulativeStatus[i, j] > 2 then
                   send a continuous signal to Pilgrim (i, j)
               else
                   send flashing signal to Pilgrim (i, j)
           else
               LostCumulativeStatus[i, j] = 0
               Stop sending signal to Pilgrim (i, j), if any

7. Insert all values of LostPilgirmRecord structure into table PilgrimTracking (PilgrimID, TimeStamp,
   GPSLocation, Status, Distance)
8. Delete LostPilgirmRecord structure from the main memory
9. Repeat step 1 to 9 every 10 minutes
```

We suppose that there are *x* pilgrims and *m* pilgrims per group. The size of the pilgrim's GPS coordinate, timestamp, status and distance is *b* bytes. The size of each structure in the memory is calculated as follows:

- **PilgrimLocationRecord**: This structure contains the tracking information (GPS coordinates, timestamp, status and distance) of all pilgrims. It is created every one hour. The size of this structure is $\sum_{i=1}^{i=x} b$ bytes, where b number of bytes needed in each cell and x is the total number of pilgrims.
- **HashTable**: This structure contains the index of all groups with lost pilgrims. The size of this structure is $\sum_{i=1}^{i=n}(\lfloor\log_2 k\rfloor + 1)/8$ bytes, where n is the number of original groups (x/m) and k is the total number of groups with lost pilgrims.
- **LostPilgirmRecord**: This structure stores the tracking information of groups who have lost pilgrims. It is created and updated every 10 minutes. The size of this structure is $\sum_{i=1}^{i=k}(b * m)$ bytes, where k is the total number of groups with lost pilgrims and b is the size of each pilgrim's tracking information and m is the m is the maximum number of pilgrims per group.
- **LostCumulativeStatus**: This structure stores the number of consecutive lost status. It is created and updated evey 10 minutes. Each cell requires one byte to store the number of lost status. The size of this structure is $\sum_{i=1}^{i=k}(m)$ bytes.

Every hour the collective size of the existing memory structures (**PilgrimLocationRecord and HashTable**) is:

$$(\sum_{i=1}^{i=x} b) + (\sum_{i=1}^{i=n}(\lfloor\log_2 k\rfloor + 1)/8) \quad (3)$$

Every ten minutes, the collective size of the existing memory structures (**HashTable, LostPilgirmRecord** and **LostCumulativeStatus**) is:

$$(\sum_{i=1}^{i=n}(\lfloor\log_2 k\rfloor + 1)/8) + (\sum_{i=1}^{i=k}(b * m)) + (\sum_{i=1}^{i=k}(m)) \quad (4)$$

In practice, the required memory capacity of the aforementioned in-memory structure would likely be no more dozen megabytes for huge number of pilgrims. As per the ministry of haj report [reference], the number of haj in 2017 was 2.4 M and there were 20000 lost pilgrim during the five-day haj period. In average, there were 4000 lost pilgrims every day and 167 lost pilgrims every hour. The maximum size of each pilgrim tracking information (GPS coordinate, timestamp, status, distance) is 32 bytes.

The required memory capacity of the in-memory structures every one hour is:

$(\sum_{i=1}^{i=2.4M} 32)$ + $(\sum_{i=1}^{i=2.4}(\lfloor\log_2 167\rfloor + 1)/8)$ = 76.8 Mbytes + 2.4 Mbytes = 79.2 Mbytes

The required memory capacity of the in-memory structures every 10 minutes:

$(\sum_{i=1}^{i=2.4M}(\lfloor\log_2 167\rfloor + 1)/8)$ + $(\sum_{i=1}^{i=167}(32 * 20))$ + $(\sum_{i=1}^{i=167}(20))$ =

2.4 Mbytes + 0.11 Mbytes + 0.00334 Mbytes = 2.513 Mbytes

B. Performance Analysis

Algorithm 1 describes the process by which the central server receives and stores the tracking information of all pilgrims and identifies the lost pilgrims. The processing time overhead to support real time detection of astray pilgrims (Algorithm 1) can be estimated as follows:

- Time to create the two dimensional array *PilgrimLocationRecord[n, m]* is O(n * m), where n is the number of groups and m is the maximum number of pilgrims in a group
- Time to receive and store the GPS coordinates and timestamps of all pilgrims into *PilgrimLocationRecord* is O(n*m). More specifically, the time to receive each pilgrim's tracking information and to store it into the *PilgrimLocationRecord* is O(1) because the pilgrim ID is used to identify the corresponding group ID (row index) and the ID of the pilgrim (column index) within the group.
- Wost case time to calculate the centroid of all groups and to identify the astray pilgrims and to update the status of pilgrims in the *PilgrimLocationRecord* is O(m * n)
- Time to crate the HashTable and to insert the indexes of the groups with loast pilgrims is O(n)

The number of I/O required to flush the tracking information of all pilgrims (*PilgrimLocationRecord)* into the database disk storage is O($\left(\frac{\sum_{i=1}^{i=x} b)}{s}\right)$, where x is the total number of pilgrims (n * m), s is the size of disk block and b is the number of bytes required to store the tracking information of each pilgrim (GPS, timestamp, status and distance). Collectively, the CPU processing time to detect astray pilgrims is O(n * m) and the number of I/O to store the tracking information into the disk storage is O($\left(\frac{\sum_{i=1}^{i=x} b)}{s}\right)$ I/O.

Algorithm 2 shows the steps by which the central server receives and stores the tracking information of lost pilgrims more frequently (every 10 minutes). The processing time overhead to support real time detection of lost pilgrims more frequently (Algorithm 2) can be estimated as follows:

- Time to create the *LostPilgirmRecord* and *LostCumulativeStatus* is O(m * k), where m is the maximum number of pilgrims in a group and k is the total number of groups with lost pilgim(s).
- Time to receive and store the GPS coordinates and timestamps of lost pilgrims into *LostPilgirmRecord* is O(m*k).
- Wost case time to calculate the centroid of all lost groups, to identify the astray pilgrims and to update the status of pilgrims in the *LostPilgirmRecord* is O(m * k)
- Time to access the HashTable is O(k)
- Worst case time to access the *LostCumulativeStatus* and update the consecutive lost status is O(m*k)

The number of I/O required to flush the tracking information of lost pilgrims (*LostPilgirmRecord*) into the database disk storage is O($\left(\frac{\sum_{i=1}^{i=k}(b*m))}{s}\right)$, where k is the total number of lost groups s is the size of disk block and b is the number of bytes required to store the tracking information of each pilgrim (GPS, timestamp, status and distance). Collectively, the CPU processing time to detect astray pilgrims is O(m * k) and the number of I/O to store the tracking information into the disk storage is O($\left(\frac{\sum_{i=1}^{i=k}(b*m))}{s}\right)$ I/O.

The aforementioned theoretical analysis of the size and performance of the proposed architecture demonstrates that the in-memory structure requirements are of an affordable size and the processing overhead is within tolerated limits to support real time detection of astray pilgrims.

VI. Simulations and Results

Actual pilgrimage data showing the path of pilgrims and their geolocations are not currently available since the proposed system is the first of its kind. Therefore, we have used a synthesized data for the purpose of running simulation of the proposed algorithm. A random geolocation data sets were continuously generated within a specific geographical region that correspond to those visited by the pilgrim. Pilgrims were assigned to groups of a maximum of 20 person in each group. At each time slot, a new set of geolocations are generated for all the groups and are used to calculate the group's centroid. In order to simplify the simulation, it was assumed that a pilgrim is considered astray if their distance from the centroid is 1.5 kilometer. Simulation runs for various values of the total number of pilgrims (from 500,000 to 3000,000 pilgrims) were performed and the time to calculate the total number of pilgrims that actually went astray was measured. Fig. 4 shows the variation of the algorithm calculation time with the number of pilgrims. The results showed linear performance which is a good agreement with the analytical analysis.

VII. Conclusions and Future Work

In this work we presented the design and implementation of a distributed architecture and the research challenges of a pilgrim tracking, guiding, and astray-pilgrim detection system. The system consists of a client side that is a wearable device built as a system-on-chip and a server that stores personal information as well as the GPS coordinates and corresponding time stamps during the full duration of the pilgrimage journey (five days). The main objectives to be achieved in this work at the client side is that the wearable device is compact, low cost, and of low-power consumption to allow for a battery life to extend for at least five days. This include employing a power-efficient algorithm by properly selecting the inter-GPS fix times. The designed system automatically determines if a given pilgrim is potentially astray and is likely to miss one of the important rituals of the pilgrimage. On the server side, the entire pilgrimage model is stored including the geolocations of the regions and paths. The system automatically reports any lost or potentially lost pilgrims as well as alerts the pilgrim him/herself and sends alarms to the authorities for timely intervention. The in-memory structure used along with the algorithms enables real-time performance to avoid lengthy database queries. Simulation runs with synthesized data that is randomly generated within a given geographical location was performed. The simulation calculated the time to determine the number of pilgrims at a specific time slot as the number of pilgrims is varied. The simulation results showed a linear time performance which is in a good agreement with analytical analysis performed.

In future work and when real data is obtained, many useful data analytics can be performed which will avail and reveal new information that opens new door for better service and less astray pilgrims.

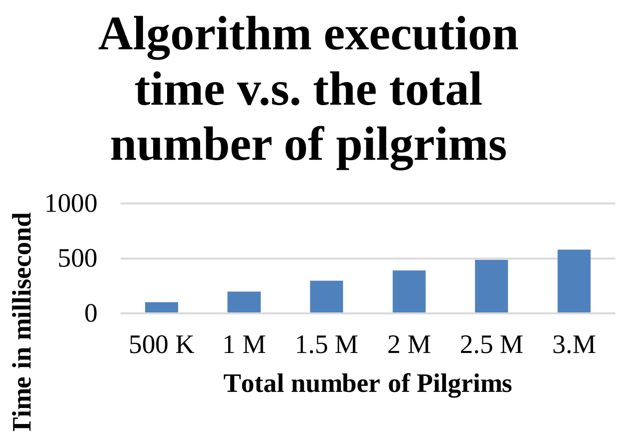

Fig. 4. Algorithm Execution Time vs. the Total Number of Pilgrims.

References

[1] "A study requesting to set upper age limits due to the spread of some diseases of the elders" 2006. [Online]. Available: http://www.alarabiya.net/articles/2006/12/28/30302.html [Accessed: 25 – Jan – 2018]

[2] Omar Elhalawy, "60% of pilgrims perform pilgrimage for the first time" 2017. [Online]. Available: http://www.alittihad.ae/details.php?id=41778&y=2017 [Accessed: 25 – Jan. – 2018]

[3] "The Origin and Activities of the ministry of Hajj and Umrah" 2014. [Online]. Available: http://www.haj.gov.sa/arabic/about/opendatapalteform/pages/ministrieorigination.aspx. [Accessed: 25-Jan.-2018].

[4] Shrooq Hisham, "For the first time, applying the electronic bracelet service in pilgrimage" 2016. [Online]. Available: http://www.hiamag.com/منوعات/اخبار/397051-لاول-مرة-تطبيق-خدمة-السوار-الالكتروني-في-الحج [Accessed: 25 – Jan. 2018]

[5] Mohandes, M., Turcu, C. "A Case Study of an RFID-based System for Pilgrims Identification and Tracking" in Sustainable Radio Frequency Identification Solutions, pp. 87–104. InTech, Dahran, Saudi-Arabia, 2010.

[6] Abeer Geabel, Khlood Jastaniah, Roaa Abu Hassan, Roaa Aljehani, Mona Babadr, Maysoon Abulkhair "Pilgrim Smart Identification Using RFID Technology (PSI)" International Conference of Design, User Experience, and Usability DUXU 2014.

[7] KC Rajwade, DH Gawali "Wearable Sensors Based Pilgrim Tracking and Health Monitoring system" International conference on Computing Communication Control and automation (ICCUBEA), 2016.

[8] "The 15 Best GPS Kids Trackers for Parents with Young Kids" 2016. [Online]. Available: https://www.safewise.com/blog/10-wearable-safety-gps-devices-kids/. [Accessed: 25 – Jan. 2018]

[9] M. A. R. Abdeen, "A Distributed Architecture and Design Challenges of an Astray Pilgrim Tracking System" The Fourth *IEEE International Conference* on Big Data Intelligence and Computing (DataCom 2018), Athens, Greece, August, 2018.

[10] H. Garcia-Molina and K. Salem, "Main memory database systems: An overview," IEEE Trans. Knowl. Data Eng., vol. 4, no. 6,pp. 509–516, Dec. 1992.

[11] V. Sikka, F. F€arber, W. Lehner, S. K. Cha, T. Peh, and C. Bornh€ovd, "Efficient transaction processing in SAP HANA database: The end of a column store myth," in Proc. ACM SIGMOD Int. Conf. Manag. Data, 2012, pp. 731–742

[12] Zhang, Hao, Gang Chen, Beng Chin Ooi, Kian-Lee Tan, and Meihui Zhang. "In-memory big data management and processing: A survey." IEEE Transactions on Knowledge and Data Engineering 27, no. 7 ,2015: pp. 1920-1948. D. B. Lomet, S. Sengupta, and J. J. Levandoski, "The Bw-Tree:

[13] A B-tree for new hardware platforms," in Proc. IEEE Int. Conf. Data Eng., 2013, pp. 302–313.

[14] V. Leis, A. Kemper, and T. Neumann, "The adaptive radix tree: ARTful indexing for main-memory databases," in Proc. IEEE 29th Int. Conf. Data Eng., 2013, pp. 38–49. Y. Li and J. M. Patel, "BitWeaving: Fast scans for main memory data processing," in Proc. ACM SIGMOD Int. Conf. Manag. Data, 2013, pp. 289–300.

[15] Z. Feng, E. Lo, B. Kao, and W. Xu, "Byteslice: Pushing the envelop of main memory data processing with a new storage layout," in Proc. ACM SIGMOD Int. Conf. Manag. Data, 2015.

[16] A. Pavlo, C. Curino, and S. Zdonik, "Skew-aware automatic database partitioning in shared-nothing, parallel OLTP systems," in Proc. ACM SIGMOD Int. Conf. Manag. Data, 2012, pp. 61–72.

[17] W. Rodiger, T.Muhlbauer, P. Unterbrunner, A. Reiser, A. Kemper, and T. Neumann, "Locality-sensitive operators for parallel mainmemory database clusters," in Proc. Int. Conf. Data Eng., 2014, pp. 592–603.

[18] V. Leis, A. Kemper, and T. Neumann, "Exploiting hardware transactional memory in main-memory databases," in Proc. Int. Conf. Data Eng., 2014, pp. 580–591.

[19] Z. Wang, H. Qian, J. Li, and H. Chen, "Using restricted transactional memory to build a scalable in-memory database," in Proc. 9th Eur. Conf. Comput. Syst., 2014, pp. 26:1–26:15.

[20] M.-C. Albutiu, A. Kemper, and T. Neumann, "Massively parallel sort-merge joins in main memory multi-core database systems," Proc. VLDB Endowment, vol. 5, pp. 1064–1075, 2012.

[21] S. D. Viglas, "Write-limited sorts and joins for persistent memory," Proc. VLDB Endowment, vol. 7, pp. 413–424, 2014.

[22] O. Polychroniou and K. A. Ross, "A comprehensive study of main-memory partitioning and its application to large-scale comparison-and radix-sort," in Proc. ACM SIGMOD Int. Conf. Manag. Data, 2014, pp. 755–766.

[23] A. Kemper and T. Neumann, "HyPer: A hybrid OLTP & OLAP main memory database system based on virtual memory snapshots," in IEEE 27th Int. Conf. Data Eng., 2011, pp. 195–206.

[24] R. Kallman, H. Kimura, J. Natkins, A. Pavlo, A. Rasin, S. Zdonik, E. P. C. Jones, S. Madden, M. Stonebraker, Y. Zhang, J. Hugg, and D. J. Abadi, "H-store: A high-performance, distributed main memory transaction processing system," Proc. VLDB Endowment, vol. 1, pp. 1496–1499, 2008.