Personal Health Record Storage on Privacy Preserving Green Clouds

Kirill Belyaev*, Indrakshi Ray*, Indrajit Ray*, and Gary Luckasen†
* Department of Computer Science
Colorado State University, Fort Collins, Colorado, USA
Email: {kirill, iray, indrajit}@cs.colostate.edu
† University of Colorado Health
Fort Collins, Colorado, USA
Email: Gary.Luckasen@uchealth.org

***Abstract*—With digitization there is a plethora of personal information, such as, health records and personal artifacts, that are stored on the data servers provided by the Internet companies. Such a solution is resource-intensive as the servers should be up and running. Moreover, the users no longer have complete control over their own data. We propose an alternative architecture where the data is no longer stored on Internet servers, but on set of new hardware devices called Secure Portable Tokens (SPTs) that are under the control of individual users. SPTs are cheap, portable, and secure devices that combine the computing power and tamper-resistant properties of the smart cards and the storage capacity of NAND flash memory chips. SPTs can be used to store personal data and can act as a Personal Data Server (PDS). In order to make such stored data reliable and available, we propose to have a set of SPTs storing personal data of individuals that form a cloud which we refer to as *Personal Data Server Clouds*. We provide protocols, based on the publish-subscribe paradigm, that demonstrate how replication and query processing are performed in the PDS clouds. We demonstrate the feasibility of our approach by developing a prototype PDS cloud that is geographically distributed and stores personal health records (PHRs).**

Keywords*—*Personal data servers, Green clouds

I. Introduction

In the digital age, there is a plethora of personal information that individuals need to store and access. Consider the storage of Personal Health Records (PHR) as a motivating example. A PHR system can store diverse information related to public health, such as, medical history and lab reports, insurance, consent forms, and other relevant information. The use of PHRs can improve the health care of individuals and communities in both developed and developing countries. A study by the California Healthcare Foundation [1] indicates that patients who have access to their health information systems are more likely to take actions for improving their health and health care. In addition, a well organized PHR system can empower a patient by allowing control over sharing medical history and symptoms. Consequently, many organizations are beginning to offer electronic PHR systems. One study estimates that there are over 200 different PHR systems in the USA [2].

We have various types of PHR systems. Some of these are supported by insurers or health care providers, such as the one deployed by Blue Cross of Northeastern Pennsylvania [3] and the My HealtheVet from the U.S. Department of Veteran Affairs [4]. Systems supported by a provider offer better integration with the provider's Electronic Health Record (EHR) system, but is constrained by the specific information made available by the provider. Moreover, if an individual has multiple providers, then consolidating all the health care information is challenging. The other alternatives are cloud based repositories, such as Microsoft HealthVault [5], where the user is responsible for uploading and managing her health information. All the above solutions make PHR systems valuable targets for attacks because they provide a high cost/benefit ratio to attackers compared to a single stand-alone system that contains the health records of an individual user. Moreover, there have been numerous incidents of confidentiality and privacy violations in commercial data servers, including those belonging to health care providers, arising from ill-defined or non-existing privacy policies, malware, negligence, errors, abusive use and external attacks [6] to assume that these PHR systems are immune to security breaches. A recent news article [7] indicates that consumers are wary about using online PHR systems because they do not trust their security.

Storing sensitive personal data, such as health data, on current cloud systems, such as Microsoft HealthVault [5], improves upon the systems supported by providers in that the users are responsible for managing their own data. However, such a solution, has several problems, including the potential for security and privacy breaches, that we outline below. First, service downtime and unavailability of Internet connection may limit an individual from accessing her data stored on the cloud. Clearly, this may be unacceptable to many users who expect their data to be available all the time. Second, such servers storing PHRs of a large number of users are also target for attackers who have much to gain by compromising the privacy of health data. Note that, attackers can be insiders as well, such as dishonest or disgruntled employees of the organization. Third, the individuals must trust these organizations for adequate protection of the data against security and privacy breaches. Typically, users are reluctant to place trust on these organizations [7]. Fourth, even though an individual trusts an organization and is willing to comply with its security and privacy policies, such policies are subject to change with time.

COLLABORATECOM 2013, October 20-23, Austin, United States
Copyright © 2013 ICST
DOI 10.4108/icst.collaboratecom.2013.254117

Moreover, security and privacy policies will almost certainly be changed if there is an acquisition or merger of companies. It is not clear how such a change will impact the existing records of an individual user and whether such a change will respect the original preferences of the user. Fifth, the servers storing data of a very large number of users must be kept online, which requires enormous power consumption. Consequently, we propose an alternative approach for storing PHR data.

Our proposed approach builds upon the use of a new emerging technology of cheap ($10-20 range), portable (carry it in your pocket / purse) and secure devices that combine the computing power and strong, tamper-resistant security of smart cards and the ever increasing storage capacity of NAND flash memory chips. We call such devices Secure Portable Tokens (or SPT in short). Examples of such SPTs can be found in wireless secure dongles, and smart USB jump drives with embedded chips. A very similar technology can be found in the SIM cards of mobile cellphones and smart phones.

The idea is to embed a full-fledged secure PHR data server and related software components within the tamper resistant hardware, and store the PHR data in encrypted manner in the accompanying flash storage. The application software as well as the user's PHR data will be protected by strong hardware-based security. No malware can damage the software since it is hardware protected. Moreover, the whole system is tailor made for PHR information, and hence will require very little administration from the user. To use the PHR data and to connect to different service providers, the SPT may need to be attached to a host device such as a smart phone or a computer. However, in such cases, the host device provides only an interface to the external world. All computations, communication, security policy enforcement, data storage and retrieval are still carried out by the SPT hardware. Security policies can be set or overridden only by the user. However, the policies are still enforced by the hardware.

Individuals own SPTs and they store their personal data on the SPT, forming what we refer to as a *Personal Data Server* (PDS). A group of such PDSs form a *personal data server cloud*. In order to build such a cloud, various research challenges must be addressed. However, we scope this paper and provide a simple baseline architecture and implementation for the cloud, without delving into the security and privacy aspects. Each PDS, storing PHRs, need not be connected to the Internet. An individual can get all his PHR information from his own PDS even though he is not connected to the Internet. However, PDSs may need to communicate with the external world to receive or send medical information to other entities, such as, health care providers or sharing their information with trusted friends.

Such communication is typically offline and is facilitated through an online broker that serves as a temporary storage for data exchange. We discuss the broker architecture in details and also the protocols that are needed to communicate with the external entities. The broker is implemented using the publish-subscribe model. The broker is online all the time and serves as a temporary storage for data exchange, but does not store the PHRs of the individuals. Thus, the broker storage is significantly smaller than that required to store the PHR data of all the users. The owner has complete control over what data he pushes to the broker for sharing by external entities and he also decides when such data is deleted from the broker.

The individuals have their PHRs on their own PDS. For reasons of availability, they may wish to replicate their PHRs on devices belonging to other individuals. The users have total control over what data they wish to replicate, where they would like to replicate, and may also change these decisions at any point of time. Thus, a user may replicate his data to one or more PDSs belonging to other trustworthy individuals.

We should state that there is an original socioeconomic assumption that replicas are chosen out of the reliable peers (such as family members or close friends) that will be responsible to keep the replica online periodically to ensure the adequate synchronization with the owner PDS.

Each PDS, under the control of an individual user, may be online or offline depending on the user's discretion. Replica propagation, in light of the fact that PDSs may be offline, poses a challenge. We have devised replica control protocols to address this issue. *Eventual consistency property* is assured provided the PDS holding the replica comes online after the update operation.

Queries may be posed on the PHR by individuals or organizations. Query processing poses similar challenges because a PDS containing the required PHR may be offline. In such a case, we demonstrate how the query response can be constructed from the PDS holding the original PHR or from the other PDSs containing the replica PHR. We try to minimize the response time for query processing, by trying to answer the query using the original or replica PDSs, whichever comes online first. We also guarantee the *data currency* property that ensures that the response to the query is always accurate, even if some replicas may have not been online all the time.

The remainder of the paper is organized as follows. We provide an overview of our approach in Section II. In Section III, we describe our architecture. We present our replication protocol in Section IV. We describe how queries are processed in Section V. We give our prototype implementation and experimental results in Section VI. We enumerate some of the related works in Section VII. We conclude the paper and point to future research directions in Section VIII.

II. Overview of Personal Data Server System

We propose a radically different approach to storing and managing personal health data in electronic format. The proposed approach builds on the new emerging technology of cheap ($10-20 range), portable (carry it in your pocket / purse) and secure devices that combine the computing power and strong, tamper-resistant security of smart cards and the ever increasing storage capacity of NAND flash memory chips. We call such devices Secure Portable Tokens (or SPT in short). Examples of such SPTs can be found in wireless secure dongles, and smart USB jump drives with embedded chips. A very similar technology can be found in the SIM cards of mobile cellphones and smart phones. Instead of storing PHR data on a centralized server, each user has her own SPT to securely store and carry the PHR data.

The idea promoted in this work is to embed a full-fledged secure PHR data server and related software components within the tamper resistant smart card hardware, and store the PHR data in encrypted manner in the accompanying flash storage. A schematic overview of this approach is shown in Fig. 1. The encrypted information on the flash storage can be accessed only via the smart card based data server. Thus, the application software as well as the user's PHR data will be protected by strong hardware-based security. No malware can damage the software since it is hardware protected. Moreover, the whole system is tailor made for PHR information, and hence will require very little administration from the user. To use the PHR data and to connect to different service providers, the SPT may need to be attached to a host device such as a smart phone or a computer. However, in such cases, the host device provides only an interface to the external world. All computations, communication, security policy enforcement, data storage and retrieval are still carried out by the SPT hardware. Security policies can be set or overridden only by the user. However, the policies are still enforced by the hardware. In this manner, user control of how her sensitive data is shared by others (by whom, for how long, according to which rule, for which purpose) can be fully re-established and convincingly enforced.

SPT owned by a user contains her PHRs, which can be accessed anytime by connecting the SPT to a host device. In fact, the user no longer needs an Internet connection to view her medical records. The user and the healthcare provider communicate through an online broker. The online broker serves as a temporary storage merely and does not store the PHRs of individual users. SPTs may get lost or stolen. Note that, even if an SPT gets lost, its contents are inaccessible to unauthorized personnel as we have cryptographic mechanisms for data protection that cannot be bypassed. However, we need to provide availability of the data even if a user loses his SPT. Consequently, we recommend that the user replicates her PHR data on multiple SPTs that are owned by her trusted peers. The user has complete control over which SPTs she would like to replicate the data, for how long, and she can change this decision at any point of time. Note that, even though the user's PHR is replicated on another SPT belonging to her peer, the peer can view the user's data only if she has provided an explicit authorization. The user's SPT communicates with its peers using the online broker. In the next few sections, we describe how the user's PHR is replicated to multiple SPTs and the mechanisms by which the PHR data can be queried. Note that, an SPT storing the PHR forms a PDS and a group of such PDSs form a PDS cloud.

III. Our Architecture

In order to make personal information reliable and available, we propose to replicate the information over various SPTs, referred to as PDS nodes, as shown in Fig. 2. Each PDS node is identified through a unique identifier, which we refer to as *PdsId*, and is owned by one user. Each PDS can take on at most two roles: *owner PDS* and *replica PDS*. A PDS acts as the *owner PDS* when it is dealing with the PHR record of the person who owns the PDS. A PDS acts in the role of *replica PDS* when it is involved with the PHR record of a person who is not the owner of that PDS. Note that, all PDSs possess the role of the *owner PDS*. However, only a select set of PDSs, may be assigned the role of *replica PDS*. We assume that a *replica PDS* contains the PHR of at most one user who is not the owner. However, an *owner PDS* may choose to replicate its data to multiple *replica PDSs*.

The PDSs communicate with each other via the broker using the publish-subscribe mechanism. The PDSs are responsible for pushing/pulling the information stored on the broker. Although the broker can have various types of architecture, we assume a centralized architecture in this paper. Thus, we have a network of PDS nodes that connect to a broker. The broker stores various types of information that are needed to communicate with the PDSs. The architecture of the broker is shown in Fig. 2.

The description of the various tables stored at the broker site are given below.

[Mappings Table:] The schema of this table is given by *mappings(PdsId, RpdsID)*. The *mappings table* stores information about owner to replica mappings, defined by the pair *(PdsId, RPdsId)*, where *PdsId* is the identifier of the *owner PDS* and *RPdsId* denotes the identifier of the *replica PDS*.

[Updates Table:] The *updates table* stores updates made by the *owner PDS* that must be propagated to its replicas. The schema of this table is given by *updates(PdsId, DataPacket)*. The *owner PDS* is identified by *PdsId* and the update that must be propagated is the *DataPacket*, which is represented as a BLOB holding java class object that encapsulates the data and methods of the corresponding type. Each *DataPacket* has a sequence number associated with it that identifies the unique PHR received from the *owner PDS*.

[Queries Table:] The *queries table* stores recent queries on the PHR data that must be answered. The schema of this table is given by *queries(PdsId, QPdsId, QId, QueryPacket)* where *PdsId* is the PDS issuing the query, *QPdsId* is the PDS which is being queried, *QId* is the unique query identifier, and *QueryPacket* is the BLOB (java class object) representing the actual query. In order to speed up query processing, this table is indexed by *QId* and *PdsId*.

[Results Table:] The *result table* stores the query responses that must be downloaded by the query issuer. The schema of this table is given by *results(PdsId, QId, ResultPacket)* where *PdsId* is the query issuer, *QId* is the query identifier, and *ResultPacket* is the BLOB representing query results. In order to speed up query processing, this table is indexed by *QId* and *PdsId*.

In addition, the broker also stores two main memory hashmaps of the following information.

[Replica Updates:] This is defined by the pair *(RPdsId,seqNo)* and it records the sequence number

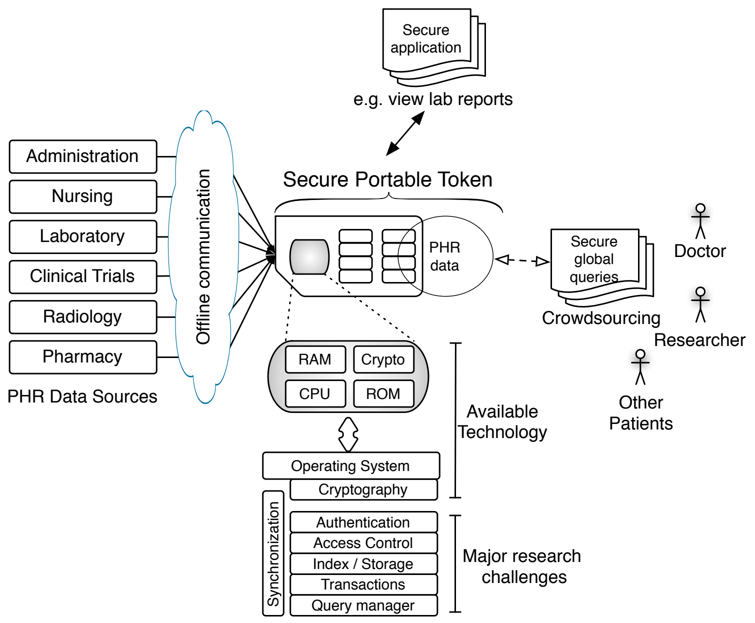

Figure 1. Personal Health Server Approach

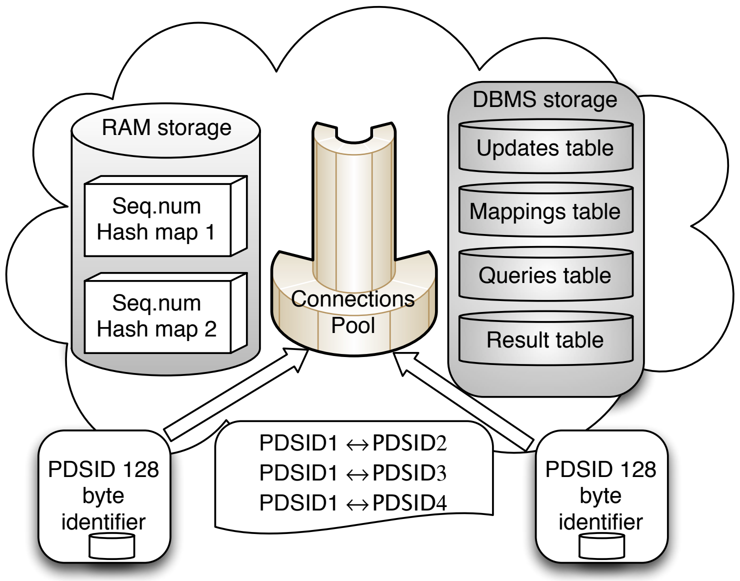

Figure 2. PDS Broker Architecture

associated with the latest update, denoted by *seqNo*, that is received by the *replica PDS* with identifier *RPdsId*.

[Owner Updates:] This mapping is defined by the pair *(PdsId, seqNo)*. The broker stores the sequence number of the latest update, denoted by *seqNo*, that it receives from the *owner PDS* with identifier *PdsId*.

IV. Replication Protocol

In order to ensure availability and reliability, the updates made to the PHR must be propagated to the replicas. In our architecture, the data owner has complete control over which PDS(s) will hold her replica at any given point of time and the replication protocol.

We should state that there is an original socioeconomic assumption that replicas are chosen out of the reliable peers (such as family members or close friends) that will be responsible to keep the replica online periodically to ensure the adequate synchronization with the owner PDS.

The owner PDS chooses the set of PDSs which are responsible for hosting its replicas. It does this by sending a *replica*

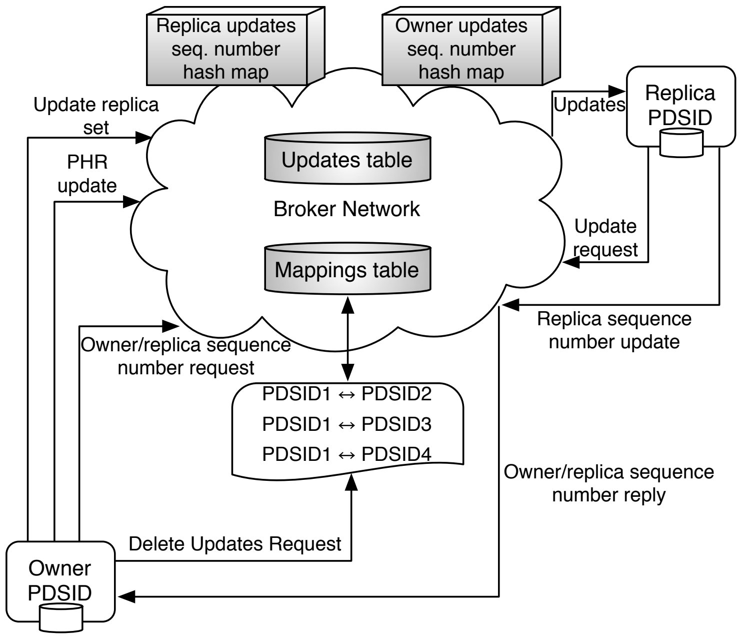

Figure 3. Replication Protocol

set message to the broker. The broker updates the *mapping table* that records the owner to replica mappings. The owner can change this set at any point of time. Note that, once an owner has revoked the replica status on a PDS, the replica will no longer receive new updates. However, if an owner decides to reinstate the replica status on this PDS, it will get all the updates that have not yet been deleted.

The new inserts into the PHR data are propagated through the broker which acts as a temporary storage. The owner PDS sends a *data update message* to the broker. The broker on receiving this message stores it in the *updates table* until it receives instructions from the *owner PDS* to delete it. In addition to the new PHR record, the *data update message* also contains a monotonically increasing sequence number, *seqNo*. The broker changes the memory hashmap *owner updates* to record the latest sequence number, *seqNo*, received from the *owner PDS*, *PdsId*.

The broker hosts the new PHR records until it receives the *delete request message* from the *owner PDS*. The *owner PDS* executes a *Replication Watchdog* that keeps track of replica freshness by sending periodic *replica seqNo request* to the broker in order to get the latest sequence number of the PHRs received by its replicas. It compares these sequence numbers with that of the last PHR it has sent; if a quorum of replicas has received the latest PHRs, then it sends a *delete request message* to the broker. The broker on receiving such a message from the *owner PDS*, deletes the PHRs belonging to the *owner PDS* from the *updates table*.

Each *replica PDS* executes a *Replication Manager* that is responsible for pulling the updates and keeping the replica that it hosts up-to-date. It does this by sending an *updates request message* to the broker. The broker retrieves the *owner PDS* corresponding to the replica and sends the corresponding messages stored in the *updates table*. If the sequence number, *seqNo*, associated with the response message is less than or equal to those updates it has already received, then the *replica PDS* does not insert this new data in its *update tables*. Note that, the broker sends new updates to replica only if there is a disparity in *seqNo*. The broker sends updates to the replicas only if the *seqNo* attribute in the *owner PDS* hash map table is different from the *seqNo* of the *PDS replica* stored in the replica hash map store. In such cases, the broker sends the records that are more recent than that given by the replica *seqNo* attribute.

Note that, the broker does not take any active role in maintaining replica consistency, but merely acts as a temporary storage. Clearly, the duration of storage depends on how often the *owner PDS* and *replica PDS* connect to the broker.

A. *Robustness of the Protocol*

In order to make our approach robust, we discuss how a PDS or the broker detects and solves the problem of lost messages. The *data update message* is used by the PDS to send its updates to the broker. The broker maintains the *owner update* hashmap that records the *seqNo* of the last message sent by the owner *PdsId*. In order to check whether the *data update message* was correctly received by the broker, the *owner PDS* sends it a *owner seqNo request message*. The broker sends the *seqNo update message* in response to announce the sequence number of the latest PHR received from the *owner PDS*. The response received from the broker can be compared with the sequence number of its latest PHR that it stores in its *myPds table*. If there is any discrepancy, the *owner PDS* resends its *data update message*. The *owner PDS* prevents out of order message arrival by not sending the next PHR before it verifies whether the previous one was received by the broker.

The *owner PDS* sends a *replicas set message* to inform the broker about its current set of replicas. If this message is lost, it can be detected subsequently when it issues a *replica seqNo request message* to the broker and receives response about the latest sequence number of the PHRs hosted at the various replicas. If the set of replica *PdsId* responding to this message does not match the ones that the *owner PDS* sent in the earlier *replicas set message*, then it knows that the earlier *replicas set message* was lost. In such a case, it retransmits the *replicas set message*.

The *owner PDS* sends a *delete request message* to the broker to delete all its PHRs as soon as they have been downloaded by its replicas. If this message is lost, the broker will continue to host the PHRs corresponding to the *owner PDS* until it receives the next *delete request message*.

The *owner PDS* sends the *replica seqNo request message* to get the sequence number of the PHRs stored in its replicas. It sends *owner seqNo request message* to get the sequence number of the latest PHR stored in the broker. If the *owner PDS* does not receive a response from a broker for any of the above messages, it retransmits the corresponding request messages.

The *replica PDS* sends an *update request message* periodically to the broker to get the latest PHRs sent by the *owner PDS*. It also sends a *seqNo update message* to the broker to announce the sequence number of the latest message it received. The broker can send the latest PHRs of the *owner PDS* if it notices any discrepancy.

V. Query Processing

We have two types of queries: *global queries* and *individual queries*. Global queries involve PHRs of a group of people. Consequently, the response of the query resides in several PDSs. Individual queries involve the PHR of one user, so the answer can be obtained from the *owner PDS* or any of its replicas. In this paper, we focus on individual queries only. We have an embedded relational database engine residing on the PDS that can process queries. The schema of our PHR table is as follows:

```
(hospitalName, patientName, medicine, date,
        doctorName, diagnosis)
```

Example queries supported by our system are given below.

```
[Q1:] SELECT COUNT(*) FROM ehr
[Q2:] SELECT date FROM ehr
        WHERE diagnosis = ``flu"
        AND prescription = "aspirin"
```

In the following, we use the term *query requester* (or simply *requester*) and *query responder* (or simply *responder*) to refer to the different roles involved in query processing. Note that, we do not focus on access control issues, but assume that the *requester* has permissions to query the PHR of some data owner. *Responder* may be the *owner PDS* or the *replica PDS* which has the PHR corresponding to the query.

We describe query processing in Fig. 4. *Requester* executes a *jobber* process that sends the *query message* to the broker (shown as Step 1 in Fig. 4). The query is temporarily stored in the *queries table*. The response to the query is stored by the broker in the *results table*. The *jobber* process in the *requester* sends *query result request message* (shown in Step 5 of Fig. 4) to get the response of the query. After receiving the responses from the broker (shown in Step 6 of Fig. 4), the *jobber* sends a *delete query message* to the broker to delete the query identified by *QId* and *PdsId* (not shown in Figure 4.) If the query result is resent to the requester, it signals that the *delete query message* has been lost. In such a case, the *jobber* sends the *delete query message* once again.

Responder has a *query manager* that sends the *query request message* (shown in Step 2 of Fig. 4) periodically to the broker to check whether there are any outstanding queries on the PHRs residing in it. If there are outstanding queries, the broker forwards them to the *responder* (shown in Step 3 of Fig. 4). The *responder* constructs a *query result message* (shown in Step 4 of Figure 4) and sends it back to the broker.

VI. Prototype Implementation and Experimental Results

The PDS model (both the broker and the PDS itself) has been implemented using Java SE (Standard Edition) using SQLite Java driver to work with the embedded database. We used MySQL for broker backend storage of the PDS storage tables.

We have deployed the PDS model prototype on the PlanetLab [8] testbed. Figure 5 shows the location of the different PDS nodes and their Round Trip Time (RTT) delays in milliseconds from PDS location to the broker network and back to the PDS. The broker infrastructure has been set up in a geographical Internet location that is different from the location of the PDS nodes. The backend database server and Broker process run on separate machines within the same network so that JDBC queries would not be affected by the network losses or delays. The PDS nodes have been deployed in United Kingdom, Poland, US (at Colorado State), Germany, Sweden, Portugal and Slovenia. No particular choice for location has been made except that the locations listed have been observed to possess relative stability of the machines uptime on PlanetLab global network. Most of the queries have been issued from US (Colorado State) location and tested with replicas or target PDSes answering the queries from the above listed geographical internet locations.

As we see from the plots different geographical locations incur different QoS values for the replication and query processing adding the RTT delays to the constant sleep periods of Replication Manager and Query Manager threads. Therefore replication and query processing delays have been defined as the sum of RTT from the source to target and back to carry the protocol messages plus the sum of sleep periods for Replication and Query Managers correspondingly at both source and target PDS nodes. Since both threads are expected to pull requests with a reasonable sleep period to spare the

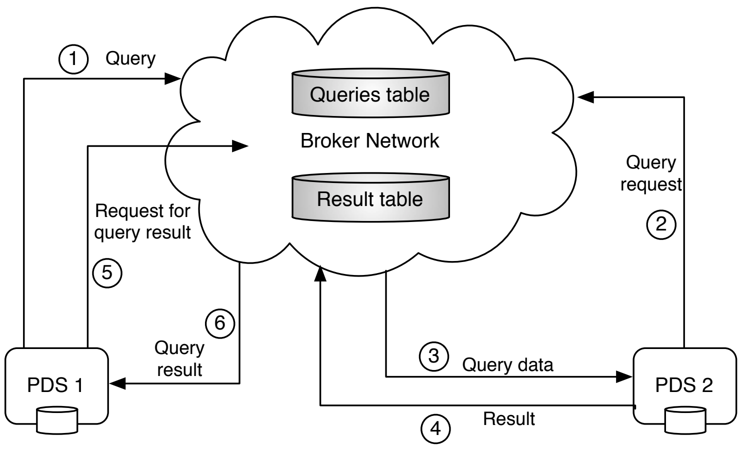

Figure 4. PDS Query Processing

broker computational resources (and network bandwidth) we could define the service as having the eventual consistency property when data is replicated and queried at unpredicted time intervals but with guaranteed eventual properties (provided that PDS nodes are able to go on-line).

Also the plots do not reflect the delays that would be introduced by broker serving requests from a large number of pulling nodes (more then a thousand) and sending JDBC queries to backend database with highly populated tables (introducing local query processing delays on the backend DB side). This would introduce additional service delays and is a subject for separate study with highly loaded broker instances. We did not provide the separate plot for the Replica Query Manager since the QoS would be identical to the Query Manager because both threads operate on the local embedded database entries (replicated database and the PDS own database respectively).

Note that, the tests of query processing do not introduce inconsistencies in query results when several replicas are answering queries, even if some of them go offline. This is possible because each *replica PDS* first downloads the updates before responding to any query. Thus, *data currency* is ensured by our protocol.

Sample queries that we executed are given below.

```
[Q1:] select * from ehr
      where medicine = "aspirin"
[Q2:] select * from ehr
      where medicine = "aspirin"
      and diagnosis = "flue"
[Q3:] select * from ehr
      where medicine = 'aspirin'
      and diagnosis = "flue"
      and hospital = "pvh"
[Q4:] select count(*) from ehr
```

The measured CPU and memory load on the Broker Java process has been observed to be relatively negligent with a testing PDS set (up to 12 nodes going online) ranging from 1 to 2 percent of CPU utilization. We anticipate that highly loaded Broker instance might be able to service a large number of pulling PDS subscriber nodes provided that JVM does correct CPU cores scheduling with underlying OS support.

Overall the developed system prototype has been deployed successfully with replication and query processing performed in the environment where PDS nodes are located in geographically dispersed locations and queries are issued on data distributed across Internet.

VII. Related Work

Personal Data Servers (PDS) suggests a radically different way of considering the management of personal data. The idea of using Secure Portable Tokens for storing personal data was advocated in one earlier work [9] where we presented some high level ideas. In this work, we look at replication and query processing at a much more detailed level with respect to health care applications, and have reported the results of actual implementation.

An attempt to facilitate data sharing in distributed Electronic Health Records (EHR) systems has been advocated in the research such as [10] where the secure and functional EHR system is proposed to support secure patient data sharing across cooperative organizations. We do not address the problem of how EHR can be shared across various organizations. However, we demonstrate how PHR data can be stored with individual users who have complete control over this data. The individuals can give the various health organizations access to their PHR via well-defined access control policies.

Our work bears some similarity to peer-to-peer (P2P) data management research. Commercial peer-to-peer systems (such as Gnutella, Kazaa and others) are rather limited when considered from the perspective of database functionality. These systems provide only file level sharing with no sophisticated content-based search/query facilities. They are developed as single-application systems that focus on performing one task, and it is not straightforward to extend them for other applications/functions [11]. Typical applications that can take

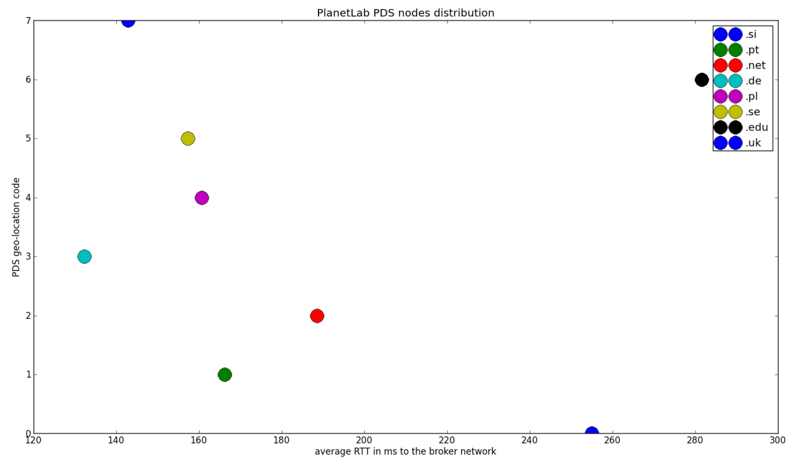

Figure 5. PDS nodes distribution on PlanetLab testbed

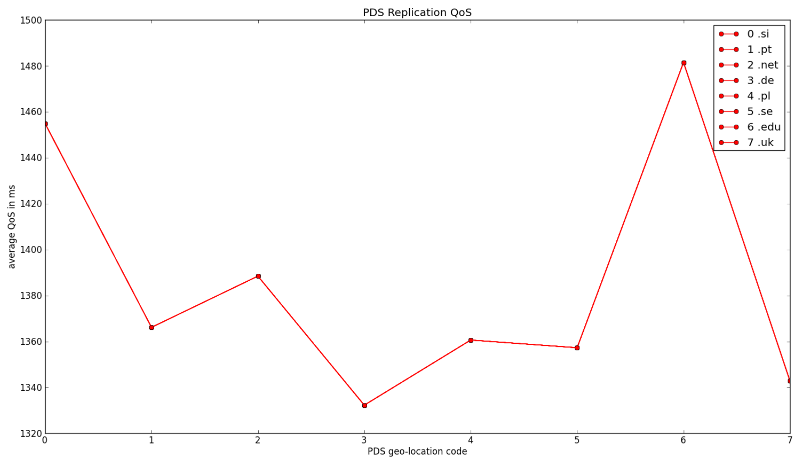

Figure 6. PDS replication QoS on PlanetLab testbed

advantage of P2P systems are probably lightweight and involve some sort of cooperation [12]. One significant difference with P2P systems is that our query-processing is very close to conventional databases. We can express most of the relational queries on any node unlike P2P systems [13]–[15]. This is effectively due to the use of embedded database engine in the PDS node that provides the underlying support for relational queries.

The need for database functionality embedded in various forms of lightweight computing devices is giving rise to personal folders on chips, networks of sensors and mobile computing devices [16]. Sensor networks gathering weather, pollution or traffic information have motivated several recent works [17] where a need arises to enable distributed query processing over a sensor network [18]. We do not focus on distributed query processing in this work, but instead on how data can be replicated across multiple nodes and query processed even when some nodes are not available.

PDS topology infrastructure uses the same concept of the overlay network - a similar infrastructure for all P2P systems, which is built on top of a physical network. PDS overlay uses unique 128 byte (and expandable) *PdsId* identifiers to distinguish nodes in the network from each other during query processing and replication operations. PDS overlay topology could be compared to the P2P hybrid networks that are commonly known as super-peer systems where some peers are responsible for controlling a set of other peers in their domain [12]. Examples of super-peer networks include Edutella [19] and JXTA [20]. PDS nodes are the nodes that interconnect with each other using a publish-subscribe model through broker network [21] that is somewhat different from the existing super-peer systems with better routing efficiency than that of peer-to-peer DHT alternative [12]. Thus, the PDS model borrows from P2P and publish-subscribe models; however,

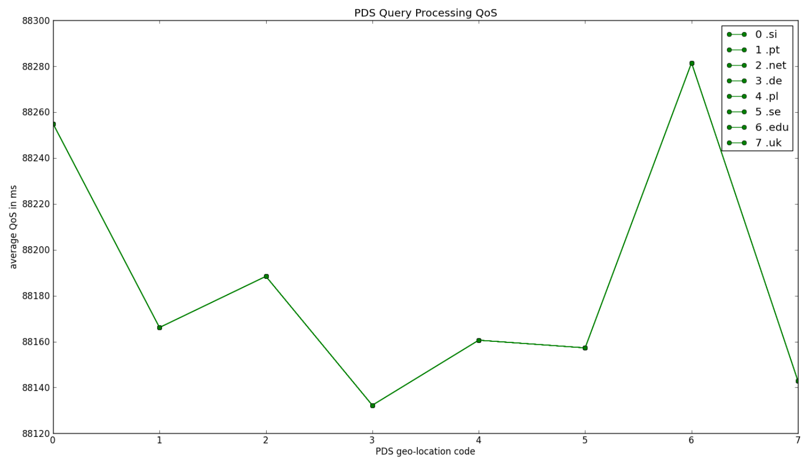

Figure 7. PDS Query Processing QoS on PlanetLab testbed

instead of relying heavily on DHT as is done in P2P systems, we use the broker network of supporting servers to carry out the efficient logical routing between the PDS nodes.

Certain applications, such as, agenda management, bulletin boards, cooperative auction management, and reservation management, require the data to be current. Replica consistency is particularly important in such applications. Supporting data currency in P2P systems is difficult as there is considerable dynamism in the peers joining and leaving the system. In P2P systems the problem is sometimes addressed by using data versioning [22] where each replica has a version number that is increased after each update. Data currency in PDS being used for PHRs is extremely important since the action taken by individuals/organizations depends on the accuracy of the health records. In PDS model, we not only ensure eventual consistency of replicated data but also ensure that queries issued by the user get accurate results.

PDS model also has some commonalities with the Local Relational Model (LRM) [13]. LRM assumes that the peers hold relational databases, and each peer knows a set of peers with which it can exchange data and services. The data residing in each peer may have semantic dependencies with that stored in the others. The authors propose coordination formulas that help resolve the semantic heterogeneity of the data stored at different peers. We do not consider semantic heterogeneity in this paper at all. We focus on how PHRs can be stored and replicated in low powered devices which are under the control of individual users.

VIII. Conclusions and Future work

In this work, we investigated how PHRs of individuals can be stored on PDSs such that individuals have more control over their data. The data of an individual is stored on the owner's PDS and it is replicated on the PDS of one or more individuals who are trusted by the owner. We proposed an architecture for PDS cloud and discussed how the data can be replicated and simple queries processed. Our replication algorithm guarantees eventual consistency provided the PDS holding the replica comes online after the update. Our query processing ensures that query response time is minimized and the latest updates are reflected in the query response. We built a prototype and performed experiments using the PlanetLab to demonstrate the feasibility of our approach.

A lot of work remains to be done. Our next step is to investigate how to process global queries, such as aggregate queries, where the answer lies in getting the information from multiple PDSs. We also plan on providing protocols for query processing when the replica is not stored in one PDS, but is partitioned across several PDSs. We need to investigate what types of partitioning schemes work best for applications involving PHRs.

Security issues also have not been addressed at all in this work. We plan on how to formally specify access control, retention, audit, and privacy policies for PHR applications. Such formalization is needed to analyze inconsistencies and conflicts among the different set of policies. One future work is formal specification and analysis of policies and their real-time enforcement using SPTs.

We have not yet addressed data privacy and anonymity issues in the current implementation. We plan to achieve basic network security via TLS encryption of the protocols and use TOR routing to anonymize the connection to the broker network. The temporary data stored in the broker network must be protected to protect against honest but curious broker [9]. In a previous work [23], we have addressed some of these data privacy and anonymity issues in the context of a centralized broker. Our future plans include extending these techniques for a distributed architecture that will work with the SPTs to provide efficient storage and retrieval of PHRs.

We plan to implement a reference monitor in the secure hardware of SPTs to enforce such policies. We will investigate how to store data on an encrypted form on the broker, and how such data can be securely and efficiently accessed. We plan to provide secure and anonymous protocols for communication with the broker to ensure the privacy of PDS users. We also plan to investigate how to partition the data, such that no single replica PDS has enough information to cause leakage of sensitive health data.

Acknowledgment

We wish to thank Mr. Fernando Pedroza and Ms. Brenda Harstad at the University of Colorado Health for helping us understand the requirements of the medical domain. This work was partially supported by the National Science Foundation under Grant 0905232. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

[1] California Healthcare Foundation, "Consumers and Health Information Technology: A National Survey," http://www.chcf.org/publications/2010/04/consumers-and-health-information-technology-a-national-survey, Apr. 2010, date accessed: July 31, 2013.

[2] C. J. Gearon, "Perspectives on the Future of Personal Health Records," http://www.chcf.org/publications/2007/06/perspectives-on-the-future-of-personal-health-records, June 2007.

[3] M. K. McGee, "Forget the Chart, Check the Cell Phone – Insurer will let members access their health data from cell phones," InformationWeek. http://www.informationweek.com/forget-the-chart-check-the-cell-phone/197001014, Jan. 2007.

[4] U.S. Department of Veteran Affairs. MyHealtheVet. [Online]. Available: http://www.myhealth.va.gov/mhv-portal-web

[5] Microsoft Corporation. HealthVault. [Online]. Available: http://www.healthvault.com/Personal/index.html

[6] Open Security Foundation. OSF DataLossDb – Data Loss News, Statistics, and Research. [Online]. Available: http://datalossdb.org

[7] L. Mearian, "Consumers Remain Wary of Personal Health Records," News article in ComputerWorld http://www.computerworld.com/s/article/9215996/Consumers_remain_wary_of_personal_health_records, Apr. 2011.

[8] PlanetLab Consortium. PLANETLAB: An Open Platform for Developing, Deploying, and Accessing Planetary Scale Services. [Online]. Available: http://www.planet-lab.org

[9] T. Allard, N. Anciaux, L. Bouganim, Y. Guo, L. L. Folgoc, B. Nguyen, P. Pucheral, I. Ray, I. Ray, and S. Yin, "Secure Personal Data Servers: A Vision Paper," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 25–35, Sep. 2010. [Online]. Available: http://dl.acm.org/citation.cfm?id=1920841.1920850

[10] J. Sun and Y. Fang, "Cross-domain data sharing in distributed electronic health record systems," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 21, no. 6, pp. 754–764, 2010.

[11] B. C. Ooi, Y. Shu, and K. L. Tan, "DB-Enabled Peers for Managing Distributed Data," in *Procedings of the Fifth Asia-Pacific Web Conference (APWeb 2003)*, 2003, pp. 10–21.

[12] M. T. Özsu and P. Valduriez, *Principles of Distributed Database Systems*, 3rd ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2011.

[13] P. A. Bernstein, F. Giunchiglia, A. Kementsietsidis, J. Mylopoulos, L. Serafini, and I. Zaihrayeu, "Data Management for Peer-to-Peer Computing : A Vision," in *Proceedings of the 5th International Workshop on the Web and Databases (WebDB)*, 2002, pp. 89–94.

[14] N. Daswani, H. Garcia-Molina, and B. Yang, "Open Problems in Data-Sharing Peer-to-Peer Systems," in *Proceedings of the 9th International Conference on Database Theory (ICDT)*. London, U.K.: Springer-Verlag, 2003, pp. 1–15. [Online]. Available: http://dl.acm.org/citation.cfm?id=645505.656446

[15] P. Valduriez and E. Pacitti, "Data Management in Large-Scale P2P Systems," in *Proceedings of the 6th international conference on High Performance Computing for Computational Science (VECPAR)*. Berlin, Heidelberg: Springer-Verlag, 2004, pp. 104–118. [Online]. Available: http://dx.doi.org/10.1007/11403937_9

[16] Kyu-Young Whang and Il-yeol Song and Taek-yoon Kim and Ki-hoon Lee, "The Ubiquitous DBMS," *SIGMOD Record*, vol. 38, no. 4, pp. 14–22, Dec. 2009.

[17] G. Bernard, J. Ben-othman, L. Bouganim, G. Canals, S. Chabridon, B. Defude, J. Ferrié, S. Gançarski, R. Guerraoui, P. Molli, P. Pucheral, C. Roncancio, P. Serrano-Alvarado, and P. Valduriez, "Mobile Databases: a Selection of Open Issues and Research Directions," *SIGMOD Record*, vol. 33, no. 2, pp. 78–83, Jun. 2004. [Online]. Available: http://doi.acm.org/10.1145/1024694.1024708

[18] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "TinyDB: an acquisitional query processing system for sensor networks," *ACM Trans. Database Syst.*, vol. 30, no. 1, pp. 122–173, Mar. 2005. [Online]. Available: http://doi.acm.org/10.1145/1061318.1061322

[19] W. Nejdl, W. Siberski, and M. Sintek, "Design Issues and Challenges for RDF- and Schema-Based Peer-to-Peer Systems," *SIGMOD Record*, vol. 32, no. 3, pp. 41–46, Sep. 2003. [Online]. Available: http://doi.acm.org/10.1145/945721.945731

[20] S. Microsystems. JXTA: The Language and Platform Independent Protocol for P2P Networking. [Online]. Available: http://jxta.kenai.org/

[21] L. Vargas, J. Bacon, and K. Moody, "Integrating Databases with Publish/Subscribe," in *Fourth International Workshop on Distributed Event-Based Systems (DEBS)*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 392–397. [Online]. Available: http://dx.doi.org/10.1109/ICDCSW.2005.79

[22] P. Knezevic, A. Wombacher, and T. Risse, "Enabling High Data Availability in a DHT," in *Proceedings of the 2nd International Workshop on Grid and Peer-to-Peer Computing Impacts on Large Scale Heterogeneous Distributed Database Systems*, 2005, pp. 363–367.

[23] I. Ray, K. Belyaev, M. Strizhov, D. Mulamba, and M. Rajaram, "Secure Logging as a Service - Delegating Log Management to the Cloud," *IEEE Systems Journal*, vol. 7, no. 2, pp. 323–334, 2013.