Persisting and Querying Biometric Event Streams with Hybrid Relational-XML DBMS *

Daby M. Sow
IBM T. J. Watson Res. Ctr.
19 Skyline Dr.
Hawthorne, NY, USA
sowdaby@us.ibm.com

Lipyeow Lim
IBM T. J. Watson Res. Ctr.
19 Skyline Dr.
Hawthorne, NY, USA
liplim@us.ibm.com

Min Wang
IBM T. J. Watson Res. Ctr.
19 Skyline Dr.
Hawthorne, NY, USA
min@us.ibm.com

Kyu Hyun Kim
IBM Ubiquitous Computing Laboratory
67-12 Dogok-Dong
Gangnam-Gu Seoul, South Korea
andrew.kim@kr.ibm.com

ABSTRACT

Remote monitoring of patients' biometric data streams offers the possibility to physicians to extend and improve their services to chronically ill patients who are away from medical institutions. This emerging technology is a promising way to address important aspects of the cost issues that most health care systems are experiencing. In order to fulfill its potential, several challenges need to be overcome. First, the data collected needs to be filtered and annotated intelligently to help physicians cope with and navigate the large amount of patient sensor data received as a result of large scale remote health monitoring deployments. Secondly, efficient stream persistence and query mechanisms for these data need to be designed to satisfy health care regulations and help physicians track patient health histories accurately and efficiently. In this paper, we concentrate on the second challenge. We leverage emerging hybrid relational-XML database management systems to design a storage sub-system for remote health monitoring. We evaluate this approach by performing series of performance tests to assess the ability of the proposed system to handle the huge amount of biometric data streams requiring persistence.

Categories and Subject Descriptors

H.3.4 [**Systems and Software**]: Distributed systems

General Terms

Design

Keywords

event processing, stream processing, hybrid XML database, relational database, stream database

1. INTRODUCTION

Cost problems in health care are making the headlines in many countries. In the United States, it has been reported that more than 1.9 trillion dollars were spent on health care in 2004 [5]. More than 75 % of this amount was spent to treat patients with chronic conditions including cancer, diabetes and congestive heart failure [20]. These alarming figures, coupled with the fact that the American society is aging rapidly (in part because of the aging of the baby boomer generation), are clear indicators that the United States are in the midst of a serious health care crisis. Similar indicators are also present in many other industrialized nations.

In order to face these problems, several alternate ways of improving the delivery of health care services are being investigated. Of particular interest to us is the concept of remote health monitoring, in which medical care is ubiquitously provided to patients with chronic diseases. A well-designed remote health monitoring system could not only reduce the number of patient visits to hospitals by monitoring them continuously while performing their daily activities, but could also reduce the incidence of diseases by being more proactive and by detecting early symptoms of chronic diseases. Early detection of symptoms often enables early and complete recovery from these illnesses. In addition, it yields significant cost savings since health care costs are known to rise drastically while the condition of patients is deteriorating [18].

As shown in Figure 1, a remote health monitoring system is typically built in a three-tier architecture [11]. The first tier is the sensor network, consisting of inexpensive biometric sensors reporting various kinds of biometric readings taken on the patients. Typical sensors are rather simple: e.g., blood pressure cuffs, pulse-oximeters, weight scales and

*This work was supported by the IT R&D program of MIC/IITA under the project number 2006-S-602-01 (Development of Stream-based Distributed Interoperable Health care Infrastructure Supporting Provenance and QoE).

DEBS '07, June 20–22, 2007 Toronto, Ontario, Canada

glucometers. However, with the recent progress in sensor technologies, we have seen the emergence of devices that generate more complex events at higher rates: e.g., multi-channel Electrocardiogram (ECG/EKG) sensors able to produce event rates in the order of hundreds of kbps. In addition, the advances in low power wireless technologies have enabled many sensor manufacturers to equip their products with short range wireless radios (e.g., Bluetooth, ZigBee) for the transmission of data to external components.

Events transmitted out of the sensor tier are received by the second tier of the system, the data hub tier. At this layer, a personal device belonging to the patient is typically used to package the received sensor data and translate it from a sensor device specific format into a common format that is well understood by the rest of the system. This tier resides either on a static machine (e.g., the patient's personal computer), or on some mobile device (e.g., a high end cellular phone belonging to the patient). In either case, it is important to keep in mind that there is at least one active hub in the system for each patient. Hence, the number of data streams ingested by the system is directly proportional to the size of the patient population.

The data hub tier is also responsible for the transmission of the formatted data to the last tier, the server tier, where analysis of the data takes place. The need to analyze raw sensor readings is dictated by the volume of data collected for medical professionals. To illustrate this point, consider a hypothetical large scale deployment of remote health monitoring kits observing patients with diabetes in the United States. Simple mathematics taking into account the ratio of the total number of diabetics to the total number of endocrinologists in the United States reveals that more than 20,000 biometric readings might be directed to one doctor on a daily basis. Clearly, these readings need to be filtered or preprocessed for the physicians. What is needed is an infrastructure that is able to extract medically important events from the huge amount of data readings.

Several analysis applications might be instantiated on the server tier, based on the current condition of the patient. For example, a type I diabetes monitoring application might be in place to pay particular attention to glucometer readings and alert physicians or even Emergency Response teams when blood glucose levels are abnormal. A congestive heart failure application might also be in place to correlate sharp patient weight increases with blood pressure and arrhythmia features extracted from ECG sensor readings to monitor the well-being of patients diagnosed with congestive heart failure. In more general cases, a generic application might be deployed for users interested in monitoring their general well-being. In such cases, basic biometric readings such as blood pressure, pulse, temperature, might be collected and analyzed at the server tier by generic well-being programs tuned to look for symptoms of common chronic diseases.

Another important function of the server tier is the persistence of the collected data. This function is a requirement on the system, mainly for two reasons. The first one is related to regulations in health care. Indeed, an increasing number of storage regulations are being imposed by governments on medical institutions [12]. In the United States, the Health Insurance Portability and Accountability Act (HIPAA) has issued strict guidelines in enforcing medical institutions to store patient medical records for many years (up to 21 years) [12]. The second reason is simply due to the fact that medical professionals might need to access detailed patient medical record histories, may it be for pure research purposes or simply for historical analysis in order to provide better services.

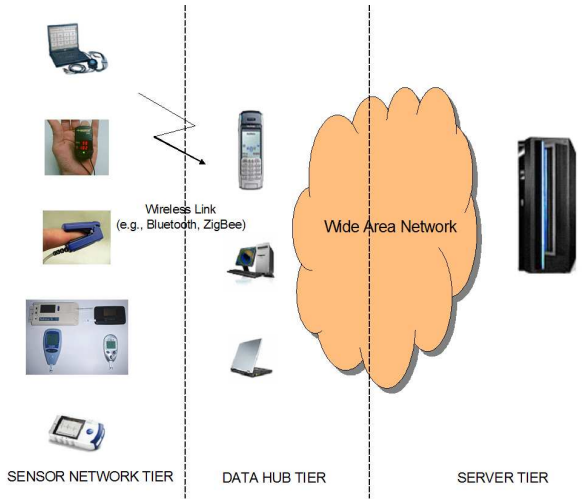

Figure 1: The three tier approach to remote health monitoring.

Designing a stream storage component at the server tier is typically done leveraging relational database management systems. However, these approaches tend to be difficult to extend as new data sources appear in the system. With the emergence of hybrid relational-XML database systems, more effective approaches become possible. Indeed, with XML becoming the standard for data retrieval and exchange, many commercial RDBMSs now support XML data in its native form. The XML support adds an extensibility dimension to traditional relational DBMS that we intend to leverage in the design of the server tier.

In this work, we evaluate the extent to which hybrid relational-XML DBMS technologies can be used to store and query large volumes of a remote health monitoring data. We start with a review of the related work discussing known approaches to design the server tier. We then propose a system architecture for the storage component of the server tier. Our approach leverages a hybrid relational-XML database system. We evaluate its performance by a series of experiments and end the paper with a summary on the capabilities of the proposed system and a discussion on future work.

2. RELATED WORK

Work related to this paper falls into two main categories: remote health monitoring systems and streaming database systems. In this section, we review the state of the art in these areas.

The concept of remote health monitoring has been around for several years. In fact, several proprietary commercial systems are available on the market. Honeywell, Health Hero Network, American Telecare, and AMD Telemedicine are notable companies providing closed, proprietary remote health monitoring system offerings. In the research literature, several systems have also been proposed. Most of them are also proprietary and not open to new data sources. One notable exception is the work by Blount et. al. in [11] where

they presented an open platform for remote health monitoring. However, the system focuses on the collection of discrete events. It does not support continuous data streams (e.g., ECG data).

The problem of persisting and querying remote health monitoring data streams has received little attention in the scientific community. Bar-Or et. al. are one of the rare research team investigating this issue in the BioStream system, as reported in [8]. However, [8] does not propose an approach, nor does it mention anything about the openness and extensibility of BioStream.

The stream database system is a relatively new concept describing a new class of data management systems designed to handle streaming data. The focus of these systems is more on the ability to make long standing queries on incoming streams, than on the ability to persist efficiently incoming streams. There are several projects such as Aurora [6] and Borealis [7] that are attempting to build such stream database systems with goal to allow applications to manipulate and query stream data from a database centric perspective. Queries for data still follow the models used in traditional relational database systems. To be specific, data streams are queried over the relational data attributes and time windows. The approach discussed in this paper takes these concepts to the next level by exploiting the extensibility of the schemas supported by hybrid relational-XML database systems to build extensible persistency mechanisms.

3. SYSTEM REQUIREMENTS

In this section, we enumerate the key requirements to a storage system for the persistence and query of patient biometric data.

- *Event rate scalability:*
  While the event rates of the biometric data streams from an individual patient may not be very high, the aggregation of streams across the patient population produces very large event rates that might stress the system significantly. Assuming a patient population of ten thousand with an average data rate per patient of 100 kbps[1], the system would need to handle rates in the order of 1 Gbps. From a storage perspective, it is imperative to ensure that the underlying storage system can cope with such high rates.

- *Long-term storage of streams:*
  The need to persist the sensor readings is dictated by several factors. Historical analysis of streams requires some form of persistence of streams. For example, an analysis algorithm tracking patient weight gains needs to keep the history of the weight of the patients. Another algorithm tracking the effect of a drug on a patient needs past biometric data to evaluate the effect of the drug. The physicians may also bring persistence requirements to the system. Indeed, a physician might simply be interested in accessing historical data about her/his patients to make better assessments on their health. Finally, health care regulations are also bringing stream persistence requirements. The management of this large volume of data will require substantial investments from medical institutions. However, these investments should be offset by the overall savings created by the wide spread adoption of remote health monitoring technologies.

- *Extensibility:*
  The open nature of the health care environment dictates the need for an extensible storage system. The heterogeneity of existing biometric sensors and the dynamism at which new devices are appearing in the market forces us to design an extensible platform, open to new data types generated by new devices.

- *Unbalanced read/write ratios:*
  While the rates at which data enter the storage system might be high, we do not expect this to be the case for the rates at which data are queried by the end users, may it be humans (e.g., physicians) or machines (e.g., analysis component performing historical queries for data). The main assumption is that out of all the data captured by the system, only a small fraction is of interest to the end users.

- *Expressiveness of the stream query language:*
  The end users need to access data of interest through a query language for streaming data. The key requirement on the query language is its expressiveness: users should be able to query not only the data attributes in the original biometric streams, but also higher level attributes that are extracted by processing the data streams as they enter the system.

[1]ECG data sources alone operate in this range.

4. SYSTEM ARCHITECTURE

The system requirements outlined in the previous section have been used to architect the storage subsystem of the server tier. As shown in Figure 2, this architecture consists of two major components: a pre-processing subsystem and a hybrid relational-XML DBMS. In the following, we describe each of these components in details.

4.1 The Pre-processing Subsystem

The pre-processing subsystem is an extensible framework where event streams are annotated. Along with the framework comes an extensible data model for the representation of the streams. This data model is instantiated by an extensible type system. When a new data source is registered with the framework, a type for this new data source is created by the developers, by extending the system type system. For instance, the registration of ECG streams requires an extension of the type system to define an ECG type. While we do not restrict the way a developer is extending the type system at this point, we foresee the emergence of a vocabulary for biometric streams that will be compatible with standard data models of the future. For example, HL7[2] defines an XML representation for biometric streams, including ECG. These representations define XML elements and attributes that are semantically relevant to medical professionals. Accordingly, we would expect that developers may want to leverage these elements and attributes in their type systems and applications.

As streaming data enters the system, it is first de-multiplexed by type. The output of this operation gets packaged into

stream segments called *chunks*. A chunk is an atomic element describing the highest level granularity at which analysis algorithms may operate. In our model, we expect hubs at the second tier of the architecture to generate stream segments while the Type Demux element shown in Figure 2 is responsible for de-multiplexing stream segments based on their types and also for creating chunks that are object representations of these segments. Accordingly, arbitrary sets of chunks with size defined by the developer may traverse a set of pre-defined annotators associated with its type. These annotators generate annotation metadata encapsulated in the attributes defined by the stream type.

Our framework architecture is open. It allows any annotator to be plugged according to a well defined API. It is architected on top of the Unstructured Information Management Architecture (UIMA) [16]. UIMA is an open source platform for the development of unstructured information analysis applications. Although mainly used for text analytics, UIMA is designed to support different modalities. In fact, Belinski et. al. have demonstrated that UIMA can be used for the mining and correlation of presence data [9]. In this paper, we aim at leveraging UIMA as a platform for the development of our pre-processing subsystem using similar approach as in [9].

The output of each chain of annotators is consumed by a special component called the consumer. The consumer extracts all the annotations that have been computed by the corresponding chain of annotators. It represents these annotations in an XML format before passing them together with the related chunks to a hybrid relational-XML DBMS.

The consumer also transmits annotated chunks to a stream analysis framework where further analysis of the data takes places. This framework is also UIMA based. It allows developers to write analysis components able to perform arbitrarily complex operations. For example, the identification of medically significant events could be encapsulated in a chain of analysis components. These analysis components may also access the hybrid relational-XML DBMS for historical data. The design of this part of the system is beyond the scope of this paper.

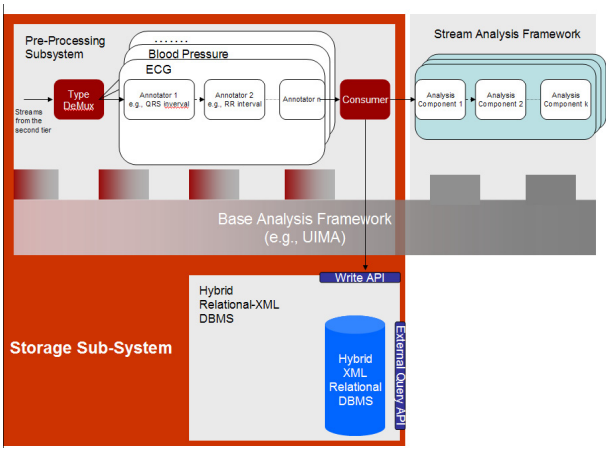

Figure 2: System architecture

4.2 Hybrid Relational-XML DBMS

We use a hybrid relational-XML DBMS, IBM's DB2 V9 PureXML [19, 23, 22], in our system to store and query annotated chunks.

In a hybrid relational-XML DBMS, XML is supported as a basic data type. Users are allowed to create a table with one or more XML type columns. A collection of XML documents can therefore be defined as a column in a table. For example, a user can create a table `ECGMessage` to store ECG data encoded in the HL7 XML format as follows:

```
CREATE TABLE ECGMessage(id integer,
                        sender VARCHAR(27),
                        msg XML);
```

Inserting an XML document into a table involves several operations. The document must first be parsed. XML parsing is known to be CPU expensive [21] and might affect the ability of the system to sustain high input rates. However, we will see in the Section 5 that parsing optimizations made on this DBMS allow it to support large number of input streams. After the XML parsing step, the document is placed into the native XML storage of the DBMS before being indexed.

```
INSERT INTO ECGMessage(1, 'ECGLABX',
XMLParse( Document '<?xml version='1.0'>
  <AnnotatedECG>
    <id root="728989ec-b8bc-49cd-9a5a-30be5ade1db5"/>
    <effectiveTime>
      <low value="20021126084800.000" inclusive="true"/>
      <high value="20021126084810.000" inclusive="false"/>
    </effectiveTime>
    ...
  <AnnotatedECG>'));
```

The validation of the document against an XML schema is an optional step that can be enabled for insert operations. Users can query relational columns and XML columns together by issuing SQL/XML queries [17, 14, 15]. For example, The following query returns the identifiers of all ECG message whose effective time covers the time point "20021101000000.000":

```
SELECT id
FROM ECGMessage AS E
WHERE XMLExists('$t/AnnotatedECG/effectiveTime/[
      low/@value < 20021101000000.000 and high/@value >
      20021101000000.000 ]'
      PASSING BY REF E.msg AS "t")
```

In this query, XMLExists is an SQL/XML boolean function that evaluates an XPath [4] expression on an XML value. If XPath returns a nonempty sequence of nodes, then XMLExists is true, otherwise, it is false.

In addition to the XMLExists function, another SQL/XML function, XMLTable can be used to transform XML data into a more relational format. The XMLTable function creates a virtual relational table using information from XML data specified using XPath. For example, the previous query can also be written using the XMLTable function as follows,

```
SELECT E.id, X.effTimeLow, X.effTimeHi
FROM ECGMessage AS E,
     XMLTable('$t/AnnotatedECG/effectiveTime/'
     PASSING BY REF E.msg AS "t",
     COLUMNS
     effTimeLow DOUBLE PATH 'low/@value',
     effTimeHi DOUBLE PATH 'high/@value') AS X
WHERE X.effTimeLow < 20021101000000.000
      AND X.effTimeHi > 20021101000000.000;
```

In this case, the XMLTable function creates, for each row in the `ECGMessage`, for each instance of `/AnnotatedECG/effectiveTime/`, a row in the virtual table `X(effTimeLow, effTimeHi)` with values extracted from the XML via the specified XPath.

4.3 Storing Data Streams

The pre-processing subsystem transforms the streaming data into chunks and associated annotations. These chunks and annotations are then persisted in a hybrid relational-XML DBMS. For extensibility, the annotations associated

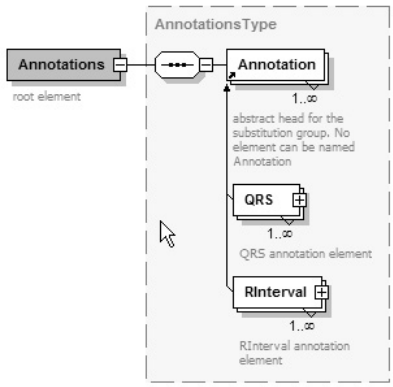

Figure 3: Schema for storing the annotations in XML.

with each chunk are stored as an XML document. A visual representation of the XML schema for the annotations used in our ECG example is shown in Figure 3. The XML document for our example consists of a list of `QRS` and `RInterval` elements under the root `Annotations` element. Schema type inheritance and substitution groups are used for extensibility in the example, but the `choice` construct can be used as well. Adding a new annotation type can be achieved simply by creating a new complex type for the new annotation and associating the new complex type with the annotation substitution group. The ability to easily add new annotation types to the schema is especially important to the biometric streams domain, because it allows diagnostic techniques from new medical research to be incorporated into the annotations.

After the annotations for each chunk has been computed, the data chunk and its associated annotations are then persisted in a table within the DBMS together with other important metadata such as timestamps, patient identifiers etc. In our running example on ECG streams, each ECG chunk and its associated annotations are stored in a table as shown in Figure 4. Each row in the table persists data associated with an ECG chunk. The `ECGChunk` column stores the actual chunk of ECG signals either in binary format or in the HL7 XML format for ECG signals. The `Features` column stores the annotations in XML format.

4.4 Querying Data Streams

External applications can access streaming data by formulating queries against the chunks stored in the DBMS. Either XPath or the XQuery language can be used in addition to SQL to make complex queries on the streams. These queries can be formulated using both relational columns and XML data elements. Using ECG data as an example, we illustrate the power of this approach by presenting a set of interesting queries that medical professionals can use to retrieve ECG segments of interest.

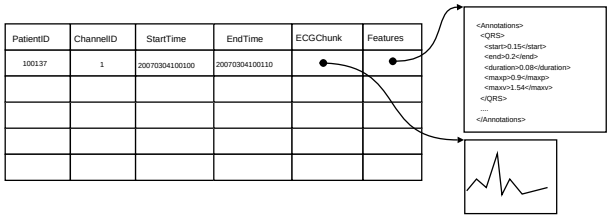

Figure 4: An example of how ECG segments from the ECG streams are persisted in a hybrid relational-XML database.

EXAMPLE 1. *Suppose we would like to find all patients whose ECG has a certain characteristic, say, QRS duration greater than 0.04, between March 1 2007 and March 2 2007. We can write the query as follows.*

```
SELECT distinct PatientID
FROM ECGTABLE
WHERE StartTime > 200703010000 AND EndTime < 200703020000
      AND XMLExists('$t/Annotations/QRS[duration > 0.04]'
      PASSING BY REF E.features as ''t'')
```

In Example 1, the time window is specified on the relational columns `StartTime` and `EndTime`. The constraint on the annotations is specified using the SQL/XML function XMLExists. The semantics of the query is to check each row in the ECGTABLE. If the chunk falls within the time window and if the annotations contains at least one QRS element whose duration is greater than 0.04, then the patientID of that chunk is returned.

EXAMPLE 2. *Suppose we would like to find all patients who have at least N occurrences of the 'QRS duration > 0.04' condition in at least one of their ECG signals between March 1 2007 and March 2 2007. The query can be formulated in SQL/XML as follows.*

```
SELECT E.patientID, E.channelID, SUM( X.CNT )
FROM ECGTABLE E,
     XMLTable( '$t/Annotations' PASSING BY REF E.FEATURES
     AS ''t''
     COLUMNS
     CNT INTEGER PATH 'fn:count(QRS[duration>0.04])' ) X
WHERE StartTime > 200703010000 AND EndTime < 200703020000
GROUP BY E.patientID, E.channelID
HAVING SUM( X.CNT ) > N
```

Example 2 is significantly more complex than Example 1. The intuition for expressing this query is to count the number of occurrences of the 'QRS duration > 0.04' condition for each chunk and then to sum the number of occurrences for each patient for each ECG channel. A final filtering on whether the sum exceeds N will yield the required results. The approach we have taken is to first extract the number of occurrences of the required condition using the XMLTable function and construct a table with the patientID, channelID, and the occurrence count. Next, we group the rows of this "table" by patientID and channelID, and sum all the occurrence counts within each group. The final step is to find the groups of which the sum of the occurrence counts satisfy the required threshold using the `HAVING` clause.

EXAMPLE 3. *Suppose we would like to find all patients whose average QRS duration exceed a certain threshold or whose QRS duration variance exceeds a certain threshold. The query can be expressed as follows.*

```
SELECT E.patientid, E.channelid, AVG( X.DUR ), VARIANCE( X.DUR )
FROM ECGTABLE E,
     XMLTable('$t/Annotations/QRS' PASSING BY REF
     E.FEATURES AS ''t"
     COLUMNS
     DUR DOUBLE PATH 'duration') X
WHERE StartTime > 200703010000 AND EndTime < 200703020000
GROUP BY E.patientid, E.channelid
HAVING AVG( X.DUR ) > 0.05 OR VARIANCE( X.DUR ) > 0.00007
```

Example 3 demonstrates how to perform queries involving statistical operators on the annotations. The first step is to extract via the XMLTable function the relevant annotation data (in this case, the QRS duration) into a relational table. The next step is to group the data into patientID and channelID, so that we can apply the statistical operators AVG and VARIANCE on each group. The final step is to filter the groups based on the statistical computations in the HAVING clause.

As our examples illustrates, very complex analytical queries can be formulated on the data streams stored in the hybrid relational-XML DBMS. Moreover, no addition query processing code need to be developed for answering these complex queries, because we leverage the DBMS query engine to process these queries.

5. PERFORMANCE EVALUATION

The performance of the architecture described above is largely dependent on the ability to cope with large stream event rates at the hybrid relational-XML DBMS. While the combination of the proposed pre-processing subsystem with a hybrid relational-XML DBMS addresses our functional requirements, it remains to be seen whether this approach also meets our performance requirements. Because of the unbalanced read/write ratio at the hybrid relational-XML DBMS, it is clear that the real system bottleneck is the insert rate at which streams can be stored in the database. In this section, we address this question with both a theoretical and an experimental evaluation of this rate.

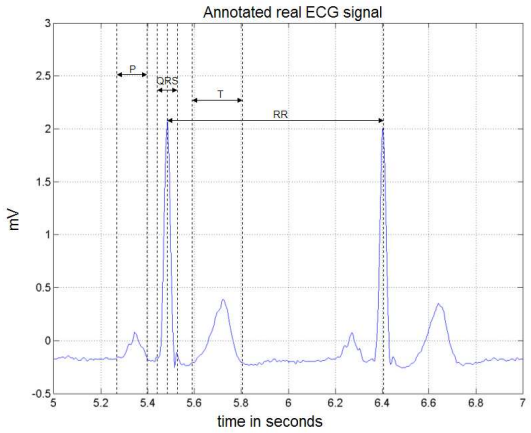

Figure 5: Manual annotation of a real ECG signal

5.1 Experimental Set-up

To illustrate the capabilities of the system, we focused on the persistence of ECG data streams. Figure 5 shows the common annotations that are extracted by most ECG analysis systems. These features define a language that is typically used by medical professionals to characterize ECG signals and make medical assessments.

For our tests, we focused on the detection of the QRS interval and the detection of the precise location for the peak in the R wave. The time coordinates of these peaks are often used to measure accurately patients heart rates.

The detection of the QRS interval is done by applying the well known Pan-Tompkins algorithm [24, 1] to ECG signals. This algorithm is a low complexity technique applying simple digital filters to the data. The following equations summarizes its operations: Let $x[n]$ represent a discrete time signal representing a regularly sampled ECG signal. The first step of the algorithm differentiates $x[n]$ with simple Finite Impulse Response (FIR) filters:

$$x^{(1)}[n] = x[n] - x[n-2] \tag{1}$$

$$x^{(2)}[n] = x[n] - 2x[n-2] + x[n-4] \tag{2}$$

The next steps magnifies and combines the first and second order derivatives of $x[n]$ according to the following equations:

$$y_0^{sm} = \frac{1}{4}[\|x^{(1)}[n]\| + 2\|x^{(1)}[n-1]\| + \|x^{(1)}[n-2]\|] \tag{3}$$

$$y_1^{sm} = \frac{1}{4}[\|x^{(2)}[n]\| + 2\|x^{(2)}[n-1]\| + \|x^{(2)}[n-2]\|] \tag{4}$$

$$y_2[n] = 1.3y_0^{sm}[n] + 1.1y_1^{sm}[n] \tag{5}$$

The algorithm ends with a non-linear step where the output of the previous step is integrated before being compared with a threshold according to the following equation:

$$y_q[n] = 1[(y_2[n] > \gamma) \wedge (\sum_{i=1}^{8} 1[y_2[n+i] > \gamma] > 6)] \tag{6}$$

where $1(.)$ is the indicator function, and $\wedge$ is the logical AND operator. The output $y_q[n]$ of the filter represented by Equation 6 is a mask representing the position of the QRS intervals in the input signal x. Figure 6 shows the output of this algorithm after the processing of a real ECG sample, obtained from the Physionet database [3].

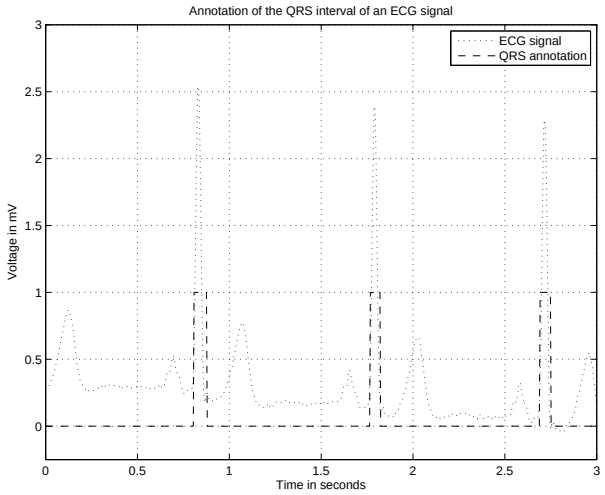

Figure 6: Annotated ECG Signal

5.2 Theoretical Predictions

In order to make predictions on the insert rate of the storage component of the system, we model it with simple queuing models. To build these models, we treat this component as a black box because we have very limited access to its internals. Consequently, our theoretical model is built from end to end characteristics of the system.

We start by modeling the arrival process at the system with a Poisson process. While this decision allows us to leverage well understood mathematical constructs, it is also supported in parts by several key observations on the way streams are generated and sent to the system. As explained in Section 1, remote health monitoring systems are often characterized by a very large number of data hubs at the second tier. These hubs are directing streams of biometric readings towards the storage system, at the server tier. Since there is no coordination among the hubs during the transmission, it is quite reasonable to assume that each hub is operating independently. It is reasonable to assume that patients are not coordinating among themselves the transmission of data to the server tier. Hence, the traffic aggregated across hubs that is received at the server tier is likely to exhibit characteristics of a Poisson distribution [13]. However, this model has its limits. Indeed, Poisson processes are stationary but the traffic directed towards our system may only be piecewise stationary. Typical users might be more likely to send biometric readings during certain periods of the day. For example, users might be more likely to issue weight and blood pressure readings in the morning and before bed time and less likely to do so in the middle of the afternoon. Nevertheless, we stick with the Poisson assumption and attempt to build a simple model that can reasonably approximate the arrival traffic.

To formalize this model, let N be the total number of hubs in the system and λ_i, c_i be respectively the arrival rate from the ith hub in chunks per second and the average chunk size from the ith hub, $1 \leq i \leq N$. Let r_i be the average data rate at which hub i is producing data in samples per second. Finally, Let λ_{total} be the rate of the aggregate process in chunks per second, as seen at the server. We have:

$$\lambda_{total} = \sum_{i=1}^{N} \lambda_i = \sum_{i=1}^{N} r_i/c_i \tag{7}$$

Clearly, if all hubs are transmitting similar chunks with the same average size c, at the same average rate r, $\lambda_{total} = \frac{r}{c}$.

Let μ denote the average number of chunks processed by the system in a second. μ is also called the average service rate. In the most general case, μ is related to the number of chunks queued in the system. To simplify our model, we assume for the rest of this work that μ is not state dependent. Let $\rho = \frac{\lambda_{total}}{\mu}$ denote the utilization of the system. When $\rho \geq 1$, the system is unstable because the arrival rate exceeds the service rate and the DBMS cannot cope with all the requests.

We measure the performance of the system with the average delay encountered by any chunk while being served in the system. This delay contains both a waiting time corresponding to the average queuing delay a chunk waits to be served, and the processing time, equal to the average time it takes to process a chunk. In this setting, the average delay encountered by any chunk entering the system is given by the Pollaczek-Khinchin (P-K) formula [10]. Let $\bar{T}$ represent the average delay experienced by chunks in the system, then the P-K formula states that:

$$\bar{T} = \frac{\frac{1}{\mu}}{(1-\rho)}[1 - \frac{\rho}{2}(1 - \mu^2\sigma_\mu^2)] \tag{8}$$

where σ_μ^2 is the variance of the service time.

Equation 8 models the average delay from measurements of the arrival rate, and the first and second moments of the service rate.

While the arrival rate can be estimated from the number of patients and the rates of the data sources, the statistics of the service rate depend completely on the hybrid relational-XML DBMS. We estimate these statistics from end to end delays experienced when a special test input traffic is used. This input tries to reduce the queue waiting time by sending periodic requests to the system at a rate much lower than the actual service rate. In essence, this input attempts to minimize the probability of having chunks in the waiting queue. From the resulting average end to end delay T_0 and its variance σ_{T_0}, we approximate the variance of the service rate σ_μ using the delta method[2].

Figure 7 plots $\bar{T}$ in milliseconds as a function of the arrival rate λ_{total} for a fixed chunk size. It compares the theoretical prediction with the experimental measurements. It also shows that despite its simplicity, this theoretical model approximates well the behavior of the DBMS. This theoretical prediction for $\bar{T}$ allows us to approximately provision our system without having to run a full set of experiments. Indeed, given a value for the maximum tolerable delays, Equation 8 can be used to determine an upper bound for λ_{total} from which an upper bound on the maximum number of patients that the system can handle can also be derived. The region of the curve after the knee is to be avoided. It corresponds to unstable operating points where $\rho = \frac{\lambda_{total}}{\mu} \geq 1$. In practice, it is recommend to stay as far as possible away from this knee.

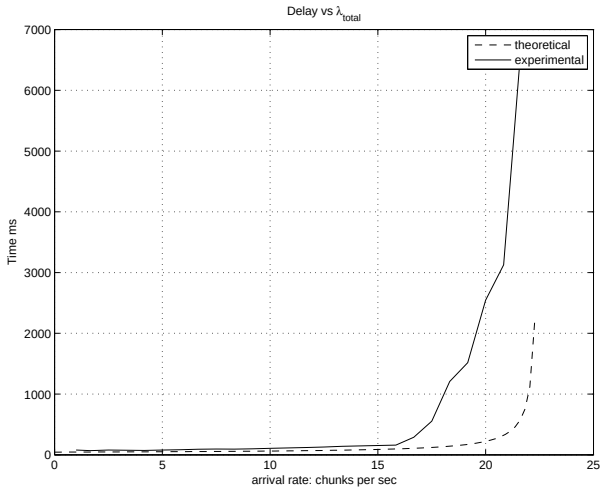

Figure 7: Validation of the theoretical model

[2]The delta method states that $\sigma_{f(x)}^2 \approx (f^{(1)}(m_x))^2\sigma_x^2$, if f is twice differentiable. m_x is simply the mean of x. In our case, since the end to end delay is exactly the inverse of the service rate when there are no jobs waiting in the queue, $f(x) = \frac{1}{x}$ and $\sigma_\mu^2 \approx (\frac{-1}{T_0^2})^2\sigma_{T_0}^2$

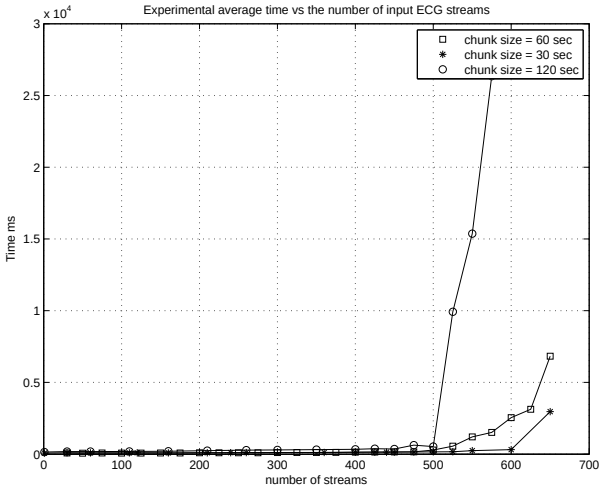

Figure 8: Impact of the number of streams on the average delay of the DBMS

5.3 Experimental Evaluation

We have performed a series of experimental tests to evaluate the performance of the DBMS. An IBM IntelliStation Z Pro 6223-64Y machine was used to host the hybrid XML-relational DBMS. This machine is a dual processor box, each running at 3.6 GHz CPU. It also has 4GB of RAM.

We wrote a simulator in Java, controlled by a set of Perl scripts to simulate ECG streams. This simulator takes two parameters as inputs: the chunk size and the number of streams that it generates. Each stream is served by a separate thread responsible for the insertion of ECG chunks at rates equal to the real event rate of our ECG signals. These threads essentially mimic different hubs pumping ECG data into the system.

Figure 8 shows a summary of the performance that we obtained on this testbed. The X axis shows the number of streams that were directed to the storage subsystem. The Y axis shows the average delay that was observed. There are 3 graphs on Figure 8, each one corresponding to a different chunk size. We observe that the system stays in the stability region in all the graphs, when the number of streams is below 500. After this point, the average delay grows exponentially, as predicted by the M/G/1 model. In addition, we notice that the system is quite sensitive to the chunk size. Moderate chunk size values yield better performance than large chunk size values.

To measure the effect of the chunk size on the performance of the system, we have performed another set of experiments. Figure 9 shows a summary of these tests. To obtain these plots, we have fixed the number of streams and have varied the chunk size to see the effect on the average end to end delay. We have also used Equation 8, to predict the average delay. There are two key observations that can be made from these plots. First, as the chunk size increases, the average delay increases linearly, as seen on both the experimental and theoretical curves. However, for large numbers of streams, as the chunk size increases, the system reaches a point after which, the average delay grows exponentially. Second, our theoretical model predicts the behavior of the system reasonably well when the system is stable. It predicts lower bounds on the end to end delays. The differences between our theoretical predictions and the experimental observations occur when the system becomes unstable. To better model this behavior in the unstability region, we believe that we need a more complex model for the service rate that would take into account the state of the system (i.e., the number of chunks waiting to be served).

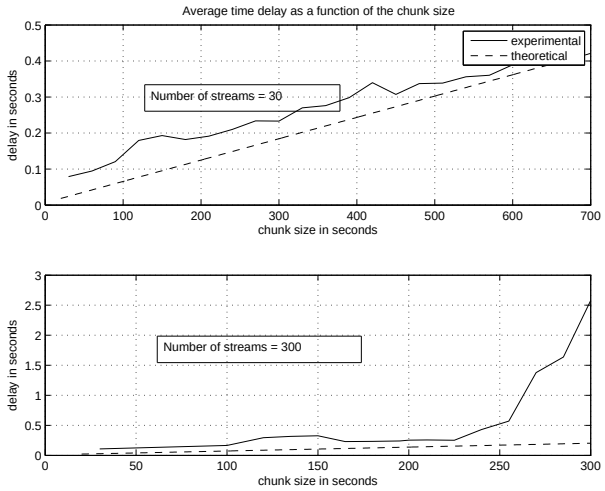

Figure 9: Impact of the chunk size on the average delay of the storage sub-system

6. CONCLUSIONS

We have presented an architecture centered around a hybrid relational-XML DBMS to persist and query biometric streams collected by health monitoring systems. The proposed architecture is extensible. With native support for XML, the system can be extended to support any type of sensor event streams. An open pre-processing framework is also designed to automate the annotation of streams as they enter the system. External applications access stream data by making semantically rich queries to the hybrid relational-XML DBMS.

We have built theoretical models predicting the performance of the storage system and have performed a series of tests to measure the ability of the system to handle large data insertion rates. From these tests, we conclude that this system can sustain large number of continuous data streams (approximately 500 multi-channel ECG streams on our testbed).

In the future, we plan on incorporating more parameters in our theoretical model to better predict the behavior of the DBMS. For instance, parameters representing the structure of XML documents (e.g., average tag to data ratio) could help build better models for the average service rate of the system.

In addition, we plan on incorporating this architecture into the server tier of a complete remote health monitoring system. We also plan on developing real remote health monitoring applications on this platform and test the reaction of the entire system under real workloads.

7. ACKNOWLEDGEMENTS

The authors would like to thank all the Century project members, for their help and support: Marion Blount, John Davis, Maria Ebling, Ji Hyun Kim, KangYoon Lee (project manager), Archan Misra, SeHun Park and Karen Witting. The authors would also like to thank the anonymous reviewers for their valuable comments.

8. REFERENCES

[1] Electrocardiography. http://www.eie.polyu.edu.hk/~ensmall/eie448/pdf/T6.pdf.

[2] Hl seven. http://www.hl7.org.

[3] Physiobank: physiologic signal archives for biomedical research. http://www.physionet.org/physiobank/.

[4] XML path language (XPath) 2.0. http://www.w3.org/TR/xpath20/.

[5] Soaring health care costs. http://www.pbs.org/newshour/bb/health/jan-june06/healthcare_1-10.html.

[6] D. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik. Aurora: A new model and architecture for data stream management. *VLDB Journal*, 12(2), 2003.

[7] D. J. Abadi, Y. Ahmad, M. Balazinska, U. Cetintemel, M. Cherniack, J.-H. Hwang, W. Lindner, A. S. Maskey, A. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. Zdonik. The design of the borealis stream processing engine. In *2005 Conference on Innovative Data System*, 2005.

[8] A. Bar-Or, D. Goddeau, J. Healey, L. Kontothanassis, B. Logan, A. Nelson, and J. V. Thong. Biostream: A system architecture for real-time processing of physiological signals. In *26th Annual International Conference of IEEE Engineering in Medicine and Biology Society*, 2004.

[9] E. Belinsky, M. Ebling, S. Jakobsson, W. Jerome, N. Marmasse, A. Misra, E. Sonsino, V. Soroka, and D. Sow. Pasta: Deriving rich presence for converged telecommunications network applications. In *Comsware*, 2005.

[10] D. Bertsekas and R. Gallager. *Data Networks, 2nd edition*. Prentice Hall, 1992.

[11] M. Blount, V. M. Batra, A. N. Capella, M. R. Ebling, W. F. Jerome, S. M. Martin, M. Nidd, M. R. Niemi, and S. P. Wright. Remote health-care monitoring using personal care connect. *IBM Systems Journal*, 46(1), 2007.

[12] Wrestling with regulations. http://storageMagazine.techtarget.com/magItem/0,291266,sid35_gci955922,00.html.

[13] E. Cinlar. Introduction to stochastic processes. Prentice Hall, 1975.

[14] A. Eisenberg and J. Melton. SQL/XML is making good progress. *SIGMOD Record*, 31(2):101–108, 2002.

[15] A. Eisenberg and J. Melton. Advancements in SQL/XML. *SIGMOD Record*, 33(3):79–86, 2004.

[16] D. Ferrucci and A. Lally. Uima: An architectural approach to unstructured information processing in the corporate research environment. *Natural Language Engineering*, 10(3), 2004.

[17] J. E. Funderburk, S. Malaika, and B. Reinwald. XML programming with SQL/XML and XQuery. *IBM Systems Journal*, 41(4), 2002.

[18] America's healthcare crisis: Understanding its causes and possible solutions.

[19] IBM DB2 Universal Database viper release. http://www.ibm.com/db2/udb/viper.

[20] Chronic care medicine: Physicians say "help!". http://www.hopkinsmedicine.org/Press_releases/2004/05_27a_04.html.

[21] M. Nicola and J. John. Xml parsing: A threat to database performance. In *12 International Conference on Information and Knowledge Managment*, 2003.

[22] M. Nicola and B. van der Linden. Native xml support in db2 universal database. In *VLDB 2005*, pages 1164–1174. VLDB Endowment, 2005.

[23] F. Ozcan, R. Cochrane, H. Pirahesh, J. Kleewein, K. Beyer, V. Josifovski, and C. Zhang. System RX: One part relational, one part XML. 2005.

[24] T. Pan and W. J. Tompkins. A real-time qrs detection algorithm. *IEEE Trans. Biomed. Eng.*, 32(3):230–236, 1985.