Scalability of an Open Source XML Database for Big Data

John Chelsom
City University, London
john.chelsom.1@city.ac.uk

Abstract

Big Data tools and techniques are starting to make significant contributions in clinical research and studies. We explore the use of XML for holding data in an electronic health record, where the primary data storage is an open source XML database of clinical documents. We evaluate the feasibility of using such a data store for Big Data and describe the techniques used to extend to the massive data sets required for meaningful clinical studies.

Using an open source Electronic Health Records system we have loaded the database with a set of patient records and measured the size of the database from 1 to 20,000 patients, together with the execution time of a typical query to retrieve and combine data across a cohort of patients.

We describe the implementation of a federated data store, whereby we can scale to millions of patient records. We then make projections for the size and search execution time at Big Data scale.

Introduction

In healthcare, the application of Big Data tools and techniques is revolutionizing clinical research and has already contributed to breakthroughs in cancer treatments and management of long term conditions such as diabetes and coronary heart disease [1,2].

In this paper we explore the scalability of the eXist open source, native XML database for Big Data. We have used an open source Electronic Health Records (EHR) system to load the database with a set of patient records and measured the size of the database from 1 to 20,000 patients, together with the execution time of a typical query to retrieve and combine data across a cohort of patients. Extrapolating these results, we have estimated the largest single database instance which would produce acceptable query performance.

In order to reach the scale required for Big Data in clinical research, which we have assumed to be the full records for 55 million patients, we have investigated the use of federated search across multiple instances of the native XML database. By combining the results from this federated search we are able to achieve the scalability required for Big Data. Our approach to testing the scalability of the XML database follows the same principles described in the BigBench data benchmark proposal [3].

Electronic Health Records in XML

Health records typically contain a mix of structured and unstructured data, with some data highly structured (laboratory test results, for example), some lightly structured (clinic attendance notes) and some with no structure, other than meta data (diagnostic images). This variation makes XML an ideal candidate for the basic data representation and storage of healthcare data and has led to the development of a number of standard representations, the most commonly used being the Health Level 7 Clinical Document Architecture (HL7 CDA) [4].

cityEHR [5] is an open source EHR which is currently deployed in five hospitals in the National Health Service in England, as well as being used for research and teaching of health informatics. It is built as an XRX application (XForms – REST – XQuery) on existing open source Enterprise Java components, primarily Orbeon Forms [6], the eXist XML database [7] and the Mirth messaging engine [8] running in Apache Tomcat. The study described in this paper used Orbeon version 3.9 and eXist version 2.2. The architecture of cityEHR, shown in Figure 1, was inspired by a previous commercial product called Case Notes, which was implemented using a relational database [9].

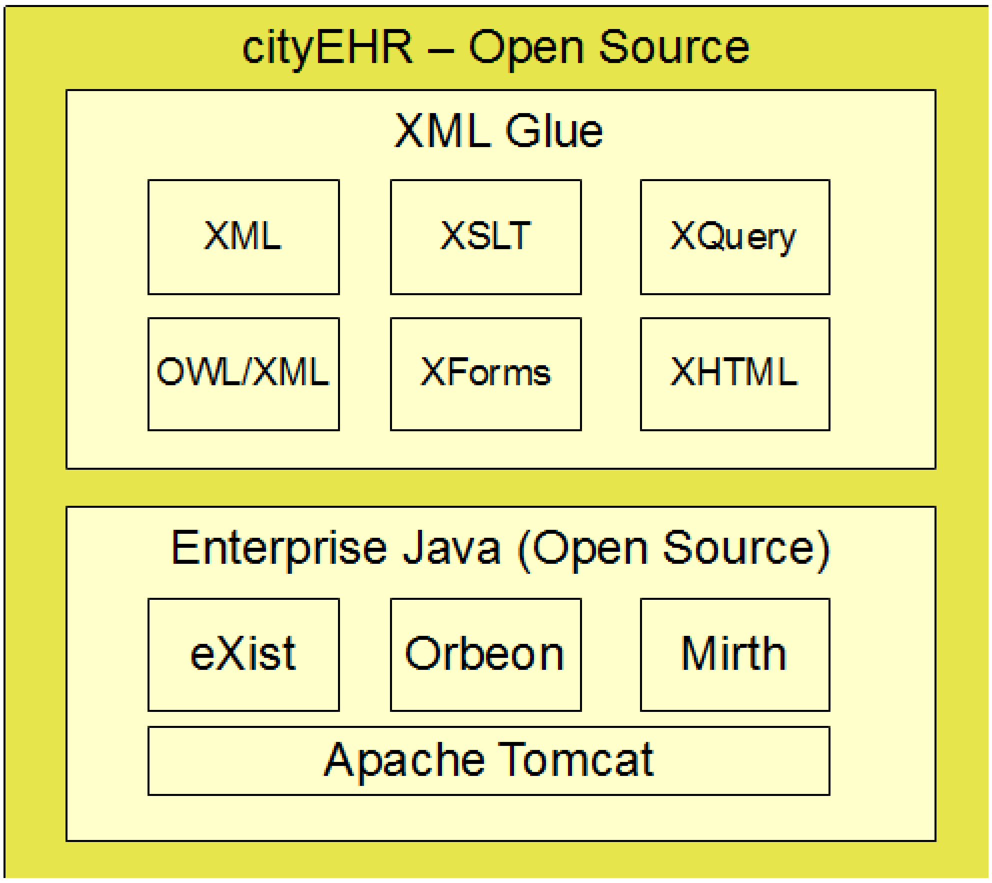

Figure 1. cityEHR as an XRX Application

We have used the eXist XML database for developing XML applications in healthcare since soon after its first release in 2000; a version of eXist also ships with Orbeon as the default persistent store for XML. It was therefore a natural choice to use eXist for cityEHR, although there are several open source alternatives, most notably BaseX [10] and Berkeley DB XML Edition [11].

eXist ships with the Jetty Java Servlet container by default and for this study we have used that default installation running as a service in the MS Windows operating system. For live deployments of cityEHR in hospitals we can also run the eXist database in the same Tomcat instance as Orbeon, which is a simpler environment to deploy and maintain.

Scalability of the XML Database

Database Size

To measure the scalability of the XML database we used the test data generation feature of cityEHR to create databases ranging in size from 1 to 20,000 records. This data generation feature imports a sample patient record and replicates that record in the database, varying key data for each test instance created. These key data include the patient identifiers (which are anonymised), dates (which are offset by a random time period) and gender (the proportion of male/female patients can be set for the generated test data).

For the current study, we used a sample record containing 34 clinical documents, which when stored on disk as a single XML file (HL7 CDA format) was approximately 1.35Mb in size. When imported to the XML database, the database of 20k patients was 31.6Gb, including indexes. The equivalent size of the raw XML for this number of patients is 27Gb, so the database 'bloat' is about 17%, which compares well with other databases. The database size up to 20k patients (680k documents) is shown in Figure 2. Based on these results, our estimate of the size of a database of 100,000 patients is 156Gb, 500,000 patients is 780Gb and 1 million patients is 1.56Tb.

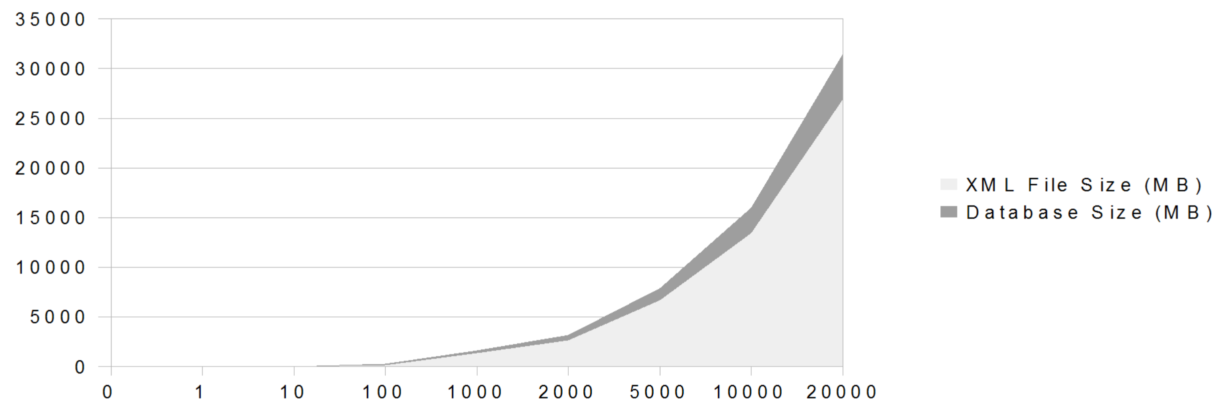

Figure 2. Database Size with Increasing Record Count

The eXist database is indexed with a range index on two XML attributes and a full text (Lucene) index on the same two attributes. These indexes are sufficient to retrieve any patient data from the HL7 CDA records, since all data follow the same pattern in terms of XML markup.

```
<collection xmlns="http://exist-db.org/collection-config/1.0">
  <index>
    <!-- Disable the standard full text index -->
    <fulltext default="none" attributes="no"/>

    <!-- Full text index based on Lucene -->
    <lucene>
      <analyzer class="org.apache.lucene.analysis.standard.StandardAnalyzer"/>
      <text qname="@extension"/>
      <text qname="@value"/>
    </lucene>

    <!-- New range index for eXist 2.2 -->
    <range>
      <create qname="@extension" type="xs:string"/>
      <create qname="@value" type="xs:string"/>
    </range>
  </index>
</collection>
```

XQuery Formulation and Indexing

For scalability testing we used a single test XQuery which finds all Female patients, returning the patient identifier. This query is typical of any query that finds clinical data in a set of HL7 CDA documents and uses both the range and Lucene full text indexes in eXist.

We also ran similar queries with fewer predicates, so that just the range or full text indexes were used and queries with XQuery for, let and return clauses. The purpose of these additional queries was to verify that the performance results reported for the documented test query are representative of a wider range of queries that may be run in cityEHR. That said, all queries on clinical data in HL7 CDA are made on the same extension and value attributes used in the test query.

```
xquery version "1.0";
declare namespace cda="urn:hl7-org:v3";

/descendant::cda:value[ft:query(@value,'Female')]
           [@extension eq '#ISO-13606:Element:Gender']
           [../cda:id/@extension eq '#ISO-13606:Entry:Gender']
           /ancestor::cda:ClinicalDocument/descendant::cda:patientRole/cda:id
```

Query Execution Time

The results of loading the database with up to 20,000 patients and running the test query are shown below. Table 1 shows the database size and number of search hits returned; Figure 3 shows these as a graph.

Patient Records	0	1	10	100	1000	2000	5000	10000	20000
Database Size (MB)	80.3	82.1	97.3	248	1700	3210	7950	16000	31600
Hits		1	4	49	499	999	2499	5025	9775

Table 1. Database size and search hits from 1 to 20,000 patients.

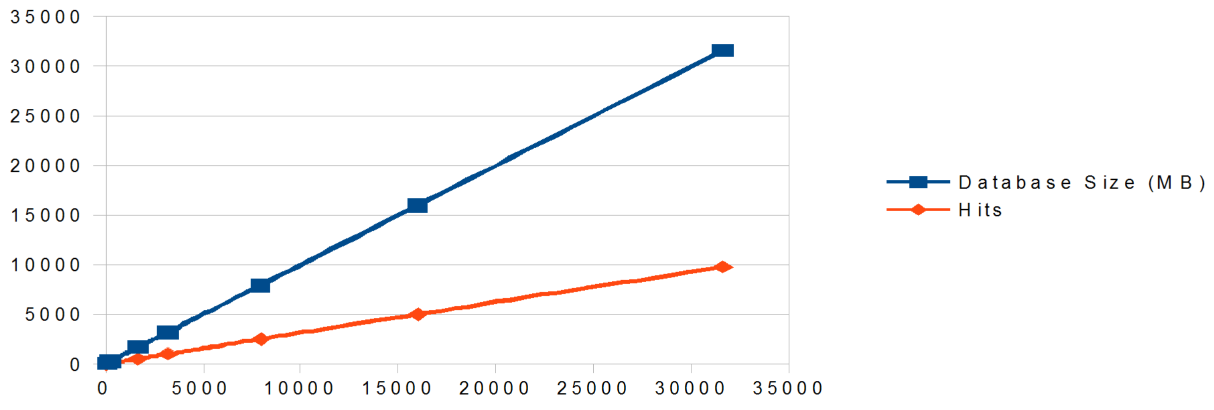

Figure 3. Database size and search hits from 1 to 20,000 patients.

Table 2 shows the execution times for the first and repeated queries; Figure 4 shows these as a graph. These results were obtained on a quad-core Intel i7 processor, with 16Gb RAM running Windows 8. cityEHR was running under Apache Tomcat with 4096Mb Java heap; eXist was running under Jetty with 4096Mb Java heap.

Patient Records	0	1	10	100	1000	2000	5000	10000	20000
First Query Time (sec)		0.61	0.58	0.59	0.98	1.15	1.92	3.02	6.7
Repeat Query Time (sec)		0.016	0.016	0.06	0.18	0.34	0.58	0.98	2.49

Table 2. Execution time for first and repeated queries from 1 to 20,000 patients.

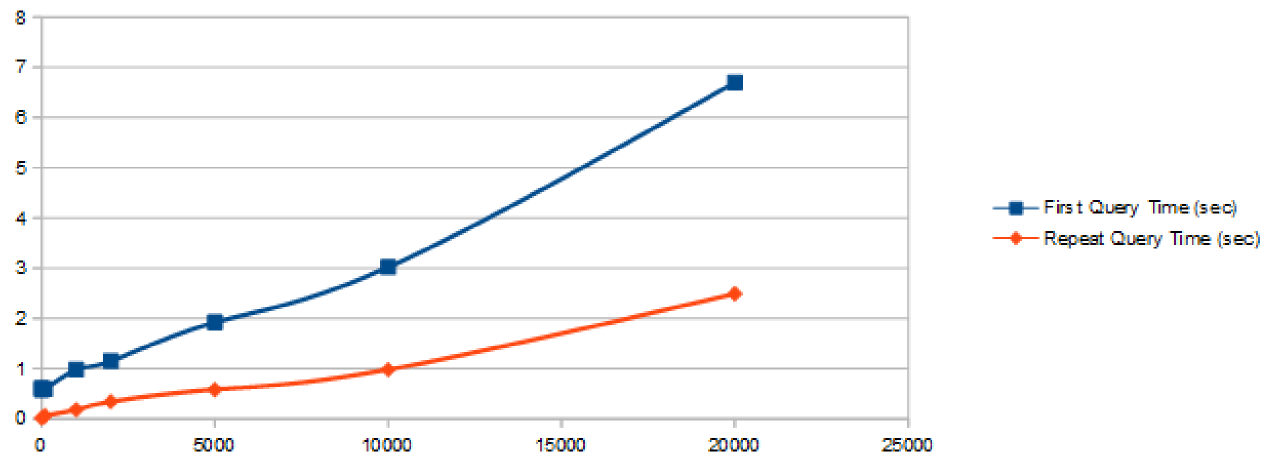

Figure 4. Execution time for first and repeated queries from 1 to 20,000 patients.

When a slightly different search is performed (e.g. to search for Male rather than Female patients) the execution time reverts to the first query time, so the cache used by eXist does not assist in this case. We have therefore based our projections and conclusions on the slower first query time, rather than the cached query, and ensured during this study that services were restarted and the cache cleared before each measurement was taken.

To check the impact of the Java heap size allocated to the eXist process, we ran the same query at heap sizes from 128Mb to 4096Mb (on a 1000 patient database running on a dual core server, with 8Gb RAM). We found that the query did not complete at 128Mb, completed in 2.9 seconds at 256Mb and then completed consistently in around 2 seconds for heap sizes from 512Mb to 4096Mb. Our conclusion is that once the Java heap is above a sufficient threshold it has no effect on query execution time.

Extension to Big Data Scale

Federated Search

To reach Big Data scale requires a step change in the scale of the XML database which is unlikely to be achieved through simple expansion of a single database instance. A large acute hospital in the NHS might cover a population of 1.5 million patients and a national database for England should cover up to 55 million patients.

Fortunately, the fact that each patient record is a discrete data set allows us to scale the database as a set of separate database instances, over which a federated search can be run. Such replication or clustering could be implemented at the database level, using whatever support the database offered for this type of scaling. For example, the eXist database has some support for clustering using the Apache ActiveMQ messaging system to marshal a master and slave databases.

Our objective has been to create a federated database using the components of cityEHR (namely Orbeon and eXist) 'out of the box' with no additional software components and no database-specific implementation. Such an approach ensures that the solution can be deployed easily on any network of servers or virtual machines.

So rather than use database-level replication or clustering facilities, we replicated the same database on five separate network servers. For the purposes of this test we used a 1000 patient database on dual core servers with 4Gb memory, using 1024Mb of Java heap for eXist.
Before testing the federated search, we ran the same test query on each database separately with the results shown in Table 3. Each individual query returned 499 hits. Node 1 was running on the same server as the cityEHR (Tomcat) instance, whereas nodes 2 to 5 were running on network servers.

Network Node	Node 1	Node 2	Node 3	Node 4	Node 5
Query Time (sec)	0.24	0.68	0.58	0.58	0.53

Table 3. Execution time for database nodes of 1000 patients.

Using XML Pipelines

Our first implementation of a federated search process was implemented using an XML Pipeline in Orbeon, which iterates through the available database nodes, makes a submission to each node to run the XQuery and then aggregates the results set. The results for running on the five database nodes are shown in Table 4.

Network Node	Node 1	Node 2	Node 3	Node 4	Node 5
Query Time (sec)	6.2	12.54	18.65	24.25	30.22
Hits	499	998	1497	1996	2495

Table 4. Results of federated search on 1 to 5 database nodes, using an XML pipeline.

Were the submissions from the pipeline asynchronous, we would expect that the time taken to complete the federated query would be equivalent to the slowest query to any database instance plus the time required to aggregate the results from all queries. The results show that not only are the submissions synchronous (i.e. each waits for the previous iteration to complete before it submits) but the overhead of the aggregation in the XML pipeline is unacceptably high. We concluded that this is not a viable approach for implementation of the federated search.

Using Iteration Within XForms

Following the disappointing results using XML pipelines, we made a second implementation using iteration of the XQuery submissions from within XForms. The results of this implementation on five database nodes are shown in Table 5.

Network Node	Node 1	Node 2	Node 3	Node 4	Node 5
Query Time (sec)	0.28	0.49	0.82	0.94	1.15
Hits	499	998	1497	1996	2495

Table 5. Results of federated search on 1 to 5 database nodes, using XForms.

These results are more encouraging, showing that the overhead in the aggregation of results sets within XForms is minimal; indeed the results seem to suggest performance that is better than simple synchronous aggregation, since the combined query time for each node running as a single database instance is 2.61 seconds, whereas the federated search on all five nodes was only 1.15 seconds.

The XForms 1.1 standard does specify asynchronous submissions as the default, but in the version of Orbeon we used for these tests (3.9, community edition) only synchronous submissions are documented as being available. Assuming this to be the case, we would therefore expect a significant improvement in the federated search performance were asynchronous submissions used.

Conclusions

The results presented here show that the eXist database scales adequately as the number of patient records increases from 1 to 20,000. Our study has shown that to achieve satisfactory performance, it is vital to formulate each XQuery to ensure optimal use of the database indexes. This may explain why some other independent analyses have reached different conclusions regarding the scalability of eXist [12].

There is a considerable difference between the execution time for the first call of a query and the times for subsequent calls. This is due to caching in eXist, which would appear to be based on the pages touched by the query, rather than the full indexes. For this particular implementation in cityEHR, it is unlikely that an identical query will be run more than once in a single session and so the benefit of caching will not necessarily be seen. However, it is very likely that similar queries will be repeated in the same session, accessing the same indexes; it would therefore be more beneficial for cityEHR if the indexes themselves were cached. Therefore for our purposes we must look at the performance of the first query execution as our benchmark for performance.

Big Data queries on complex data sets can be expected to take some time to return results. It is not obvious where the threshold of acceptable performance lies for such queries, since results on larger or more complex data sets are more valuable and therefore longer execution times are likely to be more acceptable than times for queries driving a user interface (for example).

In the study described here, we have used the eXist database in its 'out of the box' configuration and have not attempted any database tuning beyond creating indexes, carefully formulating XQueries and setting the Java heap above the minimum required to execute the queries. Performance does seem to degrade between 10,000 and 20,000 patients and this is therefore an area for future investigation which may require more sophisticated tuning or knowledge of eXist to address. Assuming we can avoid further degradation, then we would project that a search on a 100,000 patient would complete in under 35 seconds. Limitation of time and resources have so far prevented us from implementing a database of this size, which is an obvious next step for our research.

To implement a federated search using databases of 100,000 patients (100k nodes) would require 10 nodes for one million patients and 550 nodes for 55 million. Using the benchmark of 35 seconds for a 100,000 patient database node and the results of our tests on federated search, we project a search time of just under 6 minutes for 1 million patients, one hour for 10 million and about 5.5 hours for 55 million. Total database sizes (aggregate of the federated databases) for 1, 10 and 55 million would be approximately 1.5Tb, 15Tb and 85Tb.

In conclusion, we can say that although the projected search times on an 85Tb database of 55 million records seem slow, they are based on a fairly unsophisticated approach to implementation. A comparable implementation of the Secondary Uses Service (SUS) [13], to hold data on 55 million patients in the NHS in England, was originally designed to use clustered relational database technology and hardware accelerators to hold far less structured data than we propose in this paper. Hence we conclude that the results obtained so far show enough promise to justify an extension to the next order of magnitude and we will report those results at a future date.

References

[1] Wang, W. and Krishnan, E., 2014. Big data and clinicians: a review on the state of the science. *JMIR medical informatics*, *2*(1).

[2] Bizer, C., Boncz, P., Brodie, M.L. and Erling, O., 2012. The meaningful use of big data: four perspectives--four challenges. *ACM SIGMOD Record*, *40*(4), pp.56-60.

[3] Ghazal, A., Rabl, T., Hu, M., Raab, F., Poess, M., Crolotte, A. and Jacobsen, H.A., 2013, June. BigBench: towards an industry standard benchmark for big data analytics. In *Proceedings of the 2013 ACM SIGMOD international conference on Management of data* (pp. 1197-1208). ACM.

[4] Dolin RH, Alschuler L, Boyer S et al., 2006. "HL7 Clinical Document Architecture, Release 2." J Am Med Inform Assoc. 2006;13(1):30-9.

[5] Chelsom, J. J, Pande I, Summers R, Gaywood I, 2011. Ontology-driven development of a clinical research information system. *24th International Symposium on Computer-Based Medical Systems*, Bristol. June 27-June 30 ISBN: 978-1-4577-1189-3

[6] Meier, W., 2002. eXist: An open source native XML database. In *Web, Web-Services, and Database Systems* (pp. 169-183). Springer Berlin Heidelberg.

[7] Bruchez, E., 2005. Orbeon, Inc.[Online]. XTech 2005: Are Server-Side Implementations the Future of XForms?

[8] Bortis, G., 2008, May. Experiences with Mirth: an open source health care integration engine. In *Proceedings of the 30th international conference on Software engineering* (pp. 649-652). ACM.

[9] Dave Nurse, John Chelsom, 2000. XML data warehousing for browser-based electronic health records. *Proceedings of XML Europe, 2000.* IDE Alliance.

[10] Grün, C., Holupirek, A. and Scholl, M.H., 2007. Visually exploring and querying XML with BaseX.

[11] Paul Ford, 2003. Berkeley DB XML: An Embedded XML Database. Article available at: http://www.xml.com/pub/a/2003/05/07/bdb.html

[12] Robert Elwell, 2014. eXist-db: Not Ready for High Scale. Blog post at: http://robertelwell.info/blog/exist-db-not-ready-for-high-scale

[13] Powell, J. and Buchan, I., 2005. Electronic health records should support clinical research. Journal of Medical Internet Research, 7(1), p.e4.