Published in final edited form as:

Distrib Parallel Databases. 2019 June ; 37(2): 235–250. doi:10.1007/s10619-018-7230-8.

Scalable and flexible management of medical image big data

Dejun Teng[1], **Jun Kong**[2], **Fusheng Wang**[3]

[1]Department of Computer Science and Engineering, The Ohio State University, Columbus, OH, USA

[2]Department of Biomedical Informatics, School of Medicine, Emory University, Atlanta, GA, USA

[3]Department of Biomedical Informatics and Department of Computer Science, Stony Brook University, Stony Brook, NY, USA

Abstract

Digital imaging plays a critical role for image guided diagnosis and clinical trials, and the amount of image data is fast growing. There are two major requirements for image data management: scalability for massive scales and support of comprehensive queries. Traditional Picture Archiving and Communication Systems (PACS for short) are based on relational data management systems and suffer from limited scalability and query support. Therefore, new systems that support fast, scalable and comprehensive queries on image data are highly demanded. In this paper, we introduce two alternative approaches: DCMRL/XMLStore (RL/XML for short)—a parallel, hybrid relational and XML data management approach, and DCMDocStore (DOC for short)—a NoSQL document store approach. DCMRL/XMLStore manages DICOM images as binary large objects and metadata as relational tables and XML documents based on IBM DB2, which is parallelized through data partitioning. DCMDocStore manages DICOM metadata as JSON objects, and DICOM images as encoded attachments in MongoDB running on multiple nodes. We have delivered two open source systems DCMRL/XMLStore and DCMDocStore. Both systems support scalable data management and comprehensive queries. We also evaluated them with nearly one million DICOM images from National Biomedical Imaging Archive. The results show that, DCMDocStore demonstrates high data loading speed, high scalability and fault tolerance. DCMRL/XMLStore provides efficient queries, but comes with slower data loading. Traditional PACS systems have inherent limitations on flexible queries and scalability for massive amount of images.

Keywords

Medical images; Picture Archiving and Communication Systems; Database management systems; NoSQL; Extensive markup language

Dejun Teng, tengdj@gmail.com.

1 Introduction

Digital imaging is playing a critical role for image guided diagnosis and clinical trials. Digital Imaging and Communications (DICOM) is a standard for representing and managing medical images, and the standard has recently been extended to support whole slide images [1]. DICOM images are represented as a standard format with a header representing all the data elements, and a body consisting of the pixel images represented in standard image formats, such as JPEG, PNG, or GIF. DICOM has defined more than three thousand standard data elements, along with dynamic proprietary tags supported by different vendors. The tags can not only be used to support query and retrieval operations for PACS services, they are also valuable for other purposes such as semantic data integration [2], quality control [3] and medical physics [4–8]. While the large space of data elements and the extension provides high flexibility, it also makes it very challenging to represent, manage and query such information.

There are several requirements for managing medical images: (i) retrieval of images based on identifiers, such as patient IDs, study UIDs, series UIDs or image UIDs; (ii) flexible metadata based queries based on DICOM data elements; (iii) fast response of queries for real-time access of images; and (v) standard communication protocols. Traditionally, the management of DICOM based medical images is based on PACS. PACS are built on top of DICOM information model which is originally defined through an entity-relational model (ER model) [9]. Naturally, most PACS use relational data management systems (RDBMS) as for managing DICOM images [10] where the ER model is easily mapped to relational tables. Typically, such approach provides a database schema with a subset of common DICOM tags [11, 12].

Meanwhile, the amount of medical image data is fast growing with the continuing movement of electronic medical records and integration of massive medical images for decision support. With the emergence of Health Information Exchange and Accountable Care Organizations, the amounts of image data to be integrated and managed become enormous. There is a major demand to provide highly scalable data management and sharing infrastructure to manage, query, and share the data across the network.

The intrinsic characteristics of DICOM images and the increasing scales pose two major challenges for data management: scalability for massive scales and support of comprehensive queries. However, traditional RDBMS based approaches suffer from two major limitations: limited scalability due to intrinsic architecture constraints and inflexibility to support diverse or dynamic schemas. Relational databases use rigid table based schema with predefined fixed set of attributes, and it is very difficult to extend the schema to adapt to new tags. In reality, most schemas for PACS represent only a small fixed subset of DICOM header tags [10, 12]. Meanwhile, due to the rigid ACID transaction model, studies [13, 14] have shown that relational databases suffer from data loading bottleneck and have scalability limit in a distributed environment. Even though some relational database systems like DB2 support variety tag queries by enabling XML query, the poor query performance which will be shown in our work limits its usability.

Recent work on scalable medical image data management includes cloud based approach [15, 16] where large scale computing infrastructures are used to host PACS services, or NoSQL based approach where new emerging data management systems such as CoachDB [17, 18] and Cassandra [19, 20] are employed for managing DICOM images.

In our work, we propose two solutions of high performance scalable data management of medical image big data with powerful query support: DCMRL/XMLStore using traditional relational data management based approach and DCMDocStore using NoSQL based approach, respectively. DCMRL/XMLStore models DICOM data with hybrid relational and XML model supported through a native XML database engine integrated in relational database engine. It is implemented with a data partitioning based approach running multiple database instances in a distributed environment for scalable data management. DCMDocStore models DICOM data as documents represented in JSON objects [21]. We provide comparative studies of the two systems with nearly one million DICOM images downloaded from NBIA [12], and conclude that both systems meet the requirements of query flexibility and scalability, and can be adopted for different environments.

2 System design

2.1 Hybrid relational/XML based management

2.1.1 XML based DICOM modeling—DICOM headers contain structured metadata represented as data element name and value pairs, and the Value Representation (VR) defines the data type and format of the values. The Value Multiplicity (VM) of a data element specifies the number of values that can be encoded in the value field of that Data Element. A data dictionary defines the tags and the identifiers, and extension for proprietary tags is possible.

XML is not only a markup language, but also a hierarchical data model for representing information. A major advantage of XML data modeling is extensibility—an XML schema can be easily extended to accommodate new data elements. In particular, DICOM tags can be well represented in XML. Figure 1 shows an example of XML based representation of DICOM tags. A new tag can be easily added as a new XML element in the document. The XML representation also supports multi-value tags, by repeating the value element inside a tag element, for example, “ImageType” is a multi-valued tag and three child value elements are provided. XML based DIOM format has also been proposed in [22] for improving the readability of DICOM files, but it depends on a small set of tags and does not provide the extensibility. In fact, the flexibility of XML based modeling can be supported by “schema-less” native XML database.

2.1.2 XML based data management—XML data management systems provide significant advantages as they support standard data definition languages based on XML standards such as XML Schema, and standard XML query languages such as XPath [23] and XQuery [24]. Furthermore, the XML-in and XML-out approach greatly simplifies the translation of data models and query languages. XML database technology becomes mature, and XML database products are proliferating [25, 26]. XML database systems are developed through extension of traditional relational databases or new database systems, where an

XML document is the logic data object for manipulation. For example, major relational database engines such as Oracle and DB2 extend their systems with a new native data type *XML* or *XMLType* with specialized storage and indexing support. Native XML databases provide high flexibility on the schema of the XML documents. XML schemas can evolve which allows high extensibility. For example, if a new DICOM tag is added, it can be simply added as a new XML element, and there is no change needed for the database.

Meanwhile, many DICOM data elements are well-known and fixed. Indeed, most DICOM image management systems use DICOM ER model [9] based on a subset of fixed data elements to build the database [10, 12].

In DCMRL/XMLStore, we take a hybrid approach for modeling and managing DICOM images in XML. We combine both relational model and XML model (Fig. 2). Relational tables built on DICOM ER model will be used for managing a subset of frequently used tags to enable fast retrieval based on IDs, including PATIENT, STUDY, GENERAL_SERIES and GENERAL_IMAGE. XML (attribute FULL_HEADER in table GENERAL_IMAGE) is used to model the full set of DICOM header tags, which could be dynamic. The XML data type is embedded in relational tables as a special column, which is stored, indexed and queried separately, but logically integrated into a unified data model to support comprehensive queries. Original DICOM images are stored as binary large objects (BLOBs) and linked to the tables.

2.1.3 Querying DICOM images on RL/XML—

The coexistence of the data rows for the relational model and the XML documents increases the flexibility of queries conducted on the fields that are predefined in the schema. For instance, below is a SQL query to return all image instance UIDs for a given patient name.

```
SELECT i.sop_instance_uid
FROM ad.general_image i, general_series se, study s, patient p
WHERE p.patient_name=‘1.3.6.1.4.1.9328.50.2.0075’
AND p.patient_pk_id=s.patient_pk_id
AND s.study_pk_id=se.study_pk_id
AND se.general_series_pk_id = i.general_series_pk_id;
```

The query can also be expressed in XQuery as follows, with only XML document:

```
for $dicom in fn:distinct-values(db2-fn:xmlcolumn
(‘GENERAL_IMAGE.FULL_HEADER’)/dicom)
where $dicom/PatientName/value/text() = ‘1.3.6.1.4.1.9328.50.2.0075’
return $dicom/SOPInstanceUID/value;
```

One the other hand, a major advantage of the XQuery is that it can query upon all tags, compared to SQL queries relying only a small subset of tags which are predefined in the database schema. For example, we may have a query to find all patients whose injected contrast volume, which is not defined in the relational schema, is between 100 and 150 ml:

```
for $dicom in fn:distinct-values(db2-fn:xmlcolumn
('GENERAL_IMAGE.FULL_HEADER')/dicom)
where $dicom/ContrastBolusVolume/value > 100 AND $dicom/ContrastBolusVolume/
value <= 150
return $dicom/PatientName/value;
```

The performance of XQuery is proved to be slower than SQL for most cases. Therefore, In RL/XML, data requests which can be served by the relational model will be dispatched to the relational database engine and the rest requests will be dispatched to the XML query engine.

2.1.4 Scalable data management with parallel database architecture—One critical challenge for imaging big data is scalable data management. There are two major architecture principles for scalability: scaling up (or vertical scaling) and scaling out (or horizontal scaling). Scaling up is based on increasing the power of computer servers, for example, adding new expensive big servers with more computing power or storage. This approach requires higher level of skills and is not reliable in some cases. Besides, such systems such as IBM mainframe machines are also very expensive. Scaling out, on the other hand, is through data sharding or data partitioning, by dividing the database across many (inexpensive) machines. As a major bottleneck in managing and querying large data is the I/O bottleneck, by partitioning a database running on many independent machines, where each of them has its own storage, such system could significantly improve I/O throughput. Such architecture is also called a shared-nothing architecture [27].

Figure 3 illustrates the RL/XML architecture in which a parallel database runs on separate physical nodes. Each database node has multiple partitions, for example, partition 1–15 on node 1, and partition 16–30 on node 2. Each partition has its own disks, CPU and memory, and the partitions are connected through a fast switched network. Each database instance runs on one partition, for example, there are 30 database instances running together in Fig. 3. One partition is assigned as a master partition. The master partition accepts queries from users, translates and parallelizes queries across all partitions, and aggregates the results—this enables the simplicity and expressiveness of SQL queries as such distributed parallel query execution is transparent to users. Our implementation employs IBM DB2's parallel query execution support [28]. This support provides a single logical view of partitioned data so that clients can compose SQL queries as if they interacted with a serial database with no data partitioning.

While partitioning is used for balancing data across database partitions, for example, splitting GENERAL_IMAGE table based on image SOP_INSTANCE_UIDs, replication is another technique for replicating data in multiple database partitions to reduce data movement between nodes, normally for small tables, such as PATI ENT table.

2.2 Document store based DICOM data management

Recently, “NoSQL” databases [18, 29–31] emerge as a set of open-source databases which mostly are not using SQL. Such databases share common characteristics: they are non-relational, schema-free, and horizontally scalable with easy replication. They come with simple APIs for queries, and they are not following traditional “ACID” transaction model [32]. To achieve scalability, two orthogonal techniques are used for data distribution: replication and sharding (or partitioning). Replication takes the same data and copies it over multiple nodes, and sharding puts different data on different nodes.

2.2.1 Modeling DICOM images with a JSON document model—NoSQL database takes an aggregated data model by grouping multiple data elements into a single document represented in JSON object [21]. JSON (or JavaScript Object Notation) is an open-standard format for representing and transmitting data objects consisting of attribute-value pairs, as an alternative to XML. JSON can also handle hierarchical data structures as XML, but takes a lightweight approach. Similarly to XML based data modeling, for NoSQL database, we take JSON for representing DICOM header metadata, as shown in Fig. 4. Such document based model allows for new attributes to be created without the need to define them or to change the existing documents. As a primary motivation for NoSQL is the ability to run databases on a large cluster, such aggregate oriented data model fits well with scaling out because the aggregated document is a natural unit for data distribution. For DICOM image pixels, we store them as GridFS objects, which manages files as chunks (in the size of 255 KB) and their associated metadata.

2.2.2 MongoDB and its parallel architecture—MongoDB is one type of NoSQL databases with a document oriented store which takes a JSON document based data model. MongoDB extensively uses sharding for partitioning data across multiple shards. For example, Fig. 5 shows a setup for MongoDB database on a two node cluster with 30 shards, with 15 shards on each node. Meanwhile, MongoDB supports replication for fault tolerance. A dataset can be replicated across multiple shards with multiple copies, and in case of failure of one copy, the other copies are still available for access. The replication is different from the shared-nothing database architecture, where only a small fraction of data is replicated for query efficiency instead of fault tolerance. Setup of MongoDB sharding and replication in a cluster is straightforward, and the replication also makes the system robust instead of fault tolerance. However, MongoDB takes a relaxed transaction model and does not follow a rigid transaction model as traditional relational databases.

The ACID property of traditional relational databases guarantees consistency of a database transaction. For example, if there is an update to a record replicated across two nodes, the update is committed only if the update operation has received confirmation from both nodes. Intermediate transaction status is preserved in a log file which may be rolled back if the transaction fails. Such model achieves high consistency but comes with a cost of bookkeeping and long wait for synchronization.

In a distributed computer system, CAP theorem [33] states that it is impossible to simultaneously provide all three of the following guarantees: Consistency (all nodes see the same data at the same time), Availability (a guarantee that every request receives a response about whether it was successful or failed), and Partition tolerance (the system continues to operate despite arbitrary message loss or failure of part of the system). Traditional relational databases preserve the CA properties, and MongoDB preserves the CP properties. Thus, MongoDB can survive failed nodes in a cluster gracefully, but not a relational database. MongoDB takes an "eventual consistency" approach, and it has a window during which an observer might not see consistent updated values in multiple replicas. The system does guarantees that the observer will be able to see consistent values after this window.

2.2.3 Querying DICOM images in MongoDB—

To make it consistent, MongoDB provides a JSON based query language, which is intuitive and expressive. For example, to return all image instance UIDs for a given patient name, the query looks as follows (Here "dicom" represents the collection name):

```
db.dicom.find({"PatientName":"1.3.6.1.4.1.9328.50.2.0075"});
```

Due to its flexibility on schema, we can also easily specify queries for one specific tag, for example, to find all patients whose injected contrast volumes were between 100 and 150 ml:

```
db.dicom.find({ContrastBolusVolume:{$lte:150,$gt:100}});
```

3 Implementation

RL/XML is built with a hybrid relational and XML database engine, scaled through a shared-nothing parallel database architecture. RL/XML is implemented on top of IBM DB2 database [28] with data partitioning feature (DPF). The software includes SQL scripts for creating the tables and indexes for the database, a DICOM header to XML converter based on **dcm4che** [10], a data loading tool for loading DICOM images and extracted header data into the database, and querying APIs for specifying queries. A WADO [34] based implementation of the query APIs will be planned for future release. A DICOM dictionary table is also provided to quickly map DICOM tag IDs to tag names. Single value or multi-value attributes are automatically detected at parsing time. RL/XML can be setup on a single machine or a cluster with multiple nodes. The setup of the parallel database server on multiple nodes requires sharing of the installed DB2 binaries through network sharing, and each node comes with its own running instance. Database partitions can be created for each node and grouped into a partition group to be used by the database. Once the installation is done, a database can be created, and the tables can be created by the table creation SQL scripts. As the DICOM image table (GENERAL_IMAGE) dominates the storage, this table is created with a partition key on the table key GENERAL_IMAGE_PK_ID. When data gets loaded, DICOM images are hashed into different database partitions based on an internal hashing algorithm.

DOC is built on top of MongoDB, using JSON document model for representing DICOM headers, and GridFS for representing binary images. The software includes DICOM header to JSON conversion tool developed using **dcm4che**, DICOM data loading tool and querying tool, and a DICOM tag ID to tag name mapping table. Different from RL/XML, no schema creation is needed. To accelerate queries for common attributes and UIDs, scripts for creating indexes are also provided. MongoDB can be setup with multiple instances or shards, and replication can be setup by defining replica set to provide redundancy and increase data availability.

Both systems are implemented in Java and available for download at github repository [35, 36].

4 Evaluation

4.1 Experiment setup

We have performed experiments to evaluate and compare the effectiveness of the two systems. We use a two-node cluster, with each node equipped with two AMD Opteron 6128 Eight-Core CPUs (16 cores in total), 128 GB RAM, and 8 TB RAID5 storage, and installed with CentOS 5.10. RL/XML uses IBM DB2 Infosphere data warehouse edition 9.7.5, which comes with native XML support and database partitioning support. There are 30 database partitions created for DB2 to run on the two nodes. DOC uses MongoDB 2.6, and 30 MongoDB instances on the two nodes, 15 instances per node. Three Config Servers are used for reliable storage of metadata.

We use a dataset with one million DICOM images from 9298 patients downloaded from NCIA [37], and the total size of the images is 523 GB.

4.2 Performance studies

In this section, we study the performance of data loading, querying and scalability of RL/XML and DOC.

4.2.1 Performance of data loading—The tag-value pairs parsed out from DICOM images are translated into human-readable key words, and then be uploaded into the parallelized databases together with the original files by multiple threads. During this process, the number of loading threads and the number of partitions may influence the data loading performance.

Firstly, we compare the impact of exploiting multiple threads on the loading performance. For fairness, the partition numbers of both RL/XML and DOC are set to 20. Start from 1, we increase the upload threads number linearly until the uploading throughput stops increasing or starts to drop. As shown in Fig. 6a, the loading process can be accelerated with multiple threads for RL/XML. However, the increase ratio decreases along with the increasing of the threads number. The throughput stops increasing when 13 and more threads are issued. All tests are on a machine with 16 CPU cores, thus it is comfortable to increase the threads number to as many as 16 before computation resource competition between loading threads may occur. Therefore, the loading speed reaches the acceptance limitation of RL/XML when the number of loading threads increased to 13. On the other hand, no performance improvement is gained while the number of loading threads increases for DOC. That is because for document based database like MongoDB, no complicate arithmetic calculations are needed to fit the source information into restricted schema and inserted into multiple tables, instead, only log operations are needed to append the documents into the database. Thus, the loading process is highly I/O bounded. Overall, the loading speed of DOC can be up to 3X faster than that of RL/XML.

Secondly, we run tests to evaluate how the number of partitions influences the loading performance. In this set of tests, the number of partitions for RL/XML and DOC varies from 1, 10, 20 and 30. The number of threads used to loading images for RL/XML and DOC is set to 13 and 2 respectively, which are all big enough to feed the two systems as much DICOM images as they can digest. Figure 6b shows the number of images loaded per second as partition number increases. It shows that using more partitions can slightly increase the loading performance for XML/RL, but cannot improve the loading performance for DOC. For DB2, the base that XML/RL is built on, the insertion requests are assigned to different partitions to be served in parallel. However, for MongoDB, the base that DOC is built on, all newly inserted records will be dispatched to one single node called primary node, and then data in the primary node will be spitted and drained to other nodes once the data size of the primary node reaches a threshold. Therefore, using more partitions cannot bring performance enhancement as DICOM images are newly loaded into the DOC. User can improve the loading performance by assigning each node a specific key range. All images belong to a certain key range will be sent to the specific node instead of the primary node. However, this optimization needs priori knowledge, which is not applicable to all cases.

4.2.2 Query performance of systems—Queries on DICOM images can come in different forms. In this DICOM image management case, we classify them into following categories, as shown in Table 1: (1) Key-Value based lookups with low selectivity. (2) Key-Value based lookups with high selectivity. (3) Queries with aggregation operations. (4) Queries on fields which are not defined in the relational model. (5) Queries on fields which contain multiple values.

We conduct all five queries on both RL/XML and DOC. Both systems are equipped with 30 partitions, and indices are issued for all columns\fields referenced by any queries. For RL/XML, queries 1, 2 and 3 are served by the RDBMS, and queries 4 and 5 are served by XML database engine due to the inability of the RDBMS on querying fields which are not predefined.

As shown in Fig. 7, RL/XML system achieves a better performance on queries 2 and 3, which are all served by the RDBMS. Surprisingly, DOC achieves a better performance on query 1 than the RDBMS which is mature and highly optimized. Furthermore, for queries 4 and 5 which are not supported by relational database model and have to be served by XML query engine, the performance of DOC is much better than the RL/XML system. For query 5 specifically, DOC is 100 folds faster than RL/XML. Different from RL/XML, the data on DOC are stored as independent records. As a result, DOC is suitable for point lookups but is not as efficient as the RL/XML in aggregation queries.

4.2.3 Scalabilities—To evaluate the scalabilities of RL/XML and DOC on DICOM image management, we further run query 2 with variant partition numbers. Due to the high selectivity of query 2, most of the records or rows will be retrieved even the indices are issued. As shown in Fig. 8, for queries which can be divided and conquered using multiple nodes, like query 1, increasing the number of nodes can significantly improve the query performance for both systems. However, the latency of test on RL/XML with 30 nodes is even slower than the test with 20 nodes. That is because when the number of partitions is large enough, the overhead of transporting and merging the final results becomes the bottleneck.

4.3 Discussion

DCMRL/XMLStore provides comprehensive query support with scarification of performance. By using multiple partitions, both the loading and query performance can be significantly improved. Traditional RDBMS based approach will not be able to support all queries, but it is most efficient on the performance of queries it is capable to support. DCMRL/XMLStore provides strict transaction support for high reliabil ity. Partitioned databases with robust XML support are only available for commercial DBMS vendors, and come with high licensing fee.

DCMDocStore based on MongoDB also provides comprehensive query support. It has best loading performance, which cannot be further improved by increasing the number of partitions. Scalability of queries for DCMDocStore is good for small number of partitions, but limited for large number of partitions. MongoDB is open source and free.

5 Conclusion

We have developed two open source data management systems to support the modeling, managing, loading and querying large scale DICOM image data. We have tested the systems with nearly one million DICOM images and the experiment results demonstrate support of comprehensive queries and high performance of queries. In particular, DCMDOCStore has higher data loading performance.

Traditional RDBMS based PACS systems have inherent limitations on supporting flexible queries and scalable queries. Parallel XML DBMS based approach inherits the strict transaction model and is suitable for transaction-oriented image applications. NoSQL document store based approach is cost-effective, highly scalable, and provides superior data loading performance, with scarification of real-time data consistency.

Acknowledgements

Funding was provided by National Science Foundation (Grant nos. ACI 1443054 and IIS 1350885) and National Institutes of Health (Grant no. K25CA181503).

References

1. N. E. M. Association. DICOM Supplement 145: Whole Slide Microscopic Image IOD and SOP Classes (2018). ftp://medical.nema.org/medical/dicom/final/sup145_ft.pdf

2. Kahn CE Jr., Langlotz CP, Channin DS, Rubin DL: Informatics in radiology: an information model of the DICOM standard. Radiographics 31(1), 295–304 (2011) [PubMed: 20980665]
3. Kim KJ, Kim B, Lee H, Choi H, Jeon J-J, Ahn J-H, Lee KH: Predicting the fidelity of JPEG2000 compressed CT images using DICOM header information. Med. Phys 38(12), 6449–6457 (2011) [PubMed: 22149828]
4. Källman H-E, Halsius E, Folkesson M, Larsson Y, Stenström M, Båth M: Automated detection of changes in patient exposure in digital projection radiography using exposure index from DICOM header metadata. Acta Oncol. 50(6), 960–965 (2011) [PubMed: 21767197]
5. Jahnen A, Kohler S, Hermen J, Tack D, Back C: Automatic computed tomography patient dose calculation using DICOM header metadata. Radiat. Protect. Dosim 147(1–2), 317–320 (2011)
6. Dave JK, Gingold EL: Extraction of CT dose information from DICOM metadata: automated matlab-based approach. Am. J. Roentgenol 200(1), 142–145 (2013) [PubMed: 23255754]
7. Källman H-E, Halsius E, Olsson M, Stenström M: DICOM Metadata repository for technical information in digital medical images. Acta Oncol. 48(2), 285–288 (2009) [PubMed: 18720055]
8. Rampado O, Garelli E, Zatteri R, Escoffier U, De Lucchi R, Ropolo R: Patient dose evaluation by means of DICOM images for a direct radiography system. Radiol. Med 113(8), 1219–1228 (2008) [PubMed: 18956145]
9. N. E. M. Association: Digital Imaging and Communications in Medicine (DICOM) Part 3: Information Object Definitions (2018). ftp://medical.nema.org/medical/dicom/2013/output/pdf/part03.pdf
10. Open Source Clinical Image and Object Management (2018). http://www.dcm4che.org/
11. Huang HK: PACS and Imaging Informatics: Basic Principles and Applications, 2nd edn. Wiley-Blackwell, Hoboken (2010)
12. National Biomedical Imaging Archive (2018). https://imaging.nci.nih.gov/ncia/login.jsf
13. Aji A, Wang F, Vo H, Lee R, Liu Q, Zhang X, Saltz J: Hadoop-GIS: a high performance spatial data warehousing system over mapreduce. Proc VLDB Endow 6(11), 1009–1020 (2013)
14. Pavlo A, Paulson E, Rasin A, Abadi DJ, DeWitt DJ, Madden S, Stonebraker M: A comparison of approaches to large-scale data analysis. In: SIGMOD, pp. 165–178 (2009)
15. Dell, Siemens Partner On Image Sharing, Archiving (2018). http://www.informationweek.com/healthcare/clinical-information-systems/dell-siemens-partner-on-image-sharing-archiving-/d/d-id/1102877?
16. Ambra Health (2018). https://ambrahealth.com/
17. Rascovsky SJ, Delgado JA, Sanz A, Calvo VD, Castrillón G: Informatics in radiology: use of CouchDB for document-based storage of DICOM objects. Radiographics 32(3), 913–927 (2012) [PubMed: 22403116]
18. Apache CoachDB (2018). http://couchdb.apache.org/
19. Savaris A, Harder T, von Wangenheim A: Evaluating a row-store data model for full-content DICOM management. In: 2014 IEEE 27th International Symposium on Computer-Based Medical Systems (CBMS), pp. 193–198 (2014)
20. Apache Cassandra (2018). http://cassandra.apache.org/
21. JavaScript Object Notation (JSON) (2018). http://json.org/
22. Yu C, Yao Z: XML-based DICOM data format. J. Digit. Imaging 23(2), 192–202 (2010) [PubMed: 19184223]
23. W3C. XML Path Language (XPath) (2018). http://www.w3.org/TR/xpath-30/
24. W3C. XQuery 1.0: An XML Query Language (2018). http://www.w3.org/TR/xquery/
25. Oracle XML DB.
26. Beyer K, Cochrane RJ, Josifovski V, Kleewein J, Lapis G, Lohman G, Lyle B, Özcan F, Pirahesh H, Seemann N, Truong T, Linden BV, Vickery B, Zhang C: System RX: one part relational, one part XML. In: Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data, Baltimore, Maryland (2005)
27. DeWitt DJ, Gray J: Parallel database systems: the future of high performance database processing. Commun. ACM 25(6), 85–98 (1992)

28. Database Partitioning, Table Partitioning, and MDC for DB2 9 (2018). http://www.redbooks.ibm.com/redbooks/pdfs/sg247467.pdf

29. DeCandia G, Hastorun D, Jampani M, Kakulapati G, Lakshman A, Pilchin A, Sivasubramanian S, Vosshall P, Vogels W: Dynamo: amazon's highly available key-value store. In: Proceedings of Twenty-First ACM SIGOPS Symposium on Operating Systems Principles, Stevenson, Washington, USA (2007)

30. MongoDB (2018). http://www.mongodb.com

31. Chang F, Dean J, Ghemawat S, Hsieh WC, Wallach DA, Burrows M, Chandra T, Fikes A, Gruber RE: Bigtable: a distributed storage system for structured data. ACM Trans. Comput. Syst 26(2), 1–26 (2008)

32. Garcia-Molina H, Ullman JD, Widom J: Database Systems: The Complete Book. Prentice Hall Press, Upper Saddle River (2008)

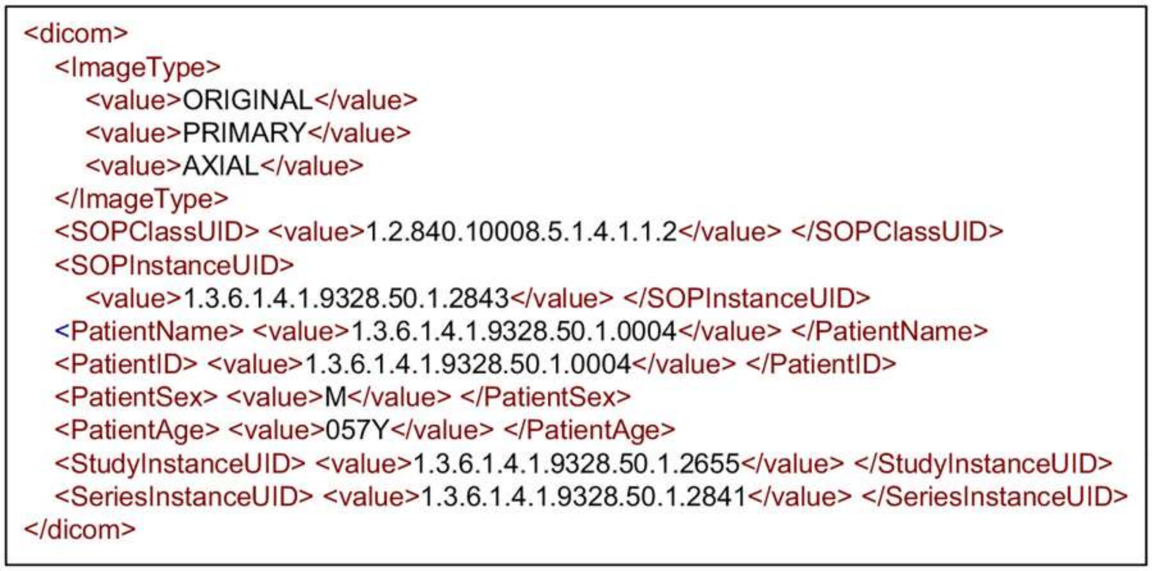
33. Gilbert S, Lynch N: Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. SIGACT News 33(2), 51–59 (2002)

34. DICOM PS3.18 2013—Web Services (2018). http://medical.nema.org/dicom/2013/output/pdf/part18.pdf

35. DCMRL/XMLStore (2018). https://github.com/tengdj/DCMRLXMLStore

36. DCMDOCStore (2018). https://github.com/tengdj/DCMDOCStore

37. The Cancer Imaging Archive (TCIA) (2018). http://www.cancerimagingarchive.net/



```
<dicom>
  <ImageType>
    <value>ORIGINAL</value>
    <value>PRIMARY</value>
    <value>AXIAL</value>
  </ImageType>
  <SOPClassUID> <value>1.2.840.10008.5.1.4.1.1.2</value> </SOPClassUID>
  <SOPInstanceUID>
    <value>1.3.6.1.4.1.9328.50.1.2843</value> </SOPInstanceUID>
  <PatientName> <value>1.3.6.1.4.1.9328.50.1.0004</value> </PatientName>
  <PatientID> <value>1.3.6.1.4.1.9328.50.1.0004</value> </PatientID>
  <PatientSex> <value>M</value> </PatientSex>
  <PatientAge> <value>057Y</value> </PatientAge>
  <StudyInstanceUID> <value>1.3.6.1.4.1.9328.50.1.2655</value> </StudyInstanceUID>
  <SeriesInstanceUID> <value>1.3.6.1.4.1.9328.50.1.2841</value> </SeriesInstanceUID>
</dicom>
```

Fig. 1.
XML based representation of DICOM headers (only a fraction listed)

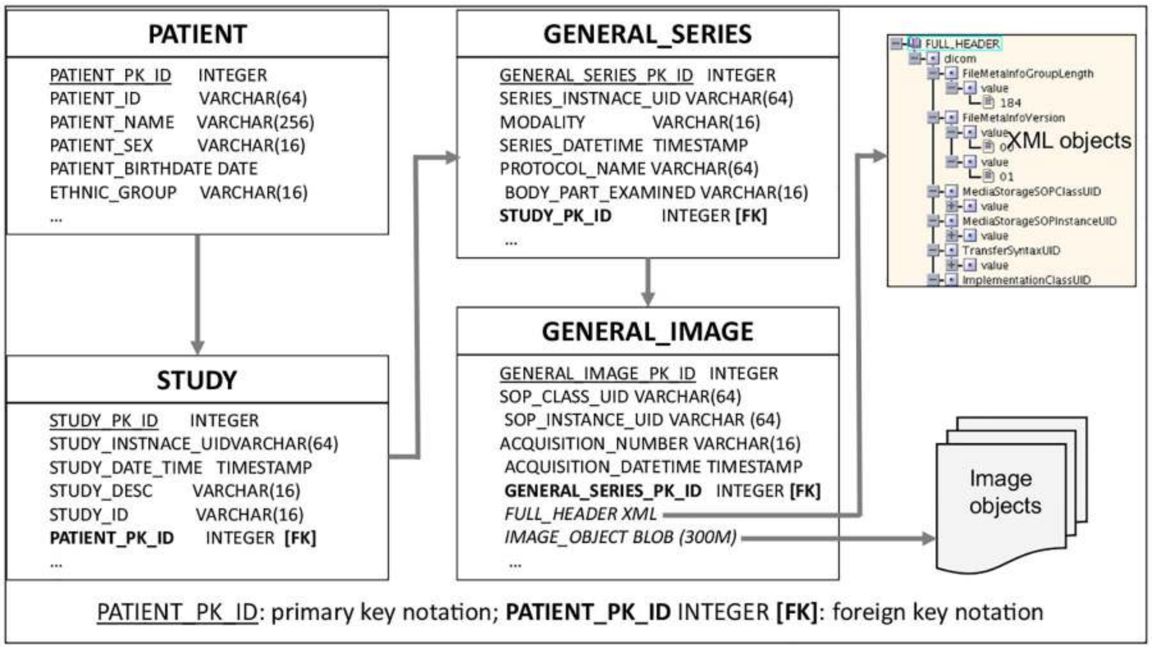

PATIENT
PATIENT_PK_ID INTEGER
PATIENT_ID VARCHAR(64)
PATIENT_NAME VARCHAR(256)
PATIENT_SEX VARCHAR(16)
PATIENT_BIRTHDATE DATE
ETHNIC_GROUP VARCHAR(16)
...
STUDY
STUDY_PK_ID INTEGER
STUDY_INSTNACE_UIDVARCHAR(64)
STUDY_DATE_TIME TIMESTAMP
STUDY_DESC VARCHAR(16)
STUDY_ID VARCHAR(16)
PATIENT_PK_ID INTEGER [FK]
...
GENERAL_SERIES
GENERAL_SERIES_PK_ID INTEGER
SERIES_INSTNACE_UID VARCHAR(64)
MODALITY VARCHAR(16)
SERIES_DATETIME TIMESTAMP
PROTOCOL_NAME VARCHAR(64)
BODY_PART_EXAMINED VARCHAR(16)
STUDY_PK_ID INTEGER [FK]
...
GENERAL_IMAGE
GENERAL_IMAGE_PK_ID INTEGER
SOP_CLASS_UID VARCHAR(64)
SOP_INSTANCE_UID VARCHAR (64)
ACQUISITION_NUMBER VARCHAR(16)
ACQUISITION_DATETIME TIMESTAMP
GENERAL_SERIES_PK_ID INTEGER [FK]
FULL_HEADER XML
IMAGE_OBJECT BLOB (300M)
...
FULL_HEADER
dicom
FileMetaInfoGroupLength
value
184
FileMetaInfoVersion
value
00
value
01
MediaStorageSOPClassUID
value
MediaStorageSOPInstanceUID
value
TransferSyntaxUID
value
ImplementationClassUID
XML objects
Image objects
PATIENT_PK_ID: primary key notation; PATIENT_PK_ID INTEGER [FK]: foreign key notation

Fig. 2.
Hybrid DICOM modeling with relational and XML models

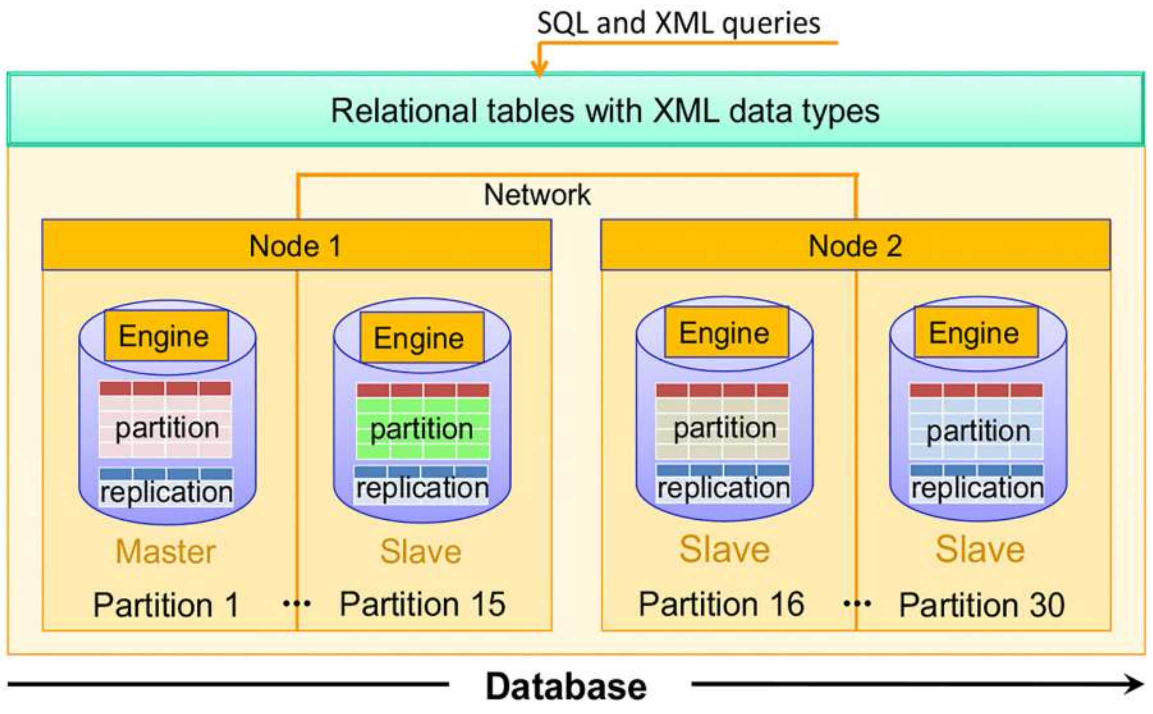
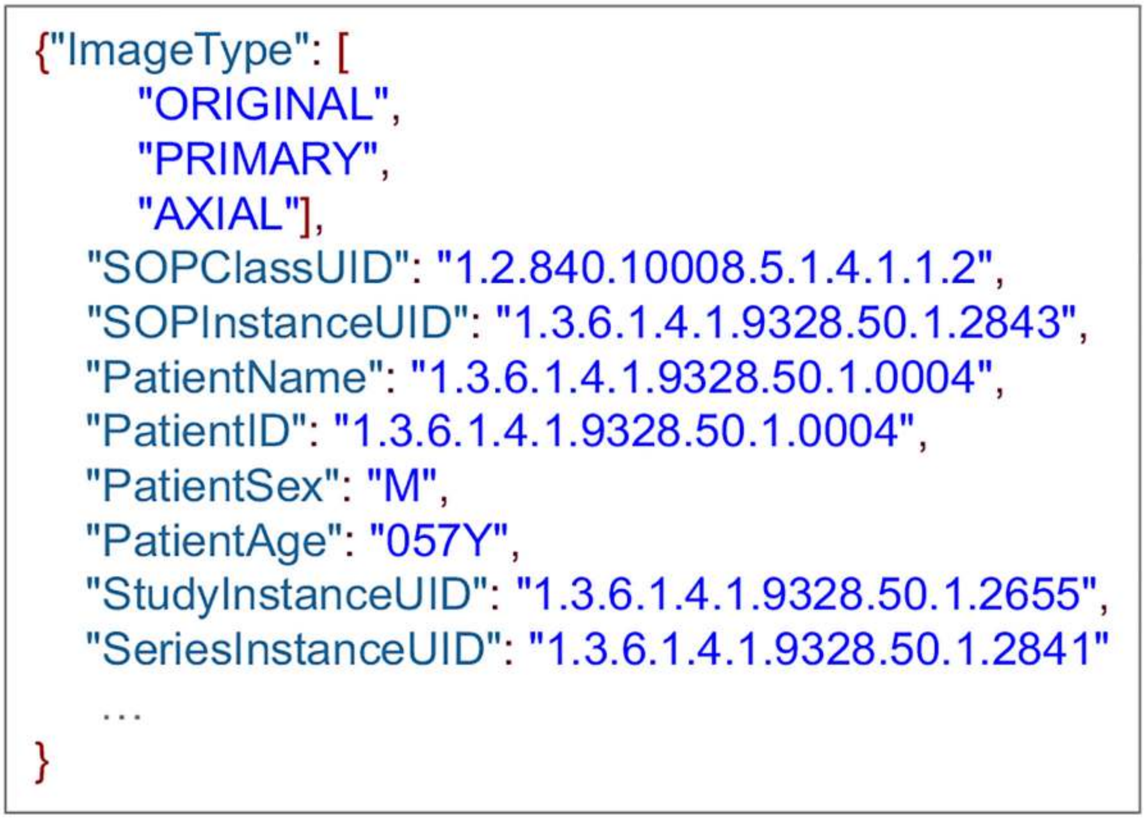


Fig. 3.
Shared-nothing parallel database architecture of the DCMRL/XMLStore



```
{"ImageType": [
        "ORIGINAL",
        "PRIMARY",
        "AXIAL"],
    "SOPClassUID": "1.2.840.10008.5.1.4.1.1.2",
    "SOPInstanceUID": "1.3.6.1.4.1.9328.50.1.2843",
    "PatientName": "1.3.6.1.4.1.9328.50.1.0004",
    "PatientID": "1.3.6.1.4.1.9328.50.1.0004",
    "PatientSex": "M",
    "PatientAge": "057Y",
    "StudyInstanceUID": "1.3.6.1.4.1.9328.50.1.2655",
    "SeriesInstanceUID": "1.3.6.1.4.1.9328.50.1.2841"
     ...
}
```

Fig. 4.
JSON based representation of DICOM headers (only a fraction listed)

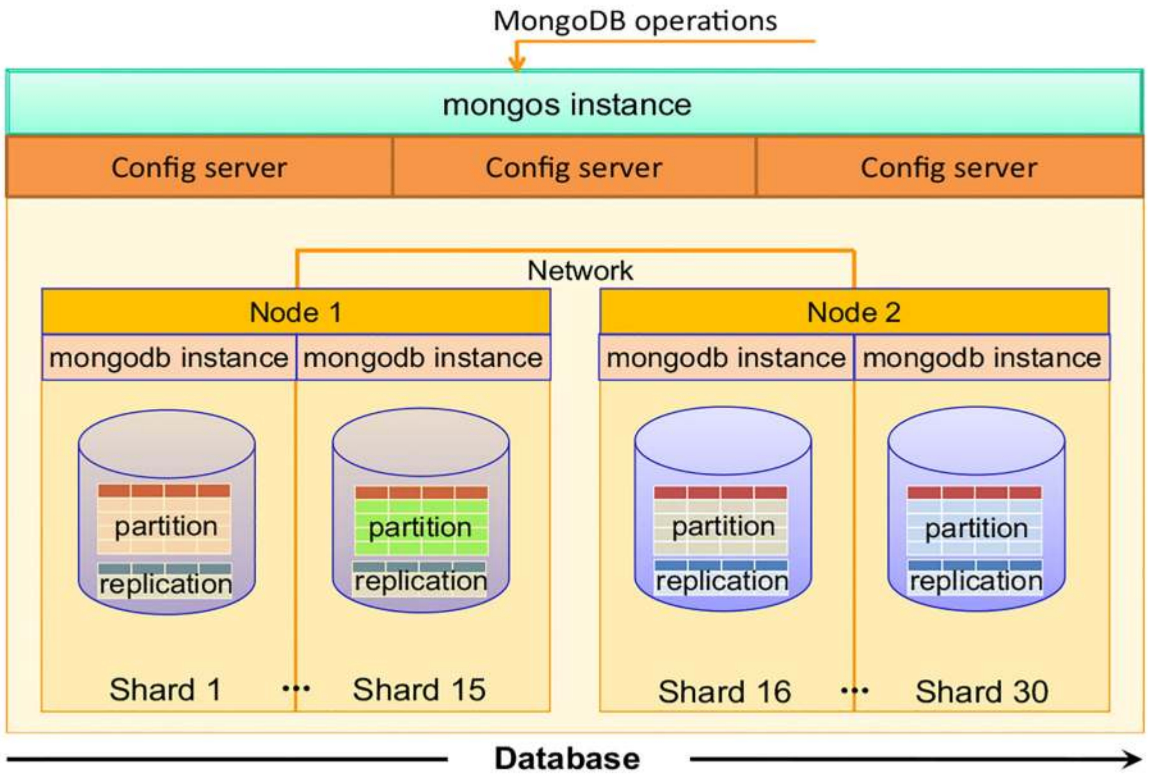


Fig. 5.
Shared-nothing parallel database architecture of the DCMDOCStore

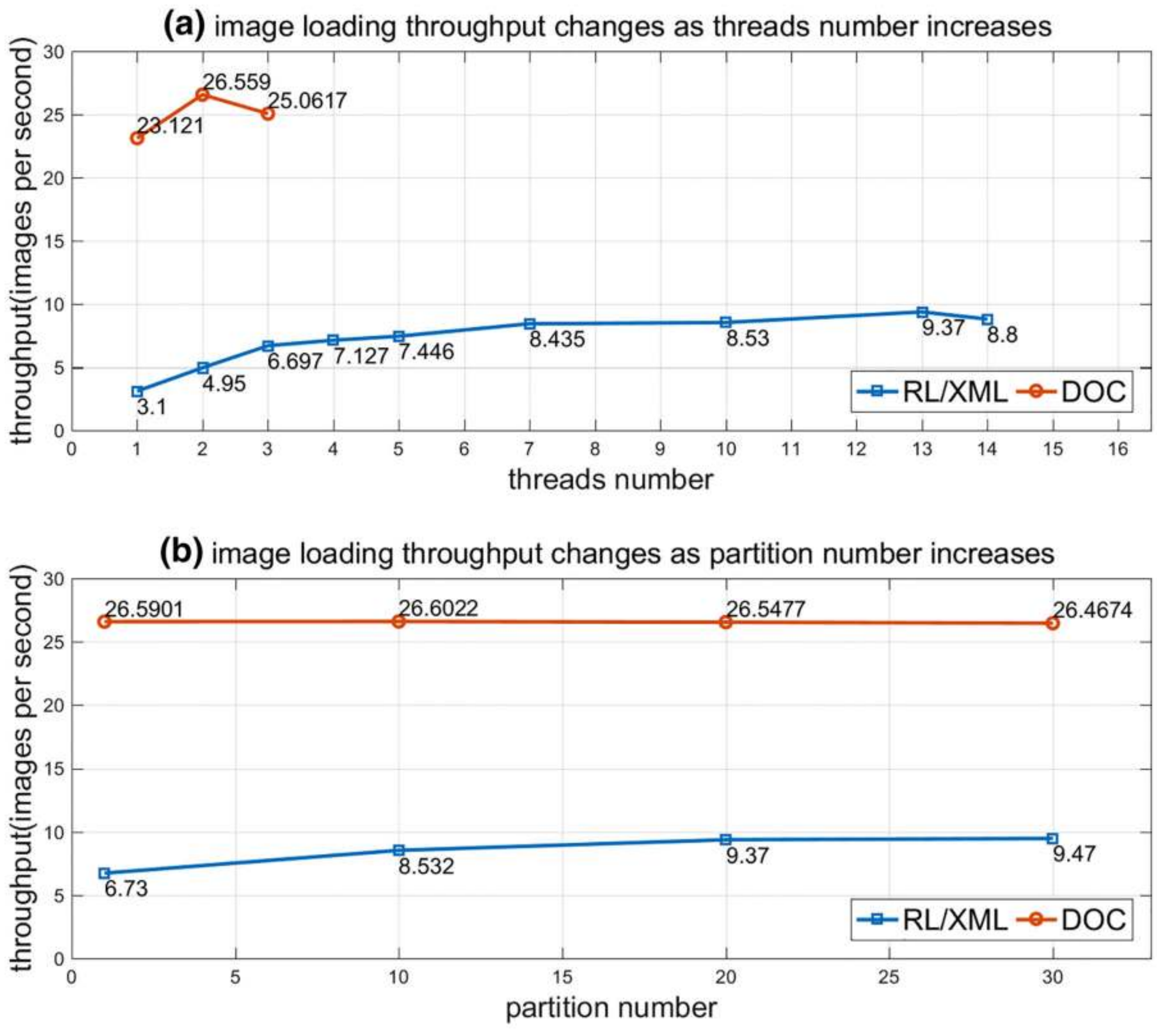


Fig. 6.
Image loading performance

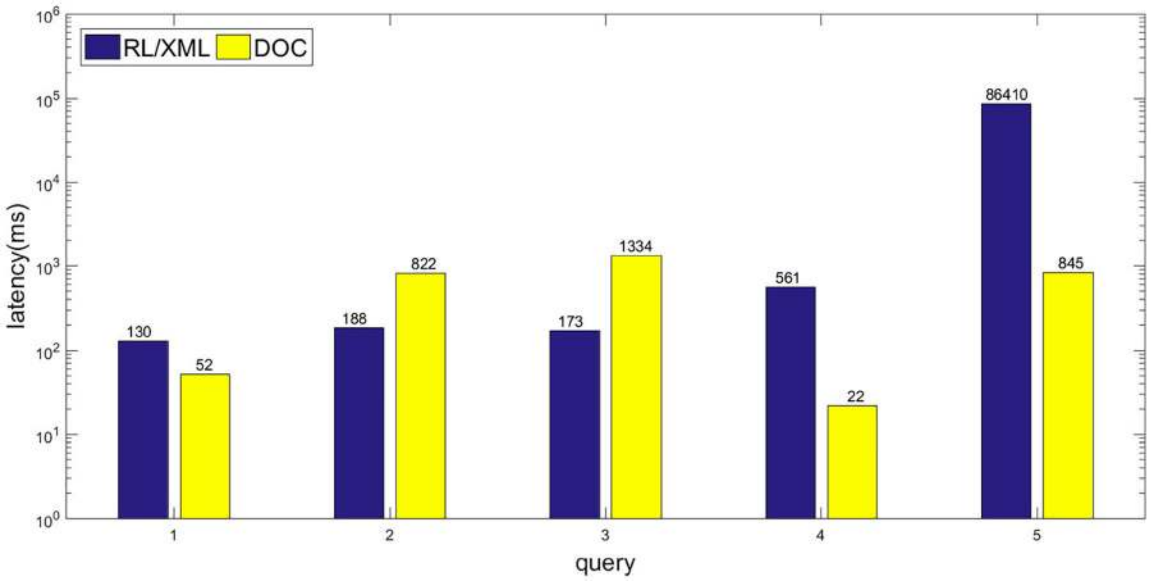


Fig. 7.
Performance of RL/XML and DOC on all queries

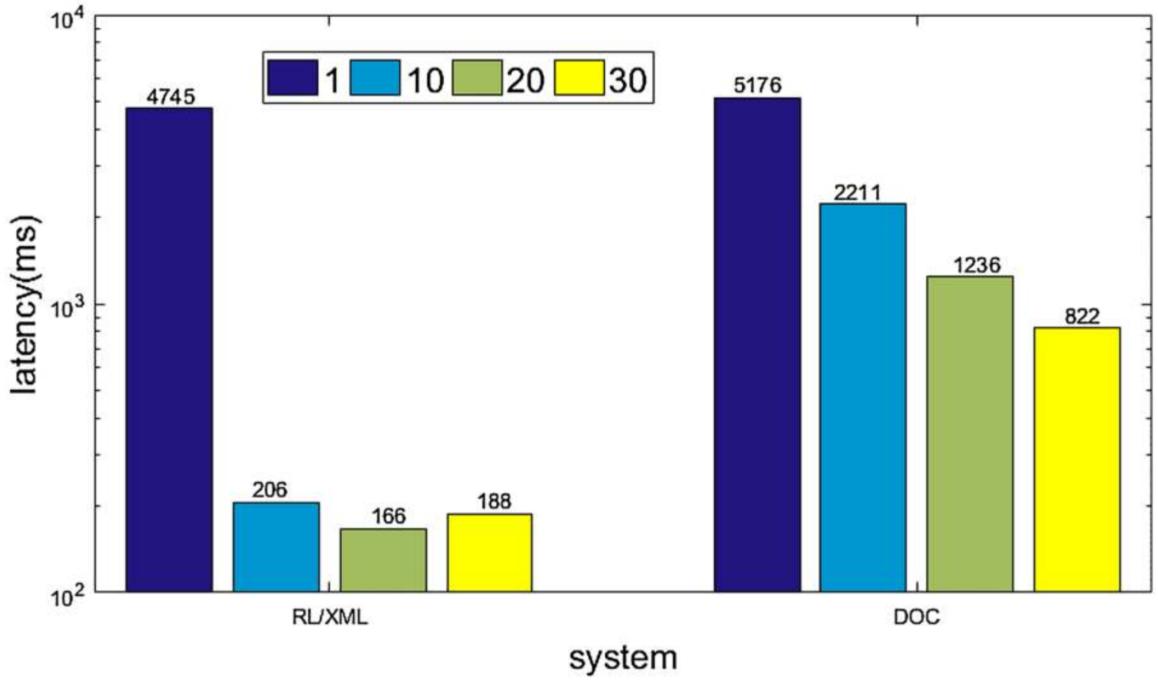


Fig. 8.
Influence of partition number for query 1

Table 1

Queries

Query	Description	Note
1	Retrieve all images of one patients	Random access with low selectivity
2	Retrieve all images of all patients excludes one's	Random access with high selectivity
3	Retrieve five patients with most images	Table scan/query with aggregation
4	Retrieve all images whose field ContrastBolusVolume covered by a certain value range	Query on field which is not predefined in RDBMS
5	Retrieve all images whose type field contains "ORIGINAL" and "PRIMARY" and "AXIAL"	Query on field with variable number of values, which is not supported by RDBMS