2010 24th IEEE International Conference on Advanced Information Networking and Applications

Outsourced XML Database: Query Assurance Optimization

Andrew Clarke and Eric Pardede
Department of Computer Science and Computer Engineering
La Trobe University, Bundoora, VIC 3083, Australia
Email: {pa2clarke@students.; E.Pardede@}latrobe.edu.au

***Abstract*—The area of XML database outsourcing, whereby the data owner enlists an external service provider to manage the storage and retrieval of their database, has been of increasing interest in recent years due to the relatively inexpensive nature of hardware/bandwidth, compared to the higher expense of in-house expert staff/software. As such it has become increasingly practical to use outsourced database solutions. However, as the service provider may not be fully trusted, XML database outsourcing introduces several security concerns that are new or more complex than those encountered in traditional database implementations. These include: data confidentiality, privacy, secure auditing, query assurance and secure and efficient storage. Of particular importance due to its relevance to most outsourced database models is query assurance – ensuring the database responds correctly to queries. In this paper, we propose the use of temporary time stamps and hash granularity to increase the efficiency of query assurance. This approach is tested against real datasets of varying type and size. Further, we consider how best to create time stamps and the issues associated with expiring versus distributed time stamp models.**

I. Introduction

The area of XML database outsourcing, whereby the data owner enlists an external service provider to manage the storage and retrieval of their database, has been of increasing interest in recent years due to the relatively inexpensive nature of hardware/bandwidth, compared to the higher expense of in-house expert staff/software. As such it has become increasingly practical to use outsourced database solutions.

However, as the outsourcing service may not be fully trusted (as shown in Figure 1), ensuring that data is not modified (correctness); the query is performed over all data (completeness); and represents the most up to date version (freshness) are significant concerns. These three issues are collectively known as query assurance and will be the primary topic of this paper. Further, query assurance needs to be approached in a way that does not create further issues – such as poor efficiency. Ideally the increases in overheads should be small enough to be transparent to end users, while still providing the benefit of offloading most of the work from the data-owners.

Various models for query assurance over outsourced XML have been proposed. However, most authentication approaches have a data-centric focus [1], [2], which is closely aligned to traditional database applications where data is strongly structured and search by value is of primary importance. Less attention has been given to document-centric models, where the structure and order of the XML is more meaningful than searching. Further, most approaches have not actively considered freshness. Generally it is assumed that time stamps will be distributed from the data owner/s to the clients by some method then matched against the database service provider. For a highly dynamic database, the overheads associated with this approach would be quite onerous. A different approach using expiring time stamps has been suggested briefly [3]. Storing time stamps and making decisions regarding expiry rates has not been fully investigated.

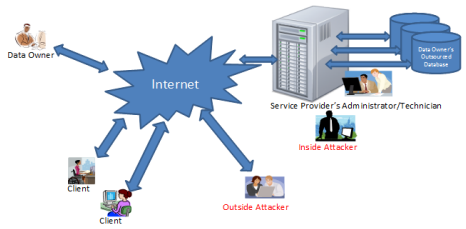

Fig. 1. Outsourced Database Model

Of concern to all approaches is efficiency. This has to be considered in a balanced way as quite often approaches that have high efficiency for the server, do so at the cost to the client/data owner [4], and the reverse is true. Performance measures in relation to efficiency are typically quantified by computational resources/time and data overheads.

The challenge of work in this area is to put forth a strong query assurance mechanism, that is efficient both in terms of data overheads and processing cost. These issues are of growing concern in recent times due to the shift to mobile devices, with associated CPU and bandwidth limitations.

Overall we make three contributions in this paper. Firstly, we propose a multi-verification tree approach that appropriately handles the queries of document and data centric XML. Secondly, the use of variable time stamps and hash granularity is considered for use in improving efficiency. Thirdly, a test implementation of real datasets is used to measure CPU and data overhead. We find that CPU overhead is not significant. However, our results show data overhead on average to be 52%, which can be reduced by our optimization approaches to 25%.

The rest of the paper is organized as follows. Section II reviews related work. Section III discusses the methodology used in our approach. Section IV presents our experiment setup

1550-445X/10 $26.00 © 2010 IEEE
DOI 10.1109/AINA.2010.130

and results. Section V presents our conclusion and details the possible future work in this area.

II. Related Work

Most approaches to query assurance can be categorized into two groups:

1) Probabilistic – Where the clients have some knowledge of the database, and combine queries over the known and unknown portions to provide reasonable certainty that the query results are correct.
2) Authenticated – Generally based on the concept of merkle hash trees [5], [4], [6]. The data owner uses hashing to create message digests of the database and time stamps, then uses public-key cryptography to validate query results. These hashes/signatures/time stamps are stored in a verification structure as per the scheme.

Much of the previous work on query assurance for outsourced database has been concerned with relational databases. However, there has been some recent work that directly addresses the unique properties of tree based databases, particularly XML.

A. Probabilistic Query Assurance

Probabilistic query assurance [7], [3], [8], [9] is based on the premise that if we know or can infer the contents of a portion of the outsourced database, we can with a high probability confirm that the content is correct, complete and fresh. This is done by performing a batch of queries over both the known and unknown sections of the database. If the known content is returned correctly, the unknown is also considered correct. This approach relies on being able to periodically insert/delete extra elements that are used for checks, and that are not detectable by the service provider. As a further issue, if even one client cooperates with a malicious service provider this approach can be invalidated [3] – as the service provider would then know what data it needs to return correctly and what it could modify.

The advantage of probabilistic verification is that it is possible to vary the overheads by relaxing or tightening required assurance levels [3] – and hence the amount of extra data needed in the database. As well, there is the ability to implement this approach without making modifications to the outsourcing server.

One approach that was used to mitigate the potential for the service provider being able to detect the fake data, was to use dual encryption [9]. Dual encryption uses two symmetrical encryption keys. The entire database is encrypted into blocks using the first key and outsourced to the service provider. Then a subset of the database is encrypted with the second key and outsourced to the service provider. As the service provider cannot detect the difference between the two types of encrypted blocks, it is not possible for the service provider to detect the verification data. As with other query assurance schemes, if the encryption key is passed onto the server, the security is lost. However, for most purposes the fake data is undetectable. The only weakness is that encrypted XML significantly restricts the type of queries the database can perform [10], [11].

B. Authenticated Query Assurance

1) Condensed-RSA and Aggregate Signatures: This was an early approach proposed for use in relational databases. This extension of RSA [4] is made possible due to its homomorphic nature. It is possible to combine a series of digital signatures by performing a product into one standard length RSA signature and preform one evaluation. This is an extremely time saving method especially in outsourced database operations where many signatures are retrieved. Condensed-RSA for relational databases was an early attempt to assure correctness of outsourced relational databases in a more efficient manner than was possible based on standard RSA/DSA. The efficiency increases are quite noticeable and Condensed-RSA had become the standard in later work. Condensed-RSA was later extended to cover completeness [6] through a method called signature chaining, whereby each signed record contains data relating to records on either side. This allowed range queries to be executed and correctness assured by returning the condensed/aggregated signature including the signature for the tuples directly outside the range on either side. BGLS (Boneh, Gentry, Lynn and Shacham) [12] is a competing aggregate signature scheme for correctness in outsourced relational database. BGLS (that is based on elliptic curves and bilinear mappings) allows the creation of a single signature even with multiple different signers. Although the BGLS scheme has better functionality and is more efficient for signing data than Condensed-RSA, the cost to verify data is very expensive [4]. This makes BGLS a good solution for a database with multiple signers where write/read operation are more balanced, rather than a model where a few data owners write to the database and many users query.

2) Embedded Merkle Hash Trees: Embedded Merkle Hash Trees[13] were created in an effort to produce a trade off between the initial construction costs of signature aggregate and chaining models and large verification objects of Merkle Hash Trees [4]. The trade off is accomplished by containing a Merkle B+ tree within each Embedded Merkle B+ tree node. The result is a considerably more collapsed tree, allowing for more compact verification objects, at the cost of higher setup/maintenance costs. The major strength of Embedded Merkle over other solutions is flexibility, as it is demonstrated that with minor modifications the trade off between setup cost and verification object size can be altered.

3) Partially Materialized Digest Scheme: Partially Materialized Digest Scheme is a recent approach that extends from the ideas apparent in Embedded Merkle b+ tree approach [13]. The Partially Materialized Digest Scheme [2] is based on the concept that hashing is less expensive than page file reads. So the verification tree is only partially constructed at any time and as queries are processed the necessary hash values are calculated. Similar to other approaches, the verification information is stored separately from the outsourced data, with the result that it is relevant to all database types. Further, it

is proposed that the user should be able to decide whether assurance is required, and when it isn't this approach is nearly as efficient as a regular database implementation. In comparison to the other verification object implementations, the Partially Materialized Digest Scheme performs well in both static and dynamic situations. Freshness isn't addressed directly by this approach but it is mentioned that it is compatible with distributed or expiring time stamp schemes.

4) Nested Merkle B+ Tree: An approach suggested that takes into account XML structure, Nested Merkle B+ trees [1] take a novel approach to query assurance of XML data. A path tree is constructed where the leaves of that tree are value and parent trees. Value trees sort the XML elements at that path by a key, and parent trees sort the parent information. In this way it is possible for the B+ tree to store most of the structural information of the XML database as well as the contents of elements. A noticeable issue is that if many search values are required the hash of an XML element is duplicated in many value trees, requiring more effort to complete update operations. Further, it does not store element order unless an extra value tree is added for that purpose.

C. Summary of previous approaches

The related work has brought forth some strong models that work well for the situations they are intended for. Probabilistic approaches can provide a reasonable level of assurance. However, they are very vulnerable to being defeated if the server has the same information as any user that accesses the database. Authenticated approaches on the other hand, have not fully addressed freshness. Further, even those approaches that directly address XML do not appear well suited for document-centric XML, where element order is meaningful and needs to be correctly queried, or situations where value based search is not particularly meaningful.

Further, all approaches have associated overheads. There is the potential to provide efficiency/optimization improvements to increase the viability of query assured outsourced XML.

Our approach extends from the earlier authenticated approaches by attempting to address efficiency in a new manner. That is, extra digital signatures and variable granularity are added to reduce the size of verification objects. Further, the verification tree structure is designed to cater better to both data and document centric XML with attempts to avoid expensive update operations.

III. Methodology

To address the issues involved with outsourced XML we propose an authenticated approach that varies significantly from previous work and addresses both data and document centric XML and further, attempts to increase efficiency through expiring time stamps. A multiple tree approach is used as shown in Figure 2, whereby the hashes of the XML elements are sorted by exact path (path including order in document) and stored in the leaf nodes. The leaf nodes are hashed back to the root and then digitally signed. To ensure completeness for search queries, index trees will be created and sorted based on that value and signed in the same manner as the path tree. However, the index trees do not contain the hash of the element they reference. Instead they store the exact path of the element. This is to decrease the amount of update overheads required.

Further, database information will be stored in the tree in the form of read and modification counts, to allow for efficiency modification and time stamp duration calculation.

The implementation will be client driven – that is the client requests verification data based on the query that it wants to process. This has the benefit of allowing the verification server to be stored with a different service provider. As the XML service provider does not have easy access to the verification information, cheating becomes more difficult. Further, it is then possible for a client to not request verification data to conserve bandwidth and it would be difficult for the XML service provider to detect. Finally, the disadvantages are that two server connections will need to be made, and the client will need to perform a query translation to match the XML query to the verification server.

A. Accessing

For each XML query the value is searched among the verification trees. Once found, boundary values to the left and right of the selection are included and the verification object is created by including the branches' values up the tree until the root or other signed node is found as shown in Figure 3. If the query was based on a search index, the index would first be queried and then the path tree queried based on the exact path retrieved from the index query.

Client driven XML accessing algorithm:

```
query XML database
query Verification server
receive XML result
receive verification object
lowerHash = hash(XML result)
while signature not reached do
    if lowerHash exists in verification object then
        hash (lowerHash)
    end if
end while
verify signature with highest level hash
if verification passes then
    XML is valid
else
    XML is invalid
end if
```

Once the data is returned it is hashed with the absolute path of the element. This is then checked against the returned value from the path tree. If the hash matches the stored value, the query is considered verified.

B. Updates/Refreshing

On an update operation, two updates are produced. One for the XML value/s that need updating and a second for the hash

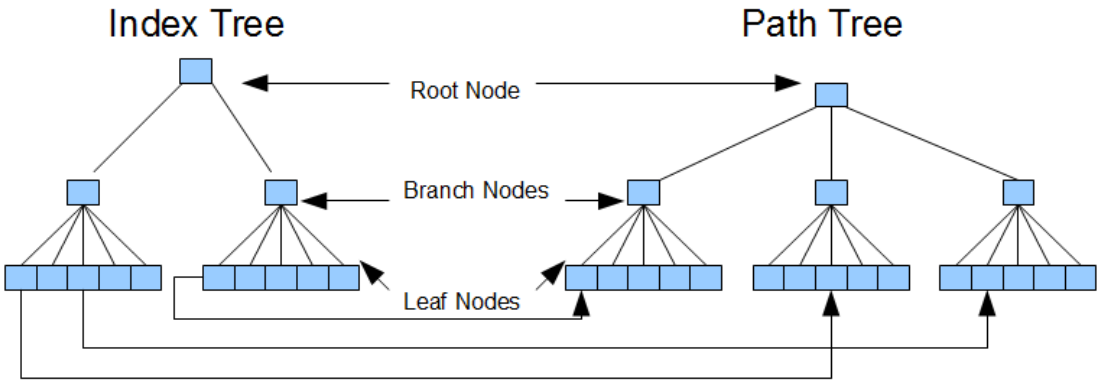

Fig. 2. Verification Tree

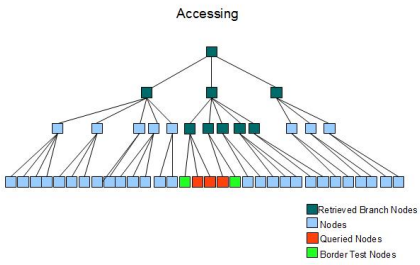

Fig. 3. Verification Tree Accessing

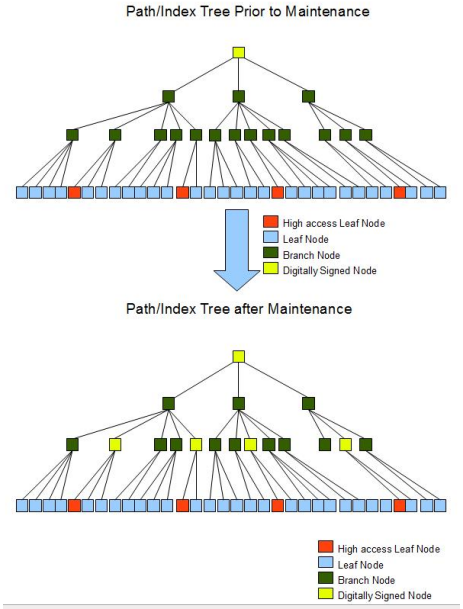

Fig. 4. Maintenance optimization of verification tree

value that was previously stored. On a hash update, parent hashes are re-calculated until a signed node is reached and resigned plus time stamp refreshed.

For time stamping, rather than a hand-off procedure whereby whoever makes a modification is responsible for freshness until the next modification [3], a master data owner who keeps the outsourced database fresh seems practical. Other data owners can make updates but do not take responsibility for freshness.

Refresh of verification tree time stamps algorithm:

1: Server maintains list of time stamped nodes
2: Server sends time stamp nodes to Data Owner
3: Data owner verifies signatures + time stamp authenticity
4: Data owner resigns the nodes and returns to Server

C. Maintenance

Of importance to this approach is the use of maintenance to adjust the verification trees to improve efficiency. These measures attempt to optimize the verification objects so that the most common queries are the most efficient based on the read rates.

The operations possible to increase efficiency are:

1) Increase or decrease hash granularity to more closely match the size of the queried elements.
2) Increase or decrease the duration of time stamps based on the modification rate.
3) Placing or removing additional time stamps in proximity to high access leaf nodes.
4) Adjust the depth and breadth of the tree by increasing or decreasing the maximum child nodes per branch.

The adjustment of time stamp placement is displayed in Figure 4. As shown, the high access nodes have digitally signed nodes added to a common ancestor. The outcome of this is that the size of the verification object (the amount of data that needs to be returned) is significantly reduced.

Maintenance algorithm:

1: **for** $i = 0$ to indexNo **do**
2: leaf = firstLeaf

```
    for j = 0 to leafCount do
      if leaf.readCount > threshold then
        temp = leaf
        while nextLeaf.readCount > threshold do
          leaf = nextLeaf
        end while
      end if
      if temp ! = null then
        find common ancestor(temp,leaf)
        apply time stamp + signature to ancestor
        temp = null
      end if
    end for
  end for
  for j = 0 to pathTreeNo do
    check readRecords
    if readRecords > granThreshold then
      re-hash XML document to finer granularity
    end if
    set time stamp duration based on modificationRate
  end for
```

Through the use of the above algorithm temporary variable time stamps will be placed close to the most frequently accessed nodes in both the index and path trees. Further, there is the ability to increase the granularity of the document based on read statistics if required.

IV. Results

A. Test Setup

The test setup was conducted on a x3 AMD 710 with 3GB RAM. The client and both server implementations were run on the one machine. To accurately measure XML DB overheads in comparison to verification services a full featured native XML database was utilized. In this case a Xindice [14] data collection running on a tomcat6 server was used. Client and verification servers were developed in Java with 1.6.0 runtime library. The verification tree objects were a custom class built on top of the DefaultMutableTreeNode java library. The tree objects loosely follow B+ tree structure – whereby the tree is self balancing and all records are stored in leaf nodes. The data setup comprises three major types:

1) A collection of ten RSS documents is used to demonstrate smaller document-centric XML files. They are outsourced with only path based verification as sorting is not useful in this context.
2) The CIA World Fact Book [15] is used to demonstrate a mid sized data-centric XML file. Path and index trees are created for this dataset.
3) Mondial [16] geographical database is used to simulate a large XML database. Path verification only is implemented.

The different types of XML files are used to demonstrate the algorithms' effectiveness in small to large files, as well as data-centric and document-centric XML. A range of different query types are attempted. Path, exact match and range search are conducted to measure the varying performances of the tree structures to each type. The sample set that is used on the XML/verification servers consists of a limited number of queries performed hundreds of times for a total of 10000 queries performed on each XML file type. The sample queries are performed once, directly following initialization being completed, then performed again on the post maintenance DB to measure the effectiveness of optimization.

As there are three databases in use, and each database is queried both pre and post maintenance, the total number of queries performed is approximately 60000.

The measurements recorded consist of data overhead and CPU computational time. For CPU overhead, database initialization, queries and maintenance will be investigated. For data overhead, the pre and post maintenance query performance will be used.

B. Initialization

As shown in Figure 5, our results demonstrate that on increasing the hash granularity, the XML setup time remains constant, while the verification tree setup time increases substantially. This is due to the RSA calculation that is performed on each addition. This could be somewhat mitigated by batching additions to the verification object together then performing a single digital signature.

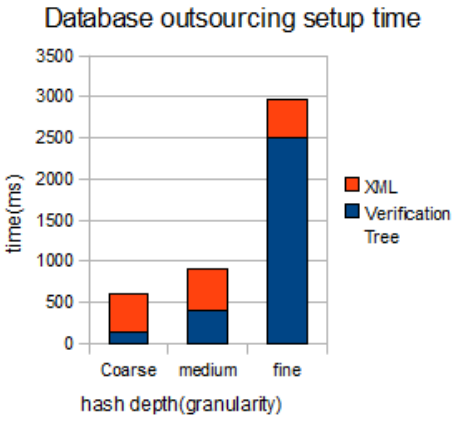

Fig. 5. Initialization Time

As a further exploration, we tested how the size of the buckets (or number of children) each branch contained affected setup time. It was found, as shown in Figure 6, that by increasing the bucket size, the setup time would decrease. These returns flattened off after a certain point was reached. These improvements were caused by the reduced number of split operations that needed to be performed on the verification tree.

C. Computation Time

Interestingly, as it is shown in Figure 8, the CPU time required to query the verification server was minimal compared to that involved with querying the XML server. This suggests that server CPU overheads are not a primary concern for query

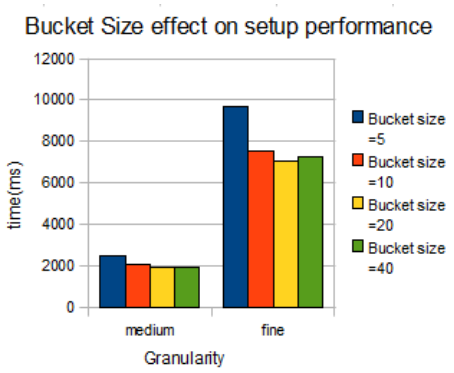

Fig. 6. Granularity effect of setup performance

assurance over outsourced XML. Further, it gives support to the client driven approach as the client can then retrieve the verification object, have it processed and ready to compare to the XML data well before the XML query is returned, rather than waiting for both.

As a further indication of CPU time, the maintenance operation was performed in 0.5% of the test runtime. Though this suggests that maintenance is too expensive to run in real time, its overall effect on server resources is not significant if run on a schedule.

D. Query Results

In this section we analyze the pre maintenance and post maintenance query results.

The pre maintenance results are interesting – verification data over the entire query set was a 52% overhead. This is quite significant. In the case of the CIA World Factbook results shown in Figure 9, the verification data was larger than the XML data returned. This is an undesirable outcome. As would be expected, the amount of verification data overheads increased as the size of database and hence the size of the verification trees increased.

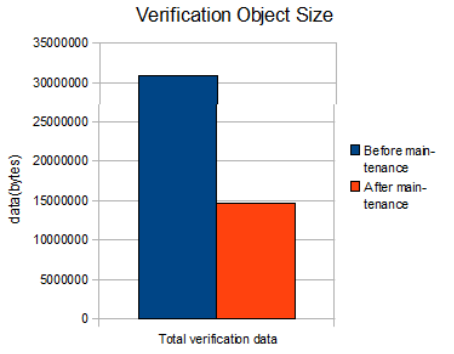

Fig. 7. Pre versus post verification object size

The post maintenance query showed strong improvements with overall verification overheads reduced by 52% as shown in Figure 7. The XML data retrieved remained the same. Further, the results varied based on the size of the database. The smallest data set was the RSS files which reduced their verification data overheads by 30%; the medium CIA file reduced by 47%; and the large Mondial file data overhead was reduced by 63%. These are significant optimizations and the improvements scale well from both small to large databases. Further, as this optimization was more effective for large database files than for smaller, it reduces the increase from the best to worse case. For our data this consists of the RSS versus Mondial database, with the difference in overheads being a decrease from 314% to 166% – a significant improvement.

Finally, though verification CPU overheads were not a significant consideration in query processing. The verification object computational time had a marginal improvement of 12%

E. Discussion

Though significant efficiency improvements were shown in the results through this optimization technique, there are various areas of strengths and weaknesses.

This approach used a more dynamic optimization technique than what is usually implemented to improve tree efficiency. The advantage of this is that it will attempt to optimize on short term trends. However, more long term optimization improvement can be gained through nesting/embedding [1], [13] to improve the entire structure of the tree rather than improving small portions on a temporary basis.

Further, this approach relies on the assumption that to some extent client access habits will be pattern based. If client access habits were purely random the efficiency of this approach would decrease.

As this approach uses an exact path basis to sort the path tree, it takes on some of the attributes of an array. The advantage of this is that there are less updates required on the index trees as they only require an update if the path or their search value changes. This has a further advantage of allowing more relaxed time stamp duration on the index tree as modifications will be less frequent. However, the downside is that delete/insert operations will be more expensive, as proceeding elements will need to be updated with their new path as will all indexes.

A non trivial problem of using server statistics of read/modification rates to decide how to optimize the database is that the read rates cannot be ensured. The potential exists for the service provider to change these statistics in a manner that would make the service inefficient. Though query assurance would still be ensured the cost would be steeper. The most logical way to approach this weakness is through secure auditing [10], which is another area of outsourced XML database research. However, it would be possible to mitigate the risk of service provider misbehaviour in this way by client data collection opt-in and comparing that data against the service providers statistics.

V. Conclusion and Future Work

This work explored the area of query assurance in outsourced XML databases. Procedures to more efficiently provide that assurance and the best approach to ensure freshness

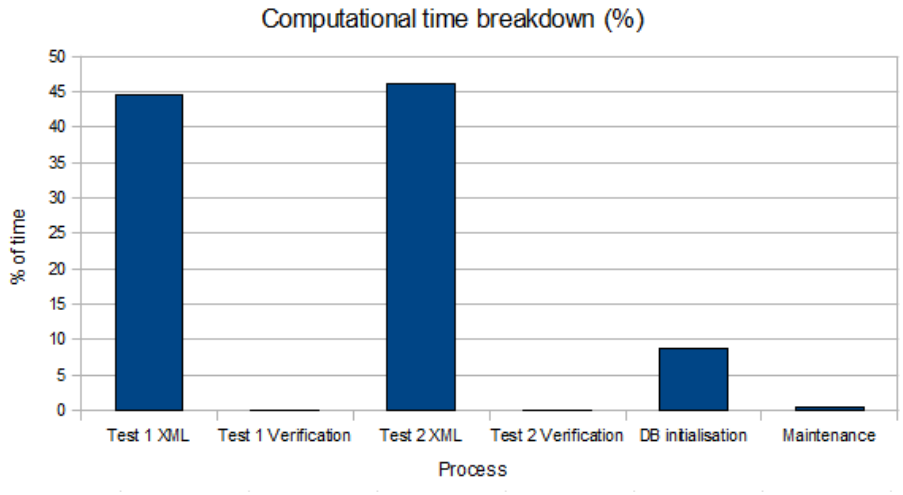

Fig. 8. Computational Time

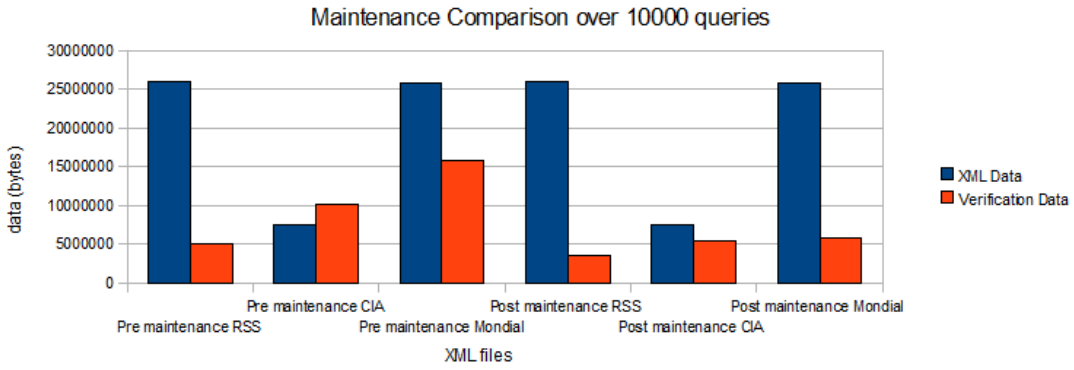

Fig. 9. Pre versus Post Maintenance Comparison

for both data and document centric XML databases were investigated. In particular an exact path approach to storing verification data and an indexing scheme for verification with increased efficiency through the use of temporary time stamps and variable levels of hash granularity were considered. Further, time stamp creation and the issues involved with read/modification counts were explored.

A test implementation was developed to measure the efficiency of this type of verification. The tests were run alongside a native XML DB (Xindice) to more accurately assess overheads. These were conducted over large query test sets (60000 queries) and performed over a range of different sample files, ranging from small RSS feeds to large XML data files.

Interestingly, it was shown that verification overheads in terms of time/CPU resources were not significant compared to that of XML queries. The results showed that the most significant area of verification overheads was data and that the overheads could be significantly reduced by the use of temporary time stamps to optimize the verification objects. Reductions of as much as 63% occurred in large files.

In the future an interesting study would be using the methods in this work in an adaptive and self maintaining/balancing way that applies the most efficient range of optimizations based on the database type and usage.

An extension of this is investigating the effect that this verification approach has on other more complex FLWOR (FOR LET WHILE ORDER RETURN) XQuery expressions. Previous work has focused on path and range searches, rather than on the effect of join type queries.

A related area of future work would involve applying this approach as part of an embedded/nested tree approach. In this approach verification trees are adaptive based on their size/usage and restructure in both temporary ways (time stamps) and more long term ways (nesting/embedding).

Finally, there is the challenging future work of implementing query assurance as part of a larger approach to outsourced database security. This involves the consideration of the other areas of outsourced database security – data confidentiality, privacy, secure auditing and secure and efficient storage.

REFERENCES

[1] V. H. Nguyen and T. K. Dang, "A novel solution to query assurance verification for dynamic outsourced xml databases," *Journal of Software*,

vol. 3, no. 4, pp. 9–16, 2008.

[2] K. Mouratidis, D. Sacharidis, and H. Pang, "Partially materialized digest scheme: an efficient verification method for outsourced databases," *Very Large Data Base*, vol. 18, no. 1, pp. 363–381, 2009.

[3] M. Xiey, H. Wang, J. Yin, and X. Meng, "Providing freshness guarantees for outsourced databases," in *Proceedings of the 11th international conference on Extending database technology: Advances in database technology*, vol. 261, 2008, pp. 323–332.

[4] G. T. Einar Mykletun, Maithili Narasimha, "Authentication and integrity in outsourced databases," *ACM Transactions on Storage (TOS)*, vol. 2, no. 2, pp. 107–138, 2006.

[5] P. T. Devanbu, M. Gertz, C. U. Martel, and S. G. Stubblebine, "Authentic third-party data publication," in *Database Security*, 2000, pp. 101–112.

[6] M. Narasimha and G. Tsudik, "Authentication of outsourced databases using signature aggregation and chaining," in *Lecture Notes in Computer Science 3882*, 2006, pp. 420–436.

[7] R. Sion, "Query execution assurance for outsourced databases," in *Proceedings of the 31st international conference on Very large data bases*, 2005, pp. 601–612.

[8] M. Xie, H. Wang, J. Yin, and X. Meng, "Integrity auditing of outsourced data," in *Very Large Data Base*, C. Koch, J. Gehrke, M. N. Garofalakis, D. Srivastava, K. Aberer, A. Deshpande, D. Florescu, C. Y. Chan, V. Ganti, C.-C. Kanne, W. Klas, and E. J. Neuhold, Eds. ACM, 2007, pp. 782–793.

[9] H. Wang, J. Yin, C.-S. Perng, and P. S. Yu, "Dual encryption for query integrity assurance," in *Conference on Information and Knowledge Management*, J. G. Shanahan, S. Amer-Yahia, I. Manolescu, Y. Zhang, D. A. Evans, A. Kolcz, K.-S. Choi, and A. Chowdhury, Eds. ACM, 2008, pp. 863–872.

[10] T. K. Dang, *Open and Novel Issues in XML Database Applications: Future Directions and Advanced Technologies*. Information Science Reference, 2009, ch. XI: Security Issues in Outsourced XML Databases.

[11] ——, "Ensuring correctness, completeness, and freshness for outsourced tree-indexed data," *Information Resources Management Journal*, vol. 21, no. 1, pp. 59–76, 2008.

[12] D. Boneh, C. Gentry, B. Lynn, and H. Shacham, "Aggregate and verifiably encrypted signatures from bilinear maps," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, ser. Lecture Notes in Computer Science, E. Biham, Ed., vol. 2656. Springer, 2003, pp. 416–432.

[13] F. Li, M. Hadjieleftheriou, G. Kollios, and L. Reyzin, "Dynamic authenticated index structures for outsourced databases," in *Special Interest Group on Management Of Data Conference*, S. Chaudhuri, V. Hristidis, and N. Polyzotis, Eds. ACM, 2006, pp. 121–132.

[14] "Apache xindice," http://xml.apache.org/xindice/. [Online]. Available: http://xml.apache.org/xindice/

[15] "Sample dataset – cia factbook: Country data," http://www.dbis.informatik.uni-goettingen.de/lopix/lopix-mondial.html. [Online]. Available: http://www.dbis.informatik.uni-goettingen.de/Mondial/lopix/cia.xml

[16] "The mondial database," http://www.dbis.informatik.uni-goettingen.de/Mondial/. [Online]. Available: http://www.dbis.informatik.uni-goettingen.de/Mondial/mondial.xml