Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 121 (2017) 260–265

Procedia Computer Science

www.elsevier.com/locate/procedia

CENTERIS - International Conference on ENTERprise Information Systems / ProjMAN - International Conference on Project MANagement / HCist - International Conference on Health and Social Care Information Systems and Technologies, CENTERIS / ProjMAN / HCist 2017, 8-10 November 2017, Barcelona, Spain

Identifying New Directions in Database Performance Tuning

[a] Derek Colley*, [b] Dr. Clare Stanier

[a]*School of Computing and Digital Technologies, Staffordshire University, Mellor Building, College Rd, Stoke-on-Trent ST4 2DE, UK*
[b]*School of Computing and Digital Technologies, Staffordshire University, Mellor Building, College Rd, Stoke-on-Trent ST4 2DE, UK*

Abstract

Database performance tuning is a complex and varied active research topic. With enterprise relational database management systems still reliant on the set-based relational concepts that defined early data management products, the disparity between the object-oriented application development model and the object-relational database model, called the object-relational impedance mismatch problem, is addressed by techniques such as object-relational mapping (ORM). This, compounded with changes in the way data is produced, stored and managed can result in generally poor query performance for SQL produced by object-oriented applications and an irregular fit with cost-based optimisation algorithms, and leads to questions about the need for the relational model to better adapt to a more diverse set of queries. This paper discusses existing database performance optimisation techniques and approaches and makes the argument that current database performance tuning approaches need revisiting to support queries developed through ORM tools. This paper also introduces our current research, which includes exploring concepts such as dynamic schema redefinition; query analysis and optimisation modelling driven by machine learning; and augmentation of the cost-based optimiser model.

Keywords: Database, SQL, performance tuning, cost-based optimiser, object-relational mapping, object-relational impedance mismatch

* Corresponding author. *Tel.*: +44(0) 161 298 5115
E-mail address: derek.colley@research.staffs.ac.uk

1. Problem Summary

Relational database systems (RDBMS) underpin a large number of today's business enterprises and RDBMS performance tuning is a well-understood field. However, relational databases have been extended over time to include object-oriented support and integration with external languages[36] and as application development paradigms have advanced, performance issues have emerged. This is particularly the case when dealing with non-static, fluctuating application models which interface with RDBMSs through paradigms such as object-relational modelling. These techniques have shown that automatic generation of SQL can lead to sub-optimal query performance in a relational environment[11,17,22] and means there is a need to identify new approaches for performance tuning to keep pace with the progress in application development methodologies. This paper reviews some existing RDBMS performance optimisation methods and comments on the strengths and limitations of traditional approaches in the current database environment; this paper also suggests directions for future work which include introducing agility and dynamic capabilities into query optimisation approaches with techniques such as pattern classification using machine learning. The rest of this paper is organised as follows. Section 2 gives the context of the investigation. Section 3 discusses current approaches to database performance tuning and Section 4 gives the conclusions and suggestions for future work.

2. Research Approach

A literature review of RDBMS performance tuning approaches was conducted. The first stage of the review was to identify seminal papers through key phrase searches. Papers were then ranked using a citation function to prioritise key sources. Topic and key conclusions were extracted from the paper and fitted into a directed graph. The process was then iterated. More than 100 highly cited papers were identified but the volume of material relating to RDBMS performance tuning means that only the sources identified as most relevant are discussed in this paper. A limitation of the approach was that the focus on seminal papers means the method is retrospective. For this reason, the review was expanded to ensure that more recent research was also included.

3. Performance Tuning

From the literature review, three key areas relating to query performance were identified: database design, query optimisation and query design.

3.1. Database Design Considerations

Relational databases are based on relational set theory and effective RDBMS design supports queries based on the relational algebra. However, although relational design concepts are well understood, adherence to good database design patterns is not enforced in the industry, nor arguably is it now even encouraged[1]. The primary technique for achieving optimal relational design is normalisation[14,15], although normalisation is often criticised for unnecessary complexity[8] particularly when regarding JOINs between tables[43]. A key assumption in the work on JOIN optimisation is that query design is driven by the schema design; in other words, that queries are developed to work with a given schema as efficiently as possible. This is not necessarily the case with SQL queries generated by applications or through mapping; for example, the ‘N+1’ problem is endemic in topologies with dependent relationships between entities (tables) and, where ORM-driven lazy loading is used, where data about the parent entities is returned on a row-by-row rather than a set basis. This results in multiple queries where only a single one is needed, causing performance issues[19]. Object-relational modelling techniques are more susceptible to this and other performance issues[17,22], especially against heavily-normalised or complex schemata.

3.2. Query Execution Optimisation

Query execution is a well-established process in RDBMSs[39] and in implementation, is handled by the cost-based optimizer (CBO) which has replaced rule-based optimisation[12]. Cost-based optimisation techniques attempt to reduce the cost of a query (measured by a variety of factors including time taken to execute; page accesses; selectivity factors; cardinality estimates; data density and more) by choosing the least costly plan[2] typically using heuristics or timeout parameters as a stop condition[21,35]. One limitation of the CBO is that cardinality is a principal factor in calculating the costs of a plan, since the number of rows is normally in direct proportion to the disk accesses required or the size of the dataset returned[34]. However, when multiple attributes are involved in a query, attribute-value independence (AVI) becomes a problem since the cardinality error multiplies proportionally to the number of attributes involved[18] and the intermediate relations[13]. The limitations of the CBO include difficulty in handling object-oriented features[23] and difficulties with nested queries. Wu et al[44] investigated whether cost-based optimiser models were now unusable due to query complexity. In the context of automatically generated queries which originate from ORM frameworks, the cost-based optimizer can struggle to efficiently analyse these complex queries and produce a viable plan, leading to problems such as sub-standard plan output caused by timeout conditions; plans which are not optimised for the parameterised inputs and plans that use inefficient JOIN mechanisms or poorly-performing scan operations as selection of the correct indexes could not take place.

Indexing is used to reduce the computational and I/O subsystem loads when fetching data[20,40]. The limitations of indexes in a traditional relational environment include performance penalties on write-heavy tables[16] and the overhead of indexes themselves[38]. Poor query design can mean that the RDBMS engine cannot apply indexes accurately, meaning that indexing can become inefficient – for example, if a query selects a column which is not present in the definition of the best-fit index, then additional row-by-row lookups may be required back to the base data to fetch the columnar values[31]. When selecting large quantities of rows using this technique, for example by the ORM method of ‘eager fetching’, this strategy can cause performance delays. In addition to indexing, there are a range of other strategies including partitioning[27]; load balancing[3]; and varying transaction isolation levels[21,30]. As with performance tuning based on efficient design, the underpinning assumption is that optimisation strategies implemented at database level will be used in queries developed at application level and that the query design is based on an understanding of relational optimisation techniques, which, as already noted, is not necessarily the case for queries automatically generated by object-relational mapping. As discussed, this mismatch of priorities causes performance issues, and could be addressed by the implementation of more adaptable methods of performance optimisation within the RDBMS rather than focused on inbound queries.

3.3. SQL Query Design

SQL has been described as an “elephant on clay feet”[1] for various reasons, including the necessary expansion of SQL to include object support (such as the support for user-defined types) but SQL syntax is well understood despite the expansion of the standard. There are a range of heuristics and techniques for optimal query design including sort tuning and the use of aggregations[6,7]; views[28]; cache management[11,42] ;use of set-based logic, not iterative logic; parallelism[26]; and the correct use of data typing. JOIN performance is a particularly important aspect as intrinsic performance issues caused by attribute-value independence, increased and varied storage reads and limitations in the generation of good-quality execution plans can have a major impact on query execution time. RDBMSs support a range of different types of joins and optimisation of JOIN performance is a continuing theme in the relational database literature[4,5,9,10,33], remaining a current research area[3,27,32]. In conventional database development, poor JOIN performance can result from many causes including over-normalisation[21]; inefficient JOIN type selection; data skew[25]; or external factors such as network performance[37] and processor architecture[24]. Given the increasing complexity of queries that may be generated from ORM frameworks, the impact of poor JOIN performance becomes more evident. In the context of ORM generated queries, one key element in any future relational database performance optimisation framework must be the implementation of contextually-correct JOINs and the de-normalisation of overly-normalised schemas.

Nested queries are also an issue. Queries generated by object-relational modelling tools can produce queries with multiple levels of nesting and large numbers of base tables, increasing the number of relations from which to extract data and increasing the query execution load through additional operations on the data (filtering and sorting). Other related reasons for poor database performance arising from ORM include pre-fetching rows before filtering; the aforementioned 'N+1' problem; fetching columns where not specified in the query, which will cause scans rather than index seeks and consequently greater I/O consumption; poor data typing; and for RDBMS systems with plan caches, excessive bloating of the cache through the generation of single-use plans[19,29,31].

4. Existing Performance Tuning Approaches: Limitations and Research Directions

Current performance tuning approaches appear to be generally schema-centred. Although queries can be refactored or limited in scope to decrease complexity, auxiliary structures to the schemas like indexes assist in efficiently searching the schemas; the cost-based optimiser uses trial-and-error to find better execution plans; techniques such as views abstract the schema objects; other approaches like partitioning focus on vertically or horizontally splitting the data within the schema objects. However, these approaches are tied to the existing structure of the schema and are static in nature. Schemas are generally fixed as changes to relations (for example, re-typing, re-definition of functional dependencies or adding or subtracting columns) have subsequent effects on other database objects; another way of stating this would be to say that tables and the objects that make up a table are closely coupled, implying brittleness, with an increased effort and risk associated with changes. With application development methodologies now iterative and techniques such as continuous integration commonplace, we have observed and can continue to expect continuing disruption to the traditional relational model.

We argue that a more intelligent, agile, dynamic query-driven approach is required to adapt to the challenges of changing query patterns. Trummer and Koch[41] arguably signal a move towards intelligent optimisation by showing how multiple factors can be used to influence the optimisation process. There have been numerous other research contributions towards a new kind of query optimisation, for example optimising for multi-core environments[26] and adaptive (dynamic) partitioning strategies[27]. Another avenue for investigation is to use a machine-learning led approach to categorise inbound queries in such a way that alternative versions of a schema can be used depending on the properties of a query, an approach we term 'dynamic schema redefinition' and which is a focus of our ongoing research. Another potential solution might be the examination of the data and the automatic aggregation, categorisation, partitioning, indexing or archiving of the data depending on both the static properties (length, type etc.) and the temporal properties (value over time, accuracy, velocity and so on). The key limitation to overcome is the inflexibility of relational schemas to respond to variable workloads.

5. Conclusions and Future Work

Since the inception of the relational model, there have been few significant changes to fundamental concepts. The ubiquity of relational database systems produced a comprehensive set of strategies and techniques to optimise query performance, but, as the discussion in this paper shows, these strategies and techniques are schema-oriented in that they rely for their effectiveness on queries being designed to fit the database, for example, by structuring the query to be accessible to the CBO. The rise of model-driven queries presents new challenges for relational query optimisation, identifying the need for revisions to the CBO and novel approaches, such as dynamic schema redefinition, or augmentation of the CBO with novel and dynamic techniques, in response to changing inbound query patterns. It is intended these approaches will be the underpinnings of our future research in this area, and these new approaches in the relational space would improve relational query performance for model-generated queries and further address the object-relational impedance mismatch problem.

References

1. Atzeni, P., Jensen, C.S., Orsi, G., Ram, S., Tanca, L. and Torlone, R., 2013. The relational model is dead, SQL is dead, and I don't feel so good myself. ACM SIGMOD Record, 42(2), pp.64-68.
2. Babcock, B. and Chaudhuri, S., 2005, June. Towards a robust query optimizer: a principled and practical approach. In Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data (pp. 119-130). ACM.
3. Barthels, C., Müller, I., Schneider, T., Alonso, G. and Hoefler, T., 2017. Distributed Join Algorithms on Thousands of Cores. Proceedings of the VLDB Endowment, 10(5).
4. Begley, S., He, Z. and Chen, Y.P.P., 2016. PaMeCo join: A parallel main memory compact hash join. Information Systems, 58, pp.105-125.
5. Blanas, S., Li, Y. and Patel, J.M., 2011, June. Design and evaluation of main memory hash join algorithms for multi-core CPUs. In Proceedings of the 2011 ACM SIGMOD International Conference on Management of data (pp. 37-48). ACM.
6. Borodin, A., Kiselev, Y., Mirvoda, S. and Porshnev, S., 2016, November. Development of data aggregation capabilities in domain-specific query language for metallurgy. In Dynamics of Systems, Mechanisms and Machines (Dynamics), 2016 (pp. 1-6). IEEE.
7. Bose, A., Smadi, M.M., Sun, J. and Velpuri, C.K., International Business Machines Corporation, 2016. Dynamic data aggregation from a plurality of data sources. U.S. Patent 9,292,575.
8. Buelow, R. "The Folklore of Normalization." Journal of Database Management, vol. 11, no. 3, 2000, p. 37.
9. Chen, M. and Zhong, Z., 2014, December. Block Nested Join and Sort Merge Join Algorithms: An Empirical Evaluation. In International Conference on Advanced Data Mining and Applications (pp. 705-715). Springer International Publishing.
10. Chen, S., Ailamaki, A., Gibbons, P.B. and Mowry, T.C., 2007. Improving hash Join performance through prefetching. ACM Transactions on Database Systems (TODS), 32(3), p.17.
11. Chen, T.H., Shang, W., Hassan, A.E., Nasser, M. and Flora, P., 2016, November. CacheOptimizer: Helping developers configure caching frameworks for Hibernate-based database-centric web applications. In Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering (pp. 666-677). ACM.
12. Cherniack, M. and Zdonik, S., 1998, June. Changing the rules: Transformations for rule-based optimizers. In ACM SIGMOD Record (Vol. 27, No. 2, pp. 61-72). ACM.
13. Christodoulakis, Stavros. "Implications of certain assumptions in database performance evaluation." ACM Transactions on Database Systems (TODS) 9.2 (1984): 163-186.
14. Codd, E. F. "Recent Investigations into Relational Data Base Systems". IBM Research Report RJ 1385 (April 23, 1974). Republished in Proc. 1974 Congress (Stockholm, Sweden, 1974). , N.Y.: North-Holland (1974).
15. Codd, E.F. "Further Normalization of the Data Base Relational Model". (Presented at Courant Computer Science Symposia Series 6, "Data Base Systems", New York City, May 24–25, 1971.) IBM Research Report RJ909 (August 31, 1971). Republished in Randall J. Rustin (ed.), Data Base Systems: Courant Computer Science Symposia Series 6. Prentice-Hall, 1972.
16. Davidson, L., Ford, T. and Berry, G., 2010. Performance Tuning Using SQL Server Dynamic Management Views. Simple Talk Pub.
17. Emmett, B. 2017. Simple Talk. [ONLINE] Available at: https://www.simple-talk.com/dotnet/net-tools/entity-framework-performance-and-what-you-can-do-about-it/. [Accessed 11 July 2017].
18. Faloutsos, C. and Kamel, I. Relaxing the uniformity and independence assumptions using the concept of fractal dimensions. Journal of Computer and System Sciences, 55(2):229–240, 1997.
19. Fink, G.. 2017. Microsoft: Select N+1 Problem: How to Decrease Your ORM Performance. [ONLINE] Available at: http://blogs.microsoft.co.il/gilf/2010/08/18/select-n1-problem-how-to-decrease-your-orm-performance/. [Accessed 11 July 2017].
20. Foster, E.C. and Godbole, S., 2016. Review of Trees. In Database Systems (pp. 471-504). Apress.
21. Fritchey, G. and Dam, S., 2013. SQL Server 2012 Query Performance Tuning. Apress.
22. Fritchey, G.. 2017. I Love Entity Framework. [ONLINE] Available at: http://www.scarydba.com/2017/07/05/love-entity-framework/. [Accessed 11 July 2017].
23. Kabra, N. and DeWitt, D.J., 1998, June. Efficient mid-query re-optimization of sub-optimal query execution plans. In ACM SIGMOD Record (Vol. 27, No. 2, pp. 106-117). ACM.
24. Kim, C., Kaldewey, T., Lee, V.W., Sedlar, E., Nguyen, A.D., Satish, N., Chhugani, J., Di Blas, A. and Dubey, P., 2009. Sort vs. Hash revisited: fast Join implementation on modern multi-core CPUs. Proceedings of the VLDB Endowment, 2(2), pp.1378-1389.

25. Lakshmi, M.S. and Yu, P.S., 2000, January. Effect of skew on Join performance in parallel architectures. In Proceedings of the first international symposium on Databases in parallel and distributed systems (pp. 107-120). IEEE Computer Society Press.
26. Leis, V., Boncz, P., Kemper, A. and Neumann, T., 2014, June. Morsel-driven parallelism: a NUMA-aware query evaluation framework for the many-core age. In Proceedings of the 2014 ACM SIGMOD international conference on Management of data (pp. 743-754). ACM.
27. Lu, Y., Shanbhag, A., Jindal, A. and Madden, S., 2017. AdaptDB: adaptive partitioning for distributed Joins. Proceedings of the VLDB Endowment, 10(5), pp.589-600.
28. Masunaga, Y., 2017, January. An intention-based approach to the updatability of views in relational databases. In Proceedings of the 11th International Conference on Ubiquitous Information Management and Communication (p. 13). ACM.
29. Microsoft Corporation. 2015. Performance Considerations (Entity Framework). [ONLINE] Available at: https://msdn.microsoft.com/en-us/library/cc853327(v=vs.110).aspx. [Accessed 29 March 2017].
30. Microsoft Corporation. 2017. Advanced Query Tuning Concepts. [ONLINE] Available at: https://technet.microsoft.com/en-us/library/ms191426(v=sql.105).aspx. [Accessed 29 March 2017].
31. Microsoft Corporation. 2017. SQL Server Index Design Guide. [ONLINE] Available at: https://technet.microsoft.com/en-us/library/jj835095(v=sql.110).aspx. [Accessed 11 July 2017].
32. Mirzadeh, N., Koçberber, Y.O., Falsafi, B. and Grot, B., 2015. Sort vs. hash Join revisited for near-memory execution. In 5th Workshop on Architectures and Systems for Big Data (ASBD 2015) (No. EPFL-TALK-209111).
33. Mishra, P. and Eich, M.H., 1992. Join processing in relational databases. ACM Computing Surveys (CSUR), 24(1), pp.63 113.
34. Oommen, B.J. and Thiyagarajah, M., 2005. Method of generating attribute cardinality maps. U.S. Patent 6,865,567. [ONLINE] Available at: https://www.google.com/patents/US6865567 [Accessed 01 March 2017]
35. Oracle Corporation (2017) Oracle Database Performance Method. [ONLINE] Available at: https://docs.oracle.com/cd/E11882_01/server.112/e10822/tdppt_method.htm#TDPPT006 [Accessed: 01/03/2017].
36. Pane, A., Goldy, N., Madoery, F., Kira, E., Reynares, E. and Caliusco, L., 2017. From Relational to a Column-based Database: A quasi-experiment. Revista Eletrônica Argentina-Brasil de Tecnologias da Informação e da Comunicação, 1(6).
37. Perrizo, W., Ram, P. and Wenberg, D., 1994. Distributed Join processing performance evaluation. In 1994 Proceedings of the Twenty-Seventh Hawaii International Conference on System Sciences.
38. Randal, P. 2015. On index key size, index depth and performance. [ONLINE] Available at: https://www.sqlskills.com/blogs/paul/on-index-key-size-index-depth-and-performance/. [Accessed 28 March 2017].
39. Selinger, P.G., Astrahan, M.M., Chamberlin, D.D., Lorie, R.A. and Price, T.G., 1979, May. Access path selection in a relational database management system. In Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data (pp. 23-34). ACM.
40. Shahvarani, A. and Jacobsen, H.A., 2016, June. A hybrid b+-tree as solution for in-memory indexing on CPU-GPU heterogeneous computing platforms. In Proceedings of the 2016 International Conference on Management of Data (pp. 1523-1538). ACM.
41. Trummer, I. and Koch, C., 2017. Multi-objective parametric query optimization. The VLDB Journal—The International Journal on Very Large Data Bases, 26(1), pp.107-124.
42. Welborne, C.R., de Voogt, D. and Eatough, M., 2016. An analysis of database caching policies. Journal of Computing Sciences in Colleges, 32(2), pp.4-10.
43. Westland, J.C., 1992. Economic incentives for database normalization. Information processing & management, 28(5), pp.647-662.
44. Wu, W., Chi, Y., Zhu, S., Tatemura, J., Hacigümüs, H. and Naughton, J.F., 2013, April. Predicting query execution time: Are optimizer cost models really unusable?. In Data Engineering (ICDE), 2013 IEEE 29th International Conference on (pp. 1081-1092). IEEE.