ADAPTABASE - ADAPTIVE MACHINE LEARNING BASED DATABASE CROSS-TECHNOLOGY SELECTION

Shay Horovitz, Alon Ben-Lavi, Refael Auerbach, Bar Brownshtein, Chen Hamdani and Ortal Yona

School of Computer Science, College of Management Academic Studies

ABSTRACT

As modern applications and systems are growing fast and continuously changing, back-end services in general and database services in particular are being challenged with dynamic loads and differential query behaviour. The traditional best practice of designing database – creating fixed relational schemas prior to deployment - becomes irrelevant. While newer database technologies such as document based and columnar are more flexible, they perform better only under certain conditions that are hard to predict. Frequent manual modifications of database structures and technologies under production require expert skills, increase management costs and often ends up with sub-optimal performance. In this paper we propose AdaptaBase - a solution for performance optimization of database technologies in accordance with application query demands by using machine learning to model application query behavioural patterns and learning the optimal database technology per each behavioural pattern. Experiments present a reduction in query execution time of over 25% for the relational-columnar model selection, and over 30% for the relation-document based model selection.

KEYWORDS

Database, Cross-Technology, Machine Learning, Adaptive

1. INTRODUCTION

Throughout the digital age, efficient mechanisms to store and organize data were always vital [1]. In 1970, Edgar Codd described a new method [19] for storing data, suggesting that records would be stored in tables with fixed length records and based the Relational database model. This initiated the development of new Relational model database management systems (RDBMS). RDBMSs were very efficient in storing and processing structured data and as a result became very popular. Along with the development of the internet, accompanied with demand for greater flexibility, a new type of data started to gain volume rapidly - unstructured data. This type of data is both non-relational and schema-less, which the traditional table-based RDBMS can't manage efficiently. Consequently, alternatives - named as No-SQL databases began to emerge.

With the presentation of new types of databases [22,37], came the industry recognition that different database types are applicable for different conditions; Relational databases fit well for applications that involve many complex queries, transactions and data analysis [8], yet - they suffer from lack of ORM orientation, as they were not originally designed to support OOP principles. Moreover, with a dramatic increase in the size of data, query performance degrades accordingly, which may cause query failures and service crashes due to timeout. Yet, the alternative of No-SQL databases also fails to serve as a one stop shop for database applications, as they come with major concerns [31] such as absence of complete ACID, limited query language, deficient support, and lack of standards. As such, modern application design accommodates multiple database model types [26,14].

Business requirements change frequently [28], hence - bringing changes in organization's data models and database schema respectively; Thus, database performance reduction is expected along time, since the original database models were designed in mind of different assumptions and data is not stored in its optimal structure any longer. For the time being, manual changes are required to overcome this problem, such as changing tables' schema or optimizing indexes - this must be done by database experts and it's a fragile, expensive [7] and complex task, thus commonly avoided. The operational cost of such database changes can be expressed with the following formula:

$$(\sum_{i}^{N} a_i \times b_i) \times x + C + D$$

Expression 1 – operational cost

Where: *N* is the number of available DBAs to work on the current problem; *x* is the problem complexity; a_i is the experience of the DBA and b_i is the estimated work time. *C* is the estimated extra space to store duplicate data and *D* is the data transfer factor.

Due to the above, it would be beneficial to have a system and methods capable of learning the application query behavior, and adaptively fit the optimal database type in accordance with query behavior evolutionary changes, while saving on operational costs. AdaptaBase - an adaptive database model optimizer is a solution for meeting the above challenge. In this paper we focus on typical query behavioral patterns that are dominated by read operations such as SELECT and SELECT JOIN queries as this is the most popular setting [5], and we examine the performance potential and feasibility of an adaptive selection of database model between relational, document-based and columnar models. Adaptabase employs machine learning classification and clustering algorithms in order to map between the characteristic query behavioral patterns or query distributions to the optimal database technology or model type. First, queries are being extracted from the MySQL relational database, then clustered into query types, grouped into query distribution patterns, tested per each pattern and database model, and lastly fits given patterns to the optimal database model and technology taking assuming seasonality of query behavioral patterns.

We tested our proposed solution on a NodeCellar [4] application - built with modern technologies such as Backbone.js, Twitter Bootstrap, Node.js, Express, and MongoDB, and adapted another version of it with MySQL for comparisons. Experiments are twofold: first, we evaluate the performance of our solution on dynamic model selection of Relational and Document based models with MySQL and MongoDB accordingly; Next, we test our solution on dynamic selection of Relational and Columnar models; Our columnar model is represented by lean tables in MySQL rather than Cassandra - which is based on BigTable and Dynamo, enclosing additional technologies side by side with the columnar structure and effect on performance. Cost wise, referring Expression (1), in Adaptabse, a_i, b_i and *x* are 0 since the solution is automatic - saving working hours and training leading to reduced OPEX.

The remainder of this paper examines these issues both analytically and empirically. In Section 2 we discuss related work in this field. Section 3 elaborates on the problem and present different scenarios where query behavior has an influence on performance. Section 4 presents AdaptaBase design and algorithms, and discusses its implementation internals. Section 5 presents our experiments on a real application and last, Section 6 summarizes this work.

2. RELATED WORK

Integration of relational and NoSQL databases has been studied deeply. In [23] a load balancer is used to monitor the performance of a hybrid db detecting hot spots for data migration. [2] tested the ways of integration of relational and NoSQL databases. [29] presented a solution to query MongoDB by SQL language. [33] converted structural to non-structural db. [20] allows migration relational to Document-oriented database. [20] presented approaches to data integration between relational and NoSQL.

The challenge of converting data between SQL and NoSQL databases has been addressed in [24,34,30]. In [30] an autonomous SQL-to-NoSQL schema migration is proposed. [12] seek most suitable NoSQL structure to migrate from relational Database. [35] presented a SQL-to-HBASE data-schema migration. [27] presented RDBMS-to-NoSQL schema and query migration. Hybrid SQL and NoSQL databases are described in [17,32,39]. Performance comparisons for relational and NoSQL can be found in [36,38].

In contrast to the above, our approach adapts alternative columnar and document-based models to a given relational model and dynamically routes the queries to the model that provides the best performance for current query distribution behavioural patterns.

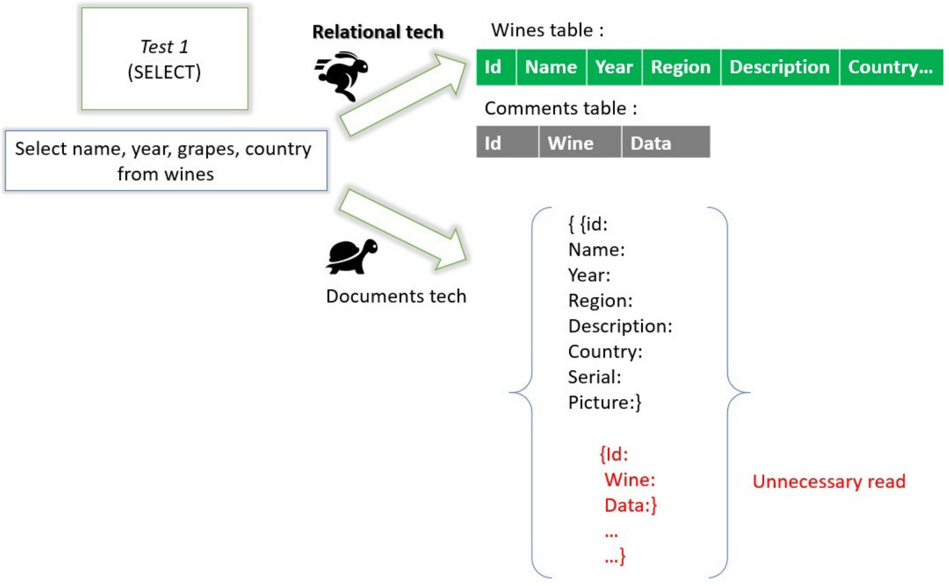
3. PROBLEM

Throughout the process of exploring the benefits and flaws of different database model types, we focused on three types of database models: **Relational** model driven databases are based on storing data in tables - sets of records, each having different attributes. Tables are durable, fast and well suited for transactional operations [25], and the popular SQL language allows a rich and diversified queries, supporting ACID. Yet, relational databases expect fixed predefined schema definition, not tolerant to model changes and are not suitable for dynamic environments with changing query distribution behavioral patterns. In addition, since each row attributes are stored in disk with a continuous form, querying specific attributes is inefficient. **Columnar** databases utilize column oriented model - data is stored and indexed in columns as oppose of rows in the relational model. This allows processing selected columns fast by skipping non relevant attributes that were not requested by the query. While the DBA can partition the relational data in lean tables having small amount of columns - supporting queries that require many columns will end up with subordinate performance due to the need to perform JOIN operations between the lean structured tables. The columnar model is ideal for data analysis applications - suitable for data mining and analytic applications. Columnar databases are not a good fit for transactional workload applications[16]. **Document-based** databases utilize a document model - data is stored in the format of XML or JSON that allows hierarchy and is best suited for schema less, non-structured and non-relational data. While this allows great flexibility, it may be unreliable and index management can be very expensive [31].

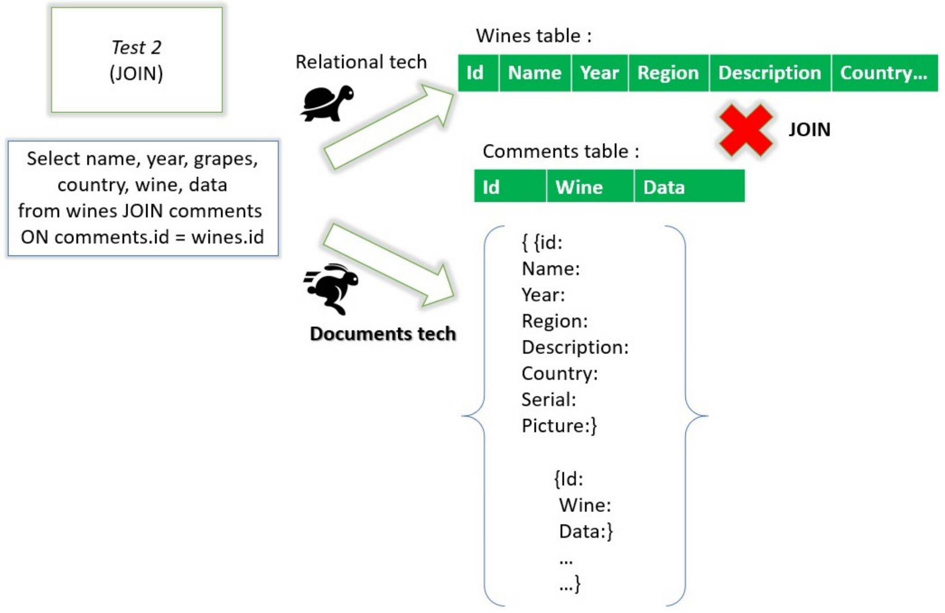
In order to get deeper insights into each database model performance, we executed a set of experiments with different query distributions for Relations vs Document-based and Relational vs Columnar models:

3.1. Relational vs Document-based models comparison

For the relational model we used MySQL and for the Document based model we used MongoDB. The experiments measure execution time of each application instance as a function of distribution of different queries by firing application events using Apache JMeter. All runs are separated by a pause of 10 seconds.



(a) Relational model faster than Document based model scenario



(b) Document-based model faster than Relational model scenario

Figure 1: Relational vs Document-based models performance per query type

In the case of a query asking for data of a single relational table, the relational model in MySQL will end up with faster execution time, whereas the hierarchical representation in MongoDB will be slower due to reading unnecessary data as in Figure 1(a). In contrast, querying data from multiple tables, the relational model requires a JOIN operation ending up with slower execution time compared to the document based model that reads the data that was asked in a single document, as in Figure 1(b).

The experiment depicted in Figure 2 consists of 200 splitted queries, ranging between 0 and 200 SELECT JOIN queries, complemented by INSERT queries. While MySQL performance is heavily dependent on the portion of SELECT JOIN queries, MongoDB in far less affected, presenting 10 times faster execution than MySQL.

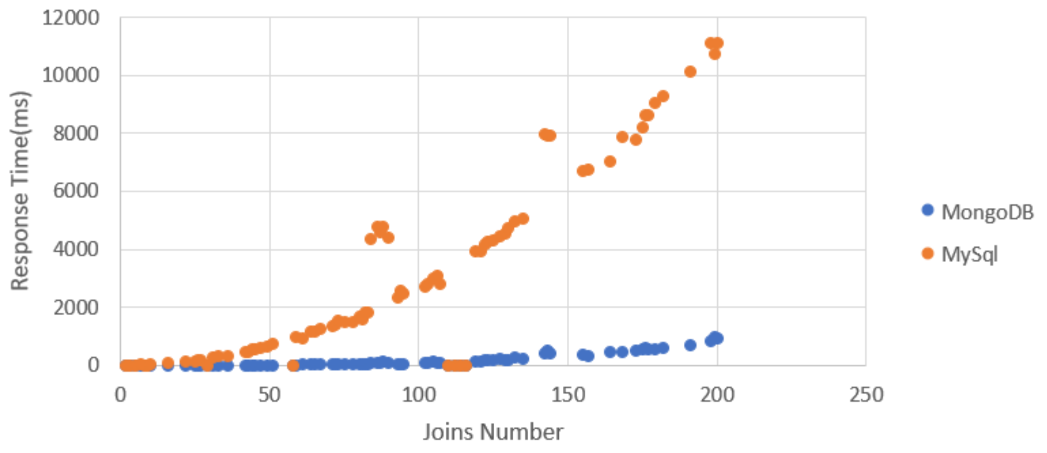


Figure 2: Execution time for distribution of SELECT-JOIN,INSERT

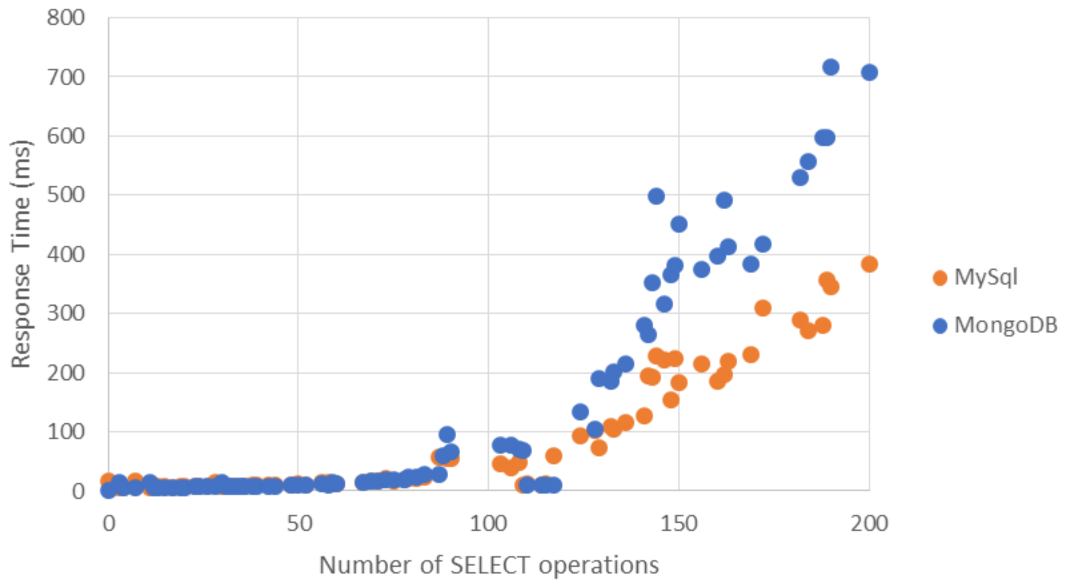


Figure 3: Execution time for distribution of SELECT,INSERT

In the next experiment, depicted in Figure 3, we execute again splitted queries, ranging between 0 and 200 SELECT (no embedded Joins) queries, complemented by INSERTs this time.

While for small ratio of INSERT queries the difference between MongoDB and MySQL is insignificant, in the case of dominant INSERTs, MongoDB performance worsens substantially, with execution time more than double compared to MySQL.

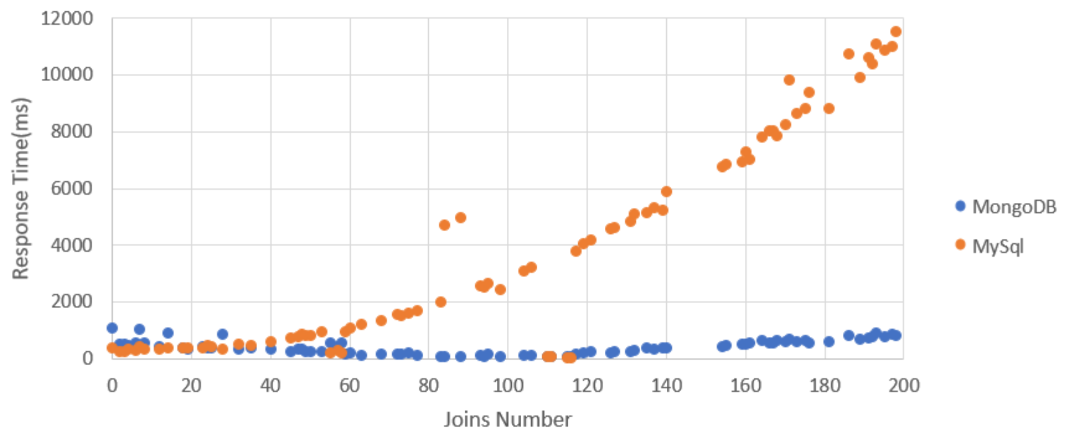


Figure 4: Execution time for distribution of SELECT-JOIN,SELECT

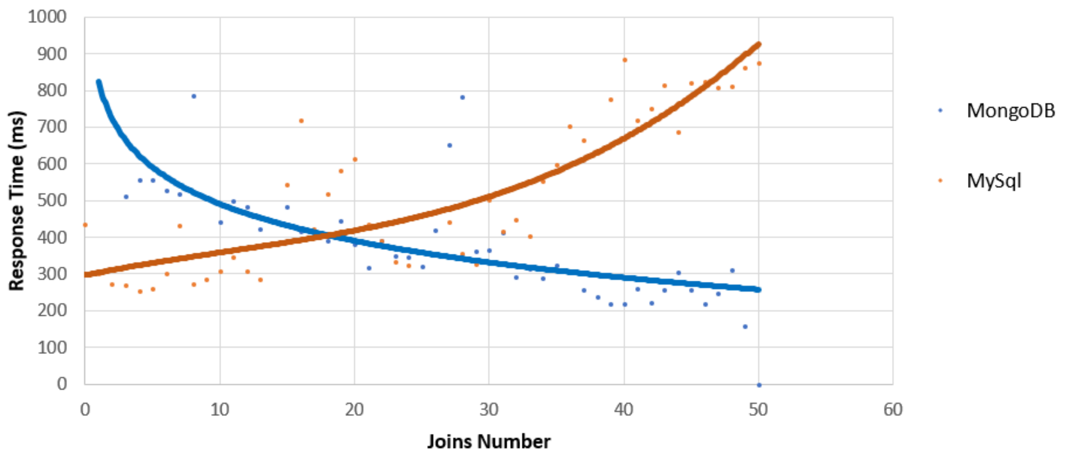
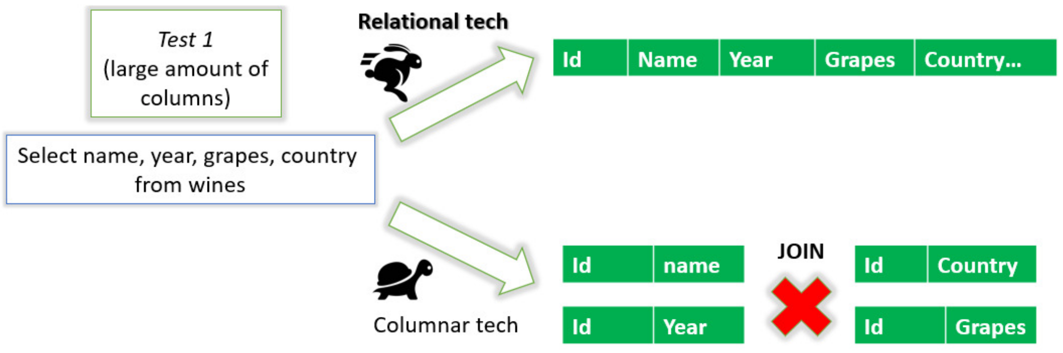


Figure 5: Execution time for distribution of SELECT-JOIN,SELECT (Zoomed on [0-50] Joins)

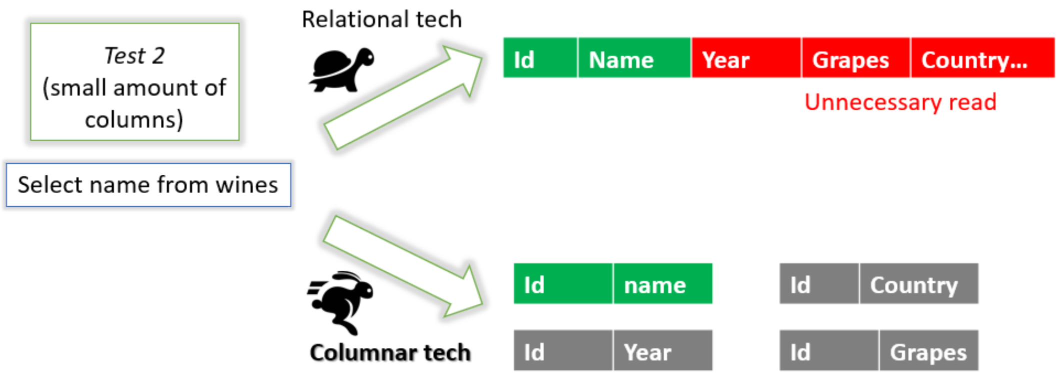
As we focus in read-only queries, we compared simple SELECT with no JOIN and SELECT with JOIN on both databases, as can be seen in Figure 4. While for small proportion of JOINs, MySQL presents better performance than MongoDB, as the proportion increases MongoDB gains extremely better performance, due to JOIN queries being slower than SELECT. A zoom in for infrequent JOINs is in Figure 5, where the cross between the performance of models is visible.

3.2. Relational vs Columnar based models comparison

For the Relational vs Columnar based model comparison, first - we compared SELECT with JOIN and no JOIN queries on both model types in order to estimate the effect of breaking a relational table to lean columnar tables on the performance.



(a) Relational model faster than Columnar model scenario

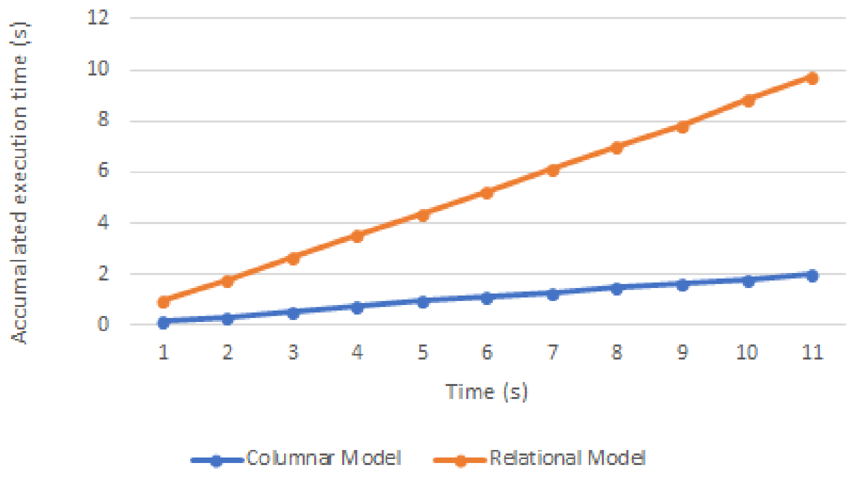


(b) Columnar model faster than Relational model scenario

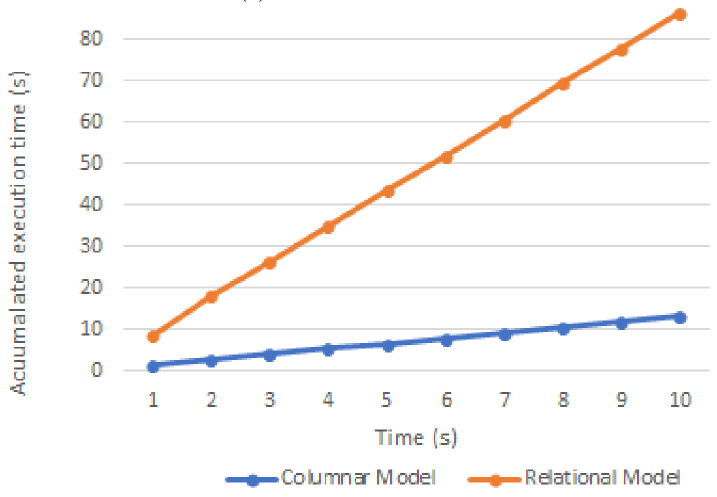
Figure 6: Relational vs Columnar models performance per query type

We used two types of tables - Fat table - that consists all the columns in a single table and Thin tables - breaking the fat table into sub-tables such that JOIN can reconstruct the original table.

As in Figure 6(a), Fat tables are common in relational databases due to representing an entity by a single table. In contrast, in Figure 6(b), thin tables are the best practice of the Columnar approach - where each column is stored separately in the disk. In our experiments, we executed identical queries into the two table types and compared the execution time. Queries that referred to a larger number of columns performed better running times in the fat table than the thin, because no JOIN action were required to join the split columns. In cases where the queries referred to a small number of columns, running times were better in the thin table, because in this case there was no unnecessary reading of information from the disk.



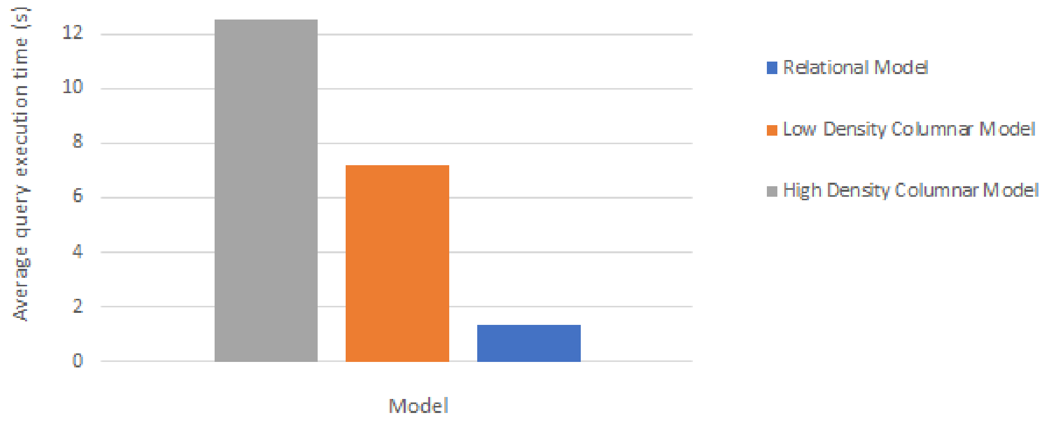
(a) Medium number of rows



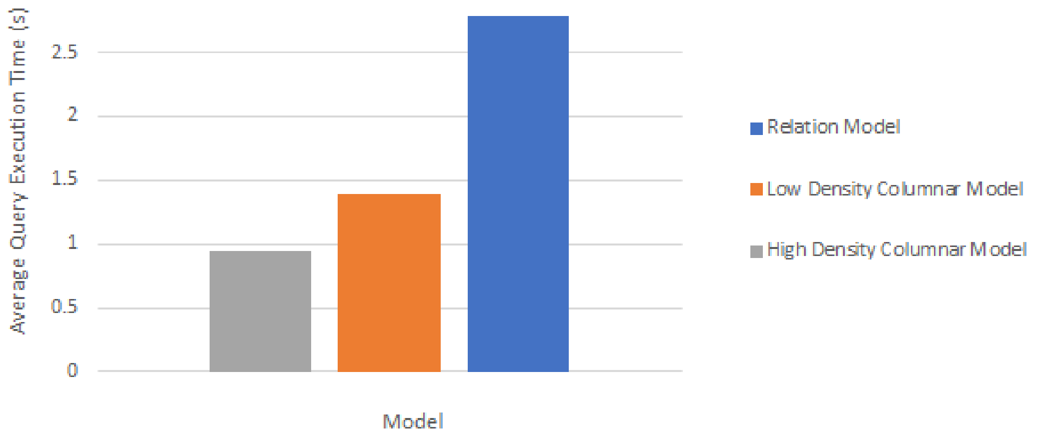
(b) Large number of rows

Figure 7: Relational vs Columnar models accumulated time comparison

In Figure 7 we execute SELECT JOIN queries for several minutes. The columnar model represents a table of 20 columns and the relational model consists of two tables, each of 10 columns - when combined yield the original table. Both tables contain the same amount of rows - 131,072 in case (a) and 1,024,576 in case (b). The columnar model's performance is much better than the relational model as it contains no JOIN. Relational model performance worsens from (a) to (b) up to being 9 times slower compared to the columnar model. The more rows the table consists, the slower query execution times we see.



(a) Large number of columns in query



(b) Small number of columns in query

Figure 8: Average query execution time per model type

In Figure 8 we executed JOIN queries against several tables - having 1,048,576 rows in (a) and 4,194,304 rows in (b). The relational model is represented by a single table; the low density columnar model is represented by 2 tables - that require JOIN in order to return the original table; The high density columnar model consists of 3 tables that require 2 JOIN operations in order to return the original full table. The SELECT queries we run in this case returns the full set of columns as in the original - relational table. We can clearly see in (a) that the more JOINS are apparent in the query, the slower execution time we observe - when querying for large no' of columns using different tables - this is caused due to the JOIN action. Yet, in (b) due to having small no' of columns in the query increases the in efficiency of the relational model.

4. Solution

AdaptaBase provides machine learning based prediction of the optimal database model for given query behavioural patterns - the distribution between query types.

In AdaptaBase, the data & analysis analysis follows the process depicted in Figure 9, and is composed of three phases: first, we learn the query behavioral patterns - the dominant distributions of SQL queries of the application along time; then we test the performance of each query behavioral pattern with each database model type, and last, with the learned mapping of query distribution to database model, match the current query distribution in time, and switch to the optimal database model.

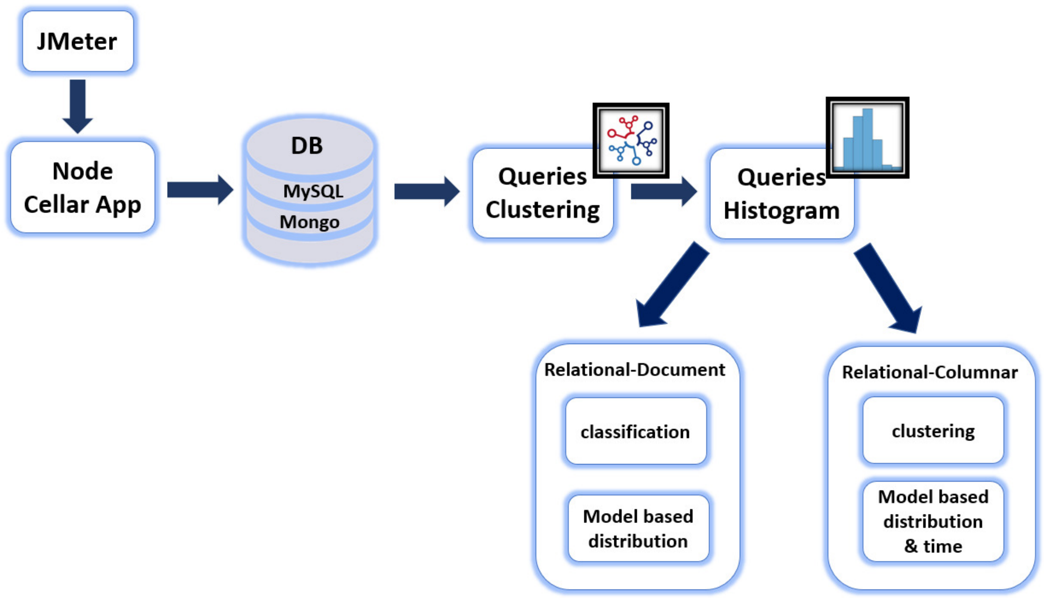


Figure 9: Adaptabase high level data flow

JMeter is used for sending scheduled HTTP requests to Node.JS based NodeCellar application server. The requests follow predefined seasonality patterns. Accordingly, the application server executes different queries against the database; query events are logged and AdaptaBase collects those logs automatically, and stores them for later use by the learning process.

Upon fixed time intervals, the query clustering module is executed, in order to learn about the different query types, and allow us to distinguish between different query distributions or query behavioural patterns in the next phase. SQL query clustering is a well studied issue, and has several solutions, ranging query clustering based on a comparison of query structures, the associated table schemes and statistics such as the sizes of tables that appear in the queries [3], performing query rewrites to standardize query structure [10], using sets of features for query clustering [18], clustering based on attributes for materialized views [13] and clustering based on similarity of the same work plan [21]. Our set of queries in the NodeCellar application was fairly simple and didn’t require a heavy query clustering mechanism; as such we performed the query clustering with as the following: First, each query is converted to a vector. Each word in that query gets a certain index in that vector, and the value in that index is the number of occurrences of the word in that query. Afterwards, the vectors are being clustered using DB-SCAN algorithm. After query clustering is done, we compute the different query distributions (behavioural patterns) over time periods, forming a set of histograms of query types counts and write those distributions to a table. We experimented two separate techniques in order to create a model for predicting the optimal database model type for a given query behavioural pattern:

In the **Relational-Columnar** case, we performed clustering on the table of query distribution counts (histograms), by running random K-Means [9] algorithm to identify the bold behaviors given query distributions and time of day. The algorithm selects the number of clusters with silhouette analysis in order to choose the optimal *k* parameter value with the highest silhouette score. This provides us *k* dominant query distributions. Each distribution is tested against the relational and the columnar models ending up with a mapping of each distribution and its optimal database model.

In the **Relational-Document based** case, first we choose a sample of rows from the table of query distribution counts; then per each sampled distribution, we test the performance of this query distribution on each database technology - MySQL and MongoDB and set the label of the technology that had the minimal execution time as the target for each row in that sample; then we run a classification algorithm (experimented different algorithms) with $k=10$ cross validation on the sampled rows of the table - allowing it to map the relation between each distribution and the optimal database model.

While the learning model has been achieved, upon each time window - a distribution of the actual current queries is computed - and then served for inference by the learning mode - yielding a decision of the predicted optimal database model for that current query distribution.

As for the transformation of queries between the different database models and technologies - In the Relational-Document based cases, for queries migrations there are industrial tools [6]. For data migration it would be possible to use [11,15] .In the Relational-Columnar based experiment, queries transformation isn't needed since we are simply dividing the table into multiple lean tables within MySQL in order to gain a columnar structure.

Since in this work we focus on read-only queries, the price of data transfer between the db technology/model types is not taken into consideration. In order to support cost-efficient transfers between the database types, one may either maintain dual copies of the data - which may be adequate in case where the query behaviour is mainly selection/reads and insertions (which price is insignificant for the DB technologies reviewed in our solution) or when required to cover update/delete operations - the synchronization of data between models may become expensive - here, in addition to the current separation of relational schema into multiple tables, one may learn which of the separated (projected) tables may be efficiently managed with document-db only model.

5. EXPERIMENTS

5.1. Environment settings

For the Relational-Document based model we conducted out tests using VM (2X4, CentOS 3.10) for the server and a separate machine (HP 14bf1xx 16GB RAM, DDR4, 512 GB SSD, Intel Family 6, 2000Mhz) that served as a client. The VM contained a layer of docker (v17.03)- running the containerized application server of NodeCellar application and the databases - MongoDB and MySQL. The client machine contains Apache JMeter testing tool to send HTTP requests that emulate user activity on the NodeCellar application. For the Relational-Columnar based model we used a VM (1X8, Ubuntu 4.4.0) on a server with 4 cores, 23 Ghz, 6GB RAM, 128GB HDD. The document based environment setting is depicted in Figure 10.

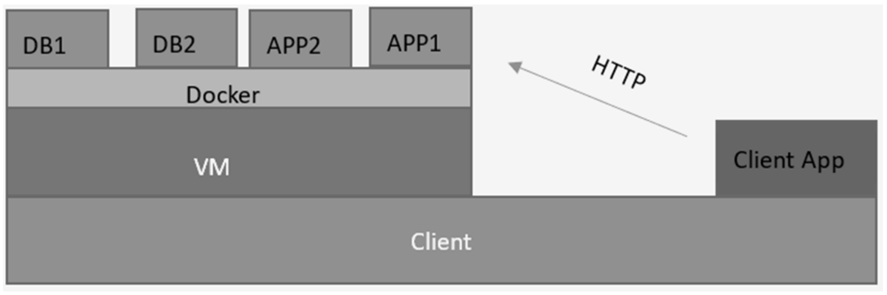


Figure 10: Adaptabase Relational-Document based test environment

5.2. NodeCellar application and queries

For testing we used NodeCellar - a wine collection managing application. For the document-rational experiment three major changes were added to the original application: Trnaslating the original MongoDB based data access layer to MySQL for the columnar case; Adding additional entity of comments. The schemas appear in Figure 11.

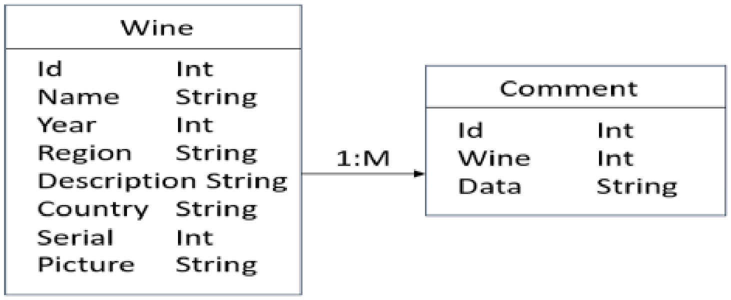


Figure 11: NodeCellar relations

Each entity is created as a single table in MySql. In MongoDB - both were combined into a single collection. In MongoDB this model was implemented using a single collection named Wines. The collection consists a complex document scheme which represents the wine and its comments. The app initializes the database with 1000 wine records and 2983 comment records. In figures 12 the queries used for the tests are described.

App Route	Category	MongoDB query	MySQL query
GET /wines	Select no join	`db.collection('wines').find({},{ _id:0, serial:0 ,comments:0 })`	`SELECT name, year, grapes, region, description, picture, country FROM wine`
GET /wines/comments	Select with join	`db.collection('wines').find({}, { _id:0, name:1, comments:1 },{ })`	`SELECT wine.name, comment.data FROM wine INNER JOIN comment ON comment.WineID=wine.Serial`
POST /wines	Insert	`collection.insert(wine, {safe:true})`	`INSERT INTO wine set ?`

(a) Relational to Document based test queries

App Route	Category	MySQL query
GET /wines	Select no join	Select name0, year0, grapes0, country0, region0, picture0, name1, year1, grapes1, country1, region1, picture1, name2, year2, grapes2, country2, region2, picture2, name3, year3, grapes3, country3, region3, picture3,count(*) from wine_test_1 where year0 = 2009 and country0 = 'Italy' and id < 300000 Group by name0, year0, grapes0,country0, region0, picture0, name1, year1, grapes1, country1, region1, picture1, name2, year2, grapes2, country2, region2, picture2, name3, year3, grapes3, country3, region3, picture3
GET /wines	Select with join	Select t1.name0, t1.year0, t2.grapes0, t2.country0, t2.region0, t2.picture0, t2.name1, t2.year1, t2.grapes1, t2.country1, t2.region1, t2.picture1, t2.name2, t2.year2, t2.grapes2, t2.country2, t2.region2, t2.picture2, t2.name3, t2.year3, grapes3, t2.country3, t2.region3, t2.picture3 from wine_test_111 as t1 inner join wine_test_112 as t2 on t1.id = t2.id where t1.year0 = 2009 and t2.country0 = 'Italy' and t1.id < 300000 Group by t1.name0, t1.year0, t2.grapes0, t2.country0, t2.region0, t2.picture0, t2.name1, t2.year1, t2.grapes1, t2.country1, t2.region1, t2.picture1, t2.name2, t2.year2, t2.grapes2, t2.country2, t2.region2, t2.picture2, t2.name3, t2.year3, grapes3, t2.country3, t2.region3, t2.picture3

(b) Relational to Columnar test queries

Figure 12: NodeCellar application queries used in tests

5.3. Relational-Document based Experiments

In Figure 13, the accuracy of 5 different machine learning algorithms is depicted for the Relational-Document based case. All the algorithms performed well (accuracy of 0.8-0.95), and for each distribution predict which database would be best suited.

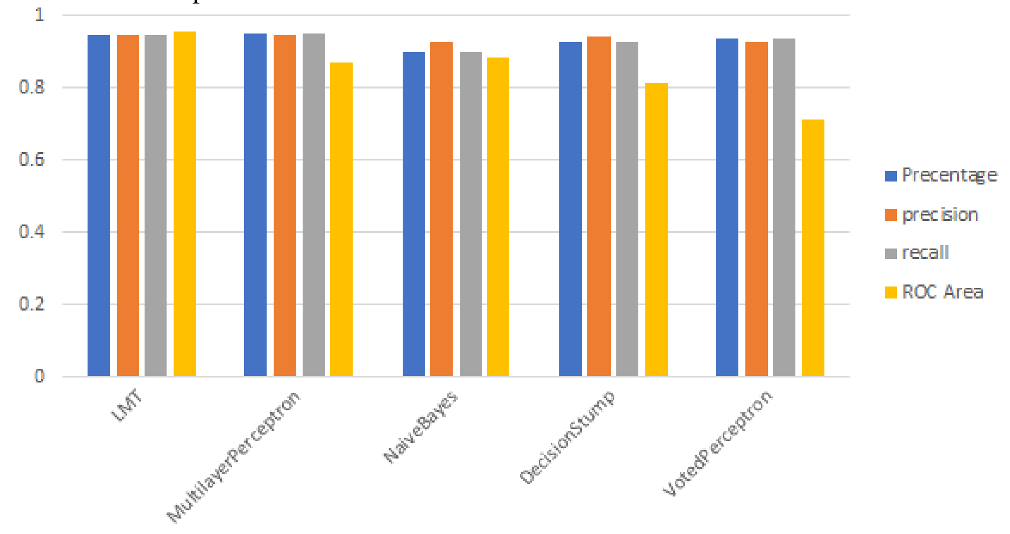
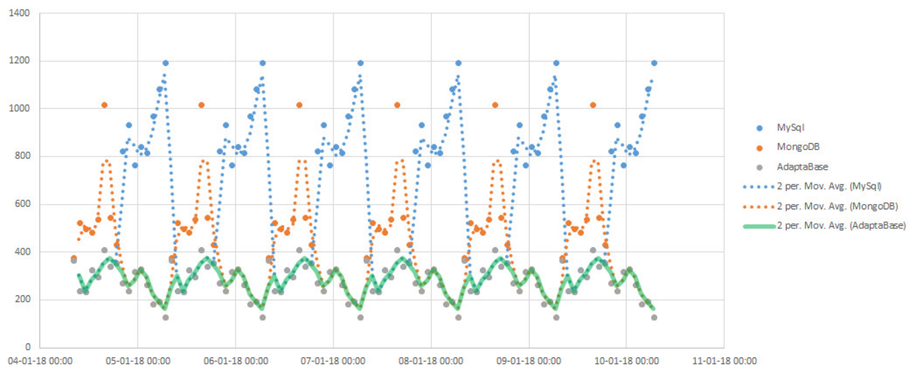


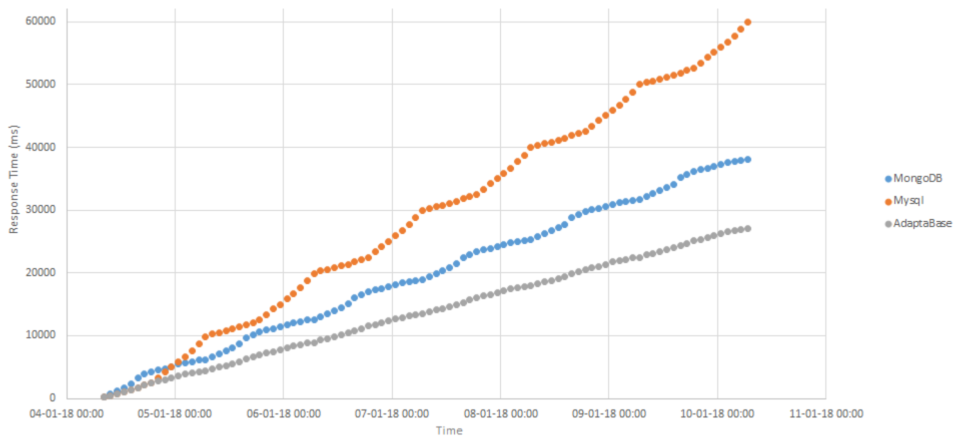
Figure 13: Machine Learning algorithms for Relational-Document prediction

In Figure 14 in (a) the query performance (execution time) was measured during a week of user work. The execution time was measured for each instance and in addition for each distribution the model predicted which database to use. The machine learning algorithm nicely adapts to the optimal database model that provides the minimal query execution times. (b) is the cumulative version - the algorithm performance gains an improvement factor of 1.2-2 compared to alternative predefined database model. Notice that the aggregated execution time of AdaptaBase is significantly shorter than the best aggregated one – which in this case is MongoDB. While

MongoDB performs better than MySQL in this specific scenario over time in general, occasionally MySQL performs better than MongoDB and for those occasions as well, AdaptaBase selects the optimal database.



(a) Regular

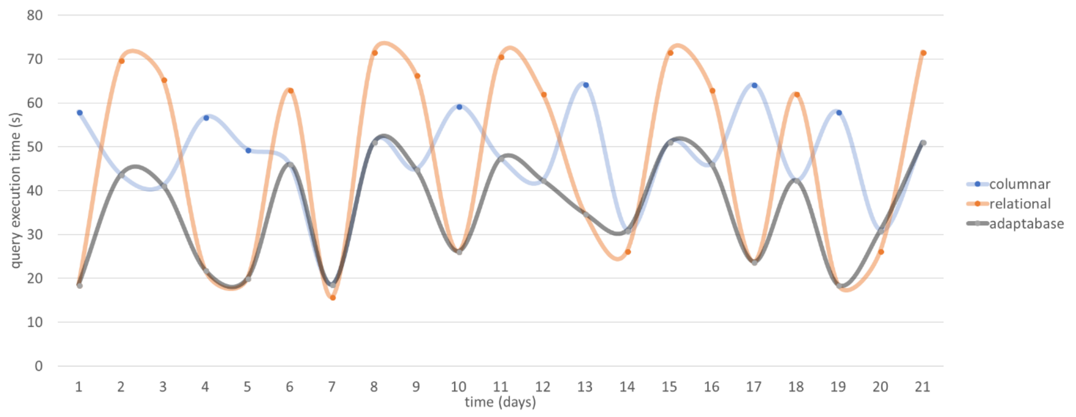


(b) Cumulative

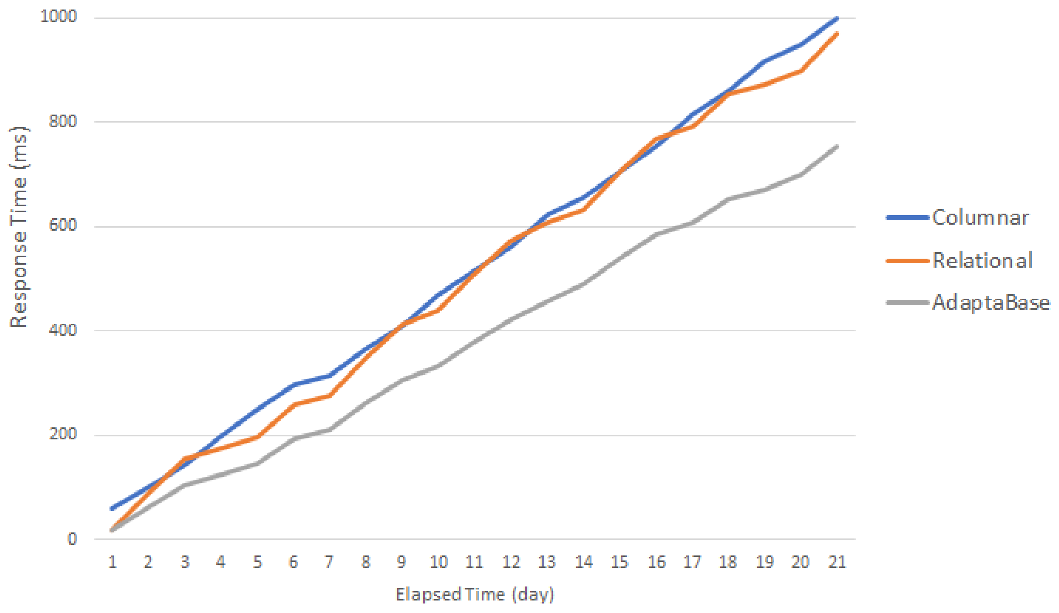
Figure 14: AdaptaBase performance vs alternatives (fixed Relational or Document-based

5.4. Relational-Columnar based Experiments

In Figure 15(a), we compare the performance of the technologies we examined - Columnar and Relational over time - to our machine learning based algorithm. In these experiments, all of the three technologies are tested in parallel. Figure 15(b) presents this experiment in accumulated execution time. In total, our solution achieves improved performance of over 25% in the period of 21 days.



(a) Regular



(b) Cumulative

Figure 15: AdaptaBase performance vs alternatives (fixed Relational or Columnar)

6. Summary

In this paper we have presented AdaptaBase - a solution that can reduce query execution times and eventually save on OPEX. Our solution is based on machine learning based prediction of the optimal db model for a given query behavioural patterns.

Our experiments based on actual query execution on real DB systems- i.e. MySql and MongoDB - presented a reduction in query execution time of 25% for the relational-columnar model selection, and up to 30% for the relation-document based model selection.

Next, we intend to evaluate modifying commands such as INSERT, UPDATE, DELETE and extend our experiments to other database types such as graph and key-value databases.

REFERENCES

[1] A Brief History of Database Management.
http://www.dataversity.net/brief-historydatabase-management.

[2] Bridging Relational and NoSQL Worlds: Case Study.
https://www.igiglobal.com/chapter/bridging-relational-and-nosql-worlds/191986.

[3] Efficient Query Recommendation.
http://www.cs.technion.ac.il/users/wwwb/cgi-bin/trget.cgi/- 2015/MSC/MSC-2015-14.pdf.

[4] Node Cellar. http://nodecellar.coenraets.org.

[5] On workload characterization of relational database environments.
http://ieeexplore.ieee.org/- abstract/document/129222/.

[6] Query Translator. http://www.querymongo.com.

[7] Relational Databases Are Not Designed To Handle Change https://www.marklogic.com/-blog/relational-databases-change.

[8] Relational vs. non-relational databases: Which one is right for you? https://www.pluralsight -.com/blog/ software-development/relational-non-relationaldatabases.

[9] Selecting the number of clusters with silhouette analysis on KMeans clustering.
http ://scikit − learn.org/stable/autoexamples/cluster/plotkmeanssilhouetteanalysis.html.

[10] Similarity Metrics for SQL Query Clustering .
https://odin.cse.buffalo.edu/papers/2018- /TKDEQuerySimilarity.pdf.

[11] Warehouse. https://github.com/dundalek/warehouse.

[12] Al Shekh Yassin, F.J.: Migrating from sql to nosql database: Practices and analysis (2017).

[13] Aouiche, K., Jouve, P.E., Darmont, J.: Clustering-based materialized view selection in data warehouses. In: East European Conference on Advances in Databases and Information Systems. pp. 81–95. Springer (2006).

[14] Arnold, J., Glavic, B., Raicu, I.: Hrdbms: Combining the best of modern and traditional relational databases. Illinois Institute of Technology, Department of Computer Science, PhD Oral Qualifier (2015).

[15] Arora, R., Aggarwal, R.R.: An algorithm for transformation of data from mysql to nosql (mongodb). International Journal of Advanced Studies in Computer Science and Engineering 2(1) (2013).

[16] Bhatia, A., Patil, S.: Column oriented dbms an approach. International Journal of Computer -Science & Communication Networks 1(2), 111–116 (2011).

[17] Bjeladinovic, S.: A fresh approach for hybrid sql/nosql database design based on data structuredness. Enterprise Information Systems pp. 1–19 (2018).

[18] Chu, W.W., Zhang, G.: Associative query answering via query feature similarity. In: Intelligent Information Systems, 1997. IIS'97. Proceedings. pp. 405–409. IEEE (1997).

[19] Codd, E.F.: A relational model of data for large shared data banks. Communications of the ACM 13(6), 377–387 (1970).

[20] El Alami, A., Bahaj, M.: Migration of a relational databases to nosql: The way forward. In: Multimedia Computing and Systems (ICMCS), 2016 5th International Conference on.pp. 18–23. IEEE (2016).

[21] Ghosh, A., Parikh, J., Sengar, V.S., Haritsa, J.R.: Plan selection based on query clustering. In: VLDB'02: Proceedings of the 28th International Conference on Very Large Databases.pp. 179–190. Elsevier (2002).

[22] Han, J., Haihong, E., Le, G., Du, J.: Survey on nosql database. In: Pervasive computing and applications (ICPCA), 2011 6th international conference on. pp. 363–366. IEEE (2011).

[23] Huang, H.S., Hung, S.H., Yeh, C.W.: Load balancing for hybrid nosql database management systems. In: Proceedings of the 2015 Conference on research in adaptive and convergent systems. pp. 80–85. ACM (2015).

[24] ISLAM, M.S.: Techniques for converting big data from sql to nosql databases.

[25] Kemper, A., Neumann, T.: Hyper: A hybrid oltp&olap main memory database system based on virtual memory snapshots. In: Data Engineering (ICDE), 2011 IEEE 27th International Conference on. pp. 195–206. IEEE (2011).

[26] Ko, C.Y.: Three approaches to a multidatabase system. In: Proceedings of the Philippine Computer Science Congress (PCSC), www. citeseer. ist. psu. edu/ko00three. html (2000).

[27] Kuderu, N., Kumari, V.: Relational database to nosql conversion by schema migration and mapping. International Journal 3(9), 506–513 (2016).

[28] Law, J., Rothermel, G.: Whole program path-based dynamic impact analysis. In: Proceedings of the 25th International Conference on Software Engineering. pp. 308–318. IEEE Computer Society (2003).

[29] Lawrence, R.: Integration and virtualization of relational sql and nosql systems including mysql and mongodb. In: Computational Science and Computational Intelligence (CSCI), 2014 International Conference on. vol. 1, pp. 285–290. IEEE (2014).

[30] Lee, C.H., Zheng, Y.L.: Sql-to-nosql schema denormalization and migration: a study on content management systems. In: Systems, Man, and Cybernetics (SMC), 2015 IEEE International Conference on. pp. 2022–2026. IEEE (2015).

[31] Nayak, A., Poriya, A., Poojary, D.: Type of nosql databases and its comparison with relational databases. International Journal of Applied Information Systems 5(4), 16–19 (2013).

[32] Okeke, K.K., Ejiofor, V.E.: Implementation of cross-platform language between sql and nosql database systems. In: OcRI. pp. 239–240 (2016).

[33] Potey, M., Digrase, M., Deshmukh, G., Nerkar, M.: Database migration from structured database to non-structured database. In: International Conference on Recent Trends & Advancements in Engineering Technology (ICRTAET 2015). pp. 1–3. Citeseer (2015).

[34] Schreiner, G.A., Duarte, D., dos Santos Mello, R.: Sqltokeynosql: a layer for relational to key-based nosql database mapping. In: Proceedings of the 17th International Conference on Information Integration and Web-based Applications & Services. p. 74. ACM (2015).

[35] Serrano, D., Stroulia, E.: From relations to multi-dimensional maps: a sql-to-hbase transformation methodology. In: Proceedings of the 26th Annual International Conference on Computer Science and Software Engineering. pp. 156–165. IBM Corp. (2016).

[36] Srividyaa, S., Varalakshmi, R.: A study on output performance of nosql data processing over rdbms in big data.

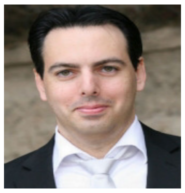
[37] Strauch, C., Sites, U.L.S., Kriha, W.: Nosql databases. Lecture Notes, Stuttgart Media University 20 (2011).

[38] Wu, C.M., Huang, Y.F., Lee, J.: Comparisons between mongodb and ms-sql databases on the twc website. American Journal of Software Engineering and Applications 4(2), 35–41 (2015).

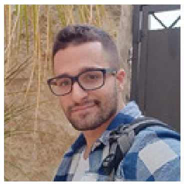
[39] Wu, H., Ambavane, A., Mukherjee, S., Mao, S.: A coherent healthcare system with rdbms,nosql and gis databases (2017).

AUTHORS

Dr. Shay Horovitz, PhD is head of Data Science specialization at the Computer Science School at the College of Management and a senior researcher & expert at Huawei. His research area is Machine Learning algorithms for the cloud, large scale networks and big data.



Alon Ben-Lavi, graduated B.Sc. at the College of Management - Academic Studies. His research focus on the effects of database models on applications performance.



Refael Auerbach, web solutionist and Big Data expert. A software engineer with experience over a decade in web development, distributed computing and machine learning. B.Sc in Computer Science.

Bar Brownshtein, owns Bs.c in computer science at Israel college of management - academic studies. Specializes in data science. Works at Indusify as software developer.

Chen Hamdani, holds a bachelor's degree in computer science with a specialization in data science at the College of Management. Works as a developer in the Prime Minister's Office.

Ortal Yona, Graduated Bsc in computer science with specialization in data science from the college of management academic studies in Israel.