Hindawi
Scientific Programming
Volume 2022, Article ID 4301944, 11 pages
https://doi.org/10.1155/2022/4301944

Research Article

Translytics: A Novel Approach for Runtime Selection of Database Layout Based on User's Context

Muhammad Makhshif Tanvir,[1] Muhammad Khuram Shahzad,[1] Muhammad Anwar,[2] and Su Man Nam[3]

[1]*Department of Computing, National University of Sciences and Technology, Islamabad 46000, Pakistan*
[2]*Department of Information Science, Division of Science and Technology, University of Education, Lahore 54000, Pakistan*
[3]*Dudu Information Technologies, Inc., Seoul, Republic of Korea*

Correspondence should be addressed to Su Man Nam; sumannam@gmail.com

Received 29 April 2022; Revised 15 June 2022; Accepted 20 June 2022; Published 10 August 2022

Academic Editor: Sikandar Ali

Currently, organizations have to maintain separate systems for transactions and analytics inside the company. Notably, different vendors provide the capabilities for either of these tasks that require specialized hardware or software. Data engineers are required to retrieve data from one source and transform it into another format to obtain the maximum benefit in a minimum time. Organizations strive for a competitive advantage that is achieved by fetching data from their customers and getting insights earliest for timely decision making. Present practices do not permit the view of the latest data for analytics since the first data have to be fetched from the source, transformed, and loaded to other systems to be utilized for analysis by relevant teams. This paper introduces a single system for both transactions and analytics. Our proposed solution would permit companies to seamlessly adapt our solution without the need to shift all of their data to newer systems and allow all the teams. It would grant all the teams to have a view of the latest available data without extra expertise and budget.

1. Introduction

In this paper, we aim at countering the limitations arising from maintaining separate systems for transactions and analytics in organizations. In today's modern world, organizations have to deal with a lot of data. These data can be generated either from their customers or within the organization. Other than generation of data, this is equally important to analyze these data to derive the hidden facts inside it. This is becoming increasingly difficult to handle these data in a way so the transactions and analysis can be run simultaneously on the same data and the same systems. If we can have a system where we can run both types of workload on the same system, it can save us from the overhead of maintaining two different systems, keeping data at 2 places, and waiting for a whole day for data to be available for analysis.

As per the present practices, corporates deploy essentially 2 different types of databases. One is for transactional processing and another is for analytical processing. The main or first data store is usually the OLTP database because the data are first inserted into the database and any sort of transaction whether insert, update, or delete performs better in row stores. Once the data from the main source are inserted in the database, the data are transferred to the OLAP databases with specialized jobs which usually run in off-peak to avoid any workload on main databases. These data are then accessed by the teams who have to perform analysis on these data. There are a few problems in this setup. Foremost is that the analysis teams do not have the latest view of data and as per usual practice they have data till 'sysdate-1.' Secondly, the same data are residing in 2 different databases which impose the hardware and support burden on companies. So, there should be a way in which the analytics team

can have a view of the latest available data. Here, we need to have different databases for transactions and analytics. Ideally, we should have the same database where users can perform transactions and analyses simultaneously.

The actual reason for this segregation is the format of the data inside the database storage. In transactional databases or OLTP systems, the data are stored in row format while in analytical databases or OLAP systems, the data are stored in column format. This is not possible to natively co-locate both formats in the database as data can either be stored in row format or column format. Some years ago the concept of HTAP was introduced. It stands for Hybrid Transactional Analytical Processing. Some vendors and researchers stepped forward and offered some of their solutions. In a few of them, the analysis could be run on the latest data while in others the users had to wait for some time so the view of the latest data could be prepared for them within the database. In the past, relational databases have been used for the transaction as well as analytical processing. However, OLTP and OLAP systems have very different characteristics. Traditionally, we have known OLTP systems by their capabilities of individual record insert, delete, or update statements and also the point queries that take advantage of indexes. On the other side, we update the OLAP systems in batches and they usually require table scans. ETL (extract transform load) systems perform batch insertion into OLAP systems which fetch and transform transactional data from OLTP environments and import it into OLAP environments for users to perform analysis on the data.

The term HTAP (hybrid transactional analytical processing) is used for the systems that support both OLTP and OLAP queries. Some of these systems can execute analytical queries on the latest data, and some of them need some time before the queries can have a view of the latest data. OLAP and OLTP systems have somehow progressed over time. OLTP and OLAP systems are different from each other; although the purpose is to provide data to the user, they have different use cases. OLTP systems usually have the data arranged row format which better supports large transactional queries, e.g., insert/update/delete while OLAP systems are usually arranged in column format which better supports the large analytical queries like table scans and selection in batches. ETL processes do the batch insertions in OLAP systems, and they consolidate and transform the transactional data from OLTP systems into an OLAP environment for analysis. This is because the source systems usually have the databases configured to better support the transactions.

We have kept in mind that organizations do not need to shift all of their data from their operational databases to a whole new infrastructure; this would rather be closer to a plug-and-play proposal. They will not need to get the training from scratch or hire experts. This will allow the users to eliminate the need for having two different and loosely coupled systems for each type of need. This will also lessen the burden of maintaining the different databases by capitalizing on the utilities of a single database. All the organizations that require regular transactions and analysis of the data for their decisions can benefit from this research. These organizations may include telecommunication companies, banks, financing companies, retail, and so on.

2. Related Work

We have seen some specialized systems in the last few years such as BLU [1], Vertica [2], ParAccel, GreenPlumDB [3], Vectorvise, as well as many in-memory OLTP systems, including VoltDB [4], Hekaton [5], and MemSQL [6]. The abovementioned systems take good advantage of multicore, multilevels of memory caches and a huge amount of memories. Recently we have also seen an outburst of many big data technologies, being used by new generation applications. NoSQL/key-value stores like Voldemort [7], Cassandra [8], and RocksDB [9] support fast inserts and lookups and good capabilities to scale out, but they lack in some query capabilities and do not offer complete transactional guarantees. There have been also many SQL-on-Hadoop [10] offerings, including Hive [11], Big SQL [12], Impala [13], and Spark SQL [14], which provide analytical capabilities on huge data sets, and they focus on OLAP queries only hence lacking transaction support. These systems started as traditional databases where they developed leaner database engines from scratch. Almost all of these started support for one type of processing and later on started adding support for the other type of processing as well.

These systems are mainly different from each other based on the data organization they use for either transactional or analytical requests. There was a project named HyPer [15], that was designed to support both fast transactions and analytics using one engine. In the beginning, it was using row-based processing of data for both OLTP and OLAP, but now it also provides the option for choosing a columnar format so it can run the analytical requests also more efficiently. Recently, there has been another project named Peloton [16] which aims to build an autonomous database system. It provides data the capability to change its format at run-time based on the type of requests. Then, there is another subsequent research [17] that aims to forecast the query workloads that the format of the storage can be changed on the fly. They have also done some research for bridging the archipelago between row stores and column stores [18] so it can be decided which data have to be kept in row format and which in column format at which particular time. Almost all of these systems require converting the data from one format to another before it can be visible to users requiring the other format.

One issue in each of these proposed solutions is that each of them has been built from scratch and none of these is the plug-and-play module so organizations can plug these into their existing architecture so they do not have to adopt a major technology shifting. In our study, we shall focus on a system that should allow the organizations to cater to the needs of each type of the user from a single database where users can do their analysis of the latest data by keeping the level of trade-offs as low as it can get.

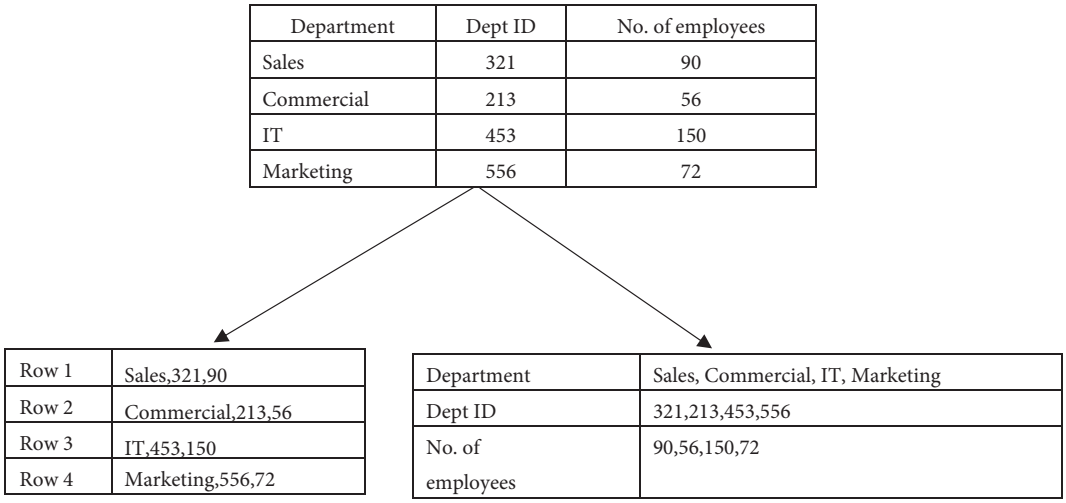

Figure 1: Row format vs. column format.

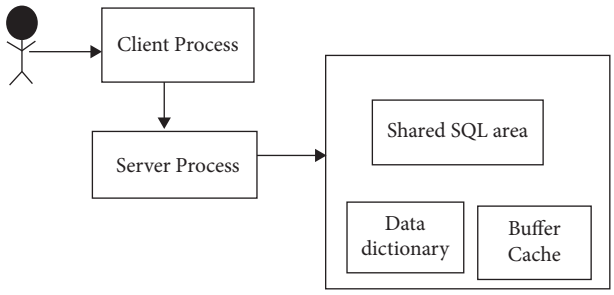

Figure 2: User connection with database.

3. Row Format vs. Column Format

Conceptually, a relational database table is a two-dimensional data structure with cells organized in rows and columns. For storing a table in linear memory, two options can be chosen, i.e., row store and column store. A row store stores a sequence of records that contains the fields of one row in the table as we can see in Figure 1. In a column store, the entries of a column are stored in contiguous memory locations as we can see in Figure 1 again. Basically, row stores are good for transactional processing while column stores are good for highly analytical query models. Row stores have the ability to write data very quickly, whereas a column store supports better for aggregating large volumes of data for a subset of columns.

4. Data Access Method

Before coming to our main strategy, let us discuss how a query runs inside a database engine so that we may know which part of the overall architecture needs to be focused on for our implementation. Figure 2 shows the high-level view of the architecture of relational databases. This is the high-level view of the architecture of relational databases. Given one is of oracle but relational databases more or less use the same concept. Let us explain this briefly. The database user connects to the database through a client where a specified amount of session memory is allocated to its server process. The query is then directed to the database instance where the query is parsed and consequently data are fetched. For every query which user issues, the data are first searched inside the database memory, and if the data are present in it, the data are returned to the user but if the data are not present in memory, the data are first brought into the memory from databases files residing on the disk and then the data are returned to the user. This is true for every computational process that data are first searched inside the RAM or memory; if it is not found in it, then the data are fetched from the disk. But keep in mind, the memory of the database is not the same as the data of the server on which the database is running. A separate chunk of memory has to be allocated to the database instance; this is an educated chunk of memory where database-specific parsing and data manipulation take place.

Now, let us look a little deeper inside a portion of architecture in Figure 3 where we will be more focused in our study, the memory of the database. This is the area where SQL parsing takes place. We can look at the high-level view In Figure 3. When a user issues an SQL query, it is first parsed. Parsing consists of three parts, i.e., syntax checking, semantics checking, and shared pool checking. In syntax checking, the database engine checks if the syntax of the query is correct. If the syntax is correct, it then checks for the semantics of the query. For example, if a user issues a query where 3 different columns are required to be checked, this step of semantics checking will check if the required table is present in the database; if the required columns are present in that table; and so on. After this step, the privileges are also checked where the database checks if the user has the required permissions and privileges to access these data. If these checks are fine, then a hash value of the given SQL command is generated and stored in the library cache. But before storing the hash in the library cache, it will first check if the same hash is already present inside the library cache or not. If the hash is already present. then there is no need to store it again; otherwise, we store it. If the hash is already present, this saves us from some subsequent efforts which we shall discuss. This step is called shared pool checking. If the

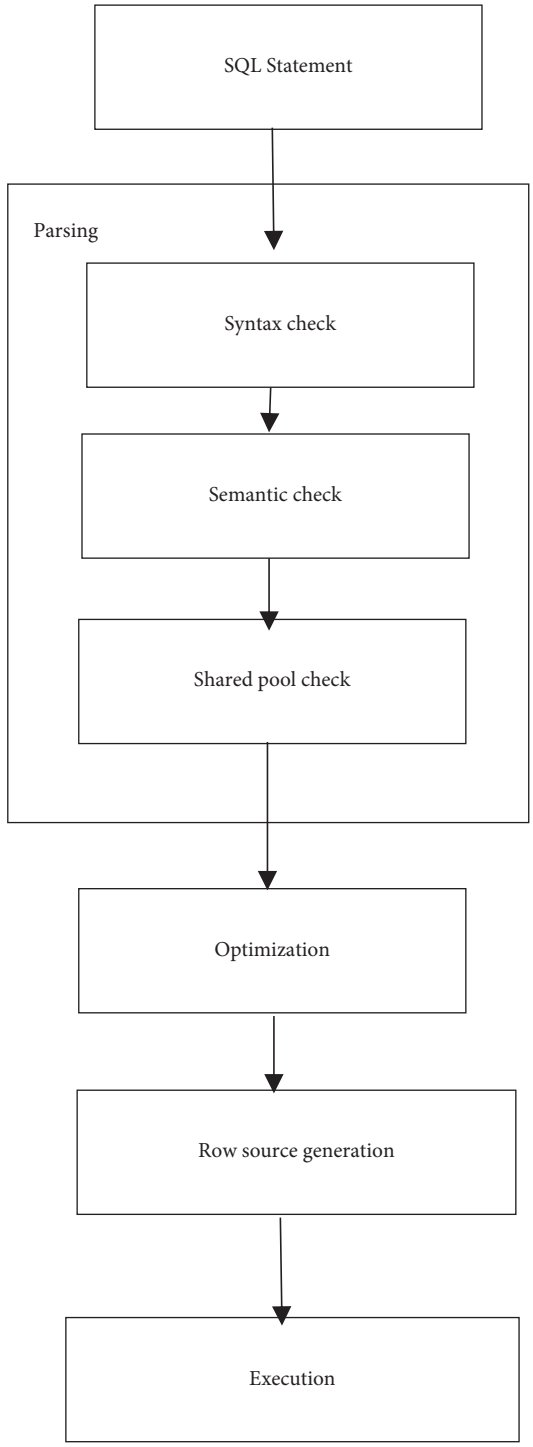

Figure 3: SQL statement lifecycle.

query hash is found to be already present in the library cache, this situation is a soft parse and the database engine goes directly to the execution of the query. If the hash is not found and has to be stored, this is called hard parse where the optimizer does its work and generated different execution plans of this query and chooses the best one out of these. Row source is then generated, the query is executed, and the result is returned to the user.

As we discussed, if the hash is not found in the library cache, this is called hard parse. This is the stage where the query goes through the whole parsing cycle. Syntax of the query is checked whether the query is in DB understandable form, semantics are checked whether the requested data are present in a database or not, if the user has permissions to access these data, and so on. Then, the optimizer does its work and generates different execution plans for this query and chooses the best one out of these. Row source is then generated, the query is executed, and the result is returned to the user.

If each new SQL statement is being parsed, this means each new SQL is a new entrant and has not been previously used or the plan has timed out. This might also mean that the shared pool is not large enough to keep the regular SQL plans. Frequent hard parsing does not always point to the problems as we can consider the possibility that each SQL is a new one but this is an extremely optimistic approach as this is a rare scenario. The hard parsing slows down the performance because we have to then frequently access the hard disk which itself is a costly process. This is the place where the DBAs of the company have to be engaged to look into the matter. The reasons for hard parsing might include the queries have not been standardized, the host variables are not being used, the size of the shared pool is very small, and regular queries have not been documented or controlled somewhere because only changing the case of the SQL statement, i.e., changing from upper case to lower case and vice versa will generate the different hash values. The reason has to be analyzed and rectified accordingly, e.g., we can plan to take care that the host variables are used in SQL so these can be treated as the re-entrants. The lesser the hard parsing the better the performance be but each SQL will be parsed at least once in the starting.

If the query hash is found to be already present in the library cache, this situation is a soft parse and the database engine goes directly to the execution of the query. If the query has already been parsed and its hash value is present in the library cache, we will not have to reparse it and we will be saved from a lot of overhead as parsing itself is a costly process as the purpose of the shared pool is to maximize the sharing and minimizing the repetitive tasks by reusing the already available information. Soft parsing might mean that the entrant SQL was as it is executed before or the only unique feature of this new entrant is its host variables. In this case, the hash value will remain the same and the input values of variables are changed. So, the size of the library cache has to be kept as much which can hold the hash values and execution plans of at least those queries which are regularly and frequently being accessed.

5. Proposed Design

Now comes the main topic of discussion. We will focus on how we can keep the 2 layouts, i.e., row store and column store in one database. As discussed earlier, while creating the database, we may have options whether to adopt a row store or column store but not both, and if both have to be done, we

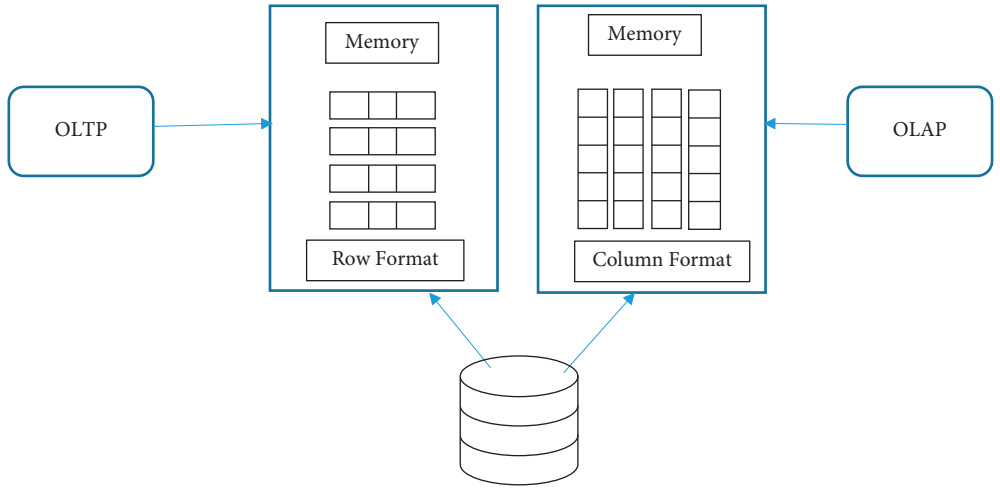

Figure 4: High-level concept overview.

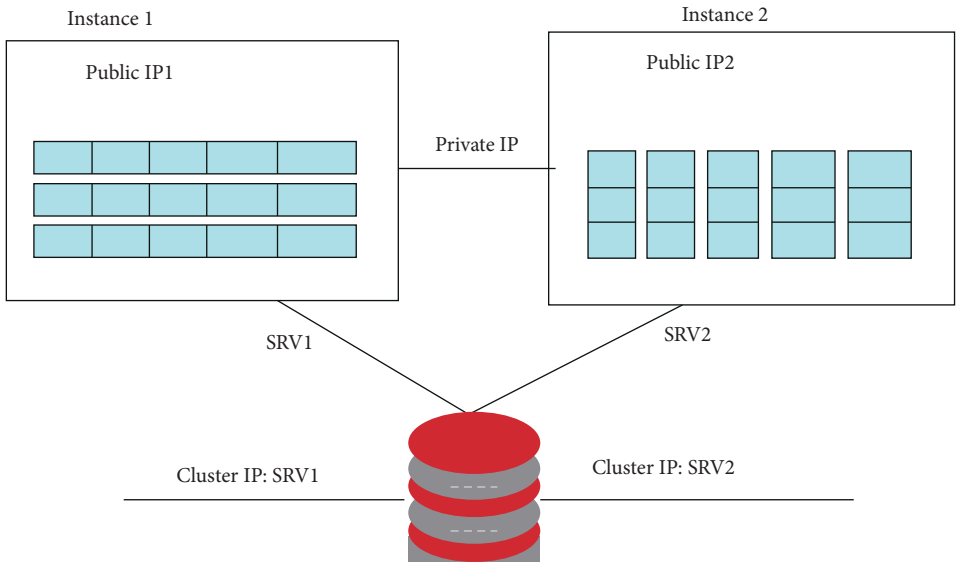

Figure 5: Segregation of users.

have to have 2 different databases and replicate data between both of these.

Practically speaking, the data analytics teams are working on a specific chunk of data on daily basis and they do not require the whole data at one time. So, instead of trying to co-locate row and column format on the disk, what if we try to manage it inside of the memory of the database. We can try to keep the database the same but define 2 separate chunks of memory for it. One where the data are in row format, and the other where data are in column format. The transactional queries can be redirected to the former and analytical queries to the latter. But a new challenge comes to the surface that how can we do this so that the underlying database remains the same and we can plug another memory set along with the default one. We shall discuss this in our next portion.

5.1. Database Cluster Configuration. Our main considerations are we have to keep the storage the same so we do not have to have the burden of extra storage for the replica database, all types of users should be able to work on the same database, the required tables, views, or partitions should have the most recently updated data, and the database should be able to comprehend the context of the query and redirect it to the suitable chunk of memory from where its needs will be catered for. Most importantly, this proposed design should not be in conflict to presently working database and its working should be seamless. We can make a database cluster which has the same storage but has multiple instances running on it as we can see in the high-level diagram in Figure 4.

So, in the proposed database cluster, the instances are 2 servers or 2 virtual machines as we can see in Figure 5. These can be running on any operating system whether it is Linux, Windows, UNIX, and so on. These instances or nodes of databases will have their own memory and separate CPUs but storage is shared between both of the instances. Storage can be provisioned through SAN or by making shared mount points or disks. Both of these instances communicate

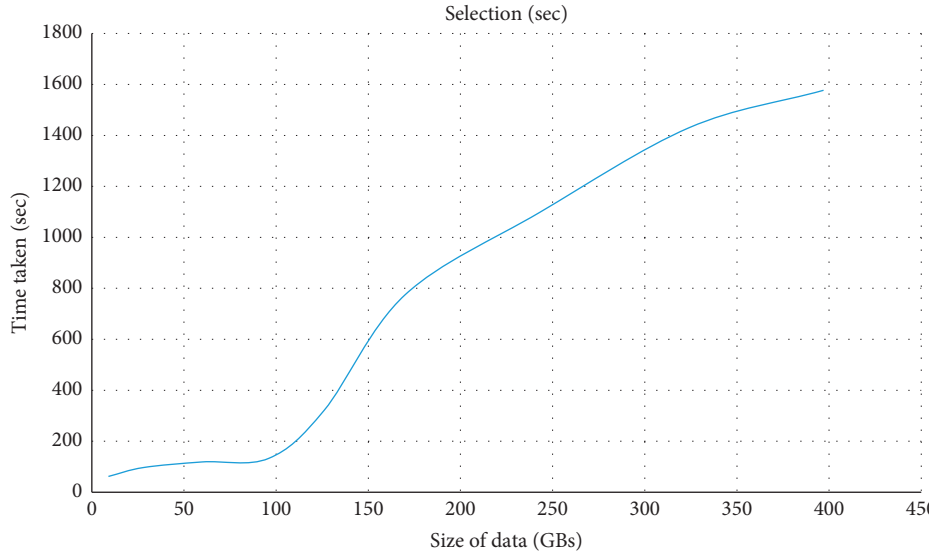

Figure 6: Selection in row format.

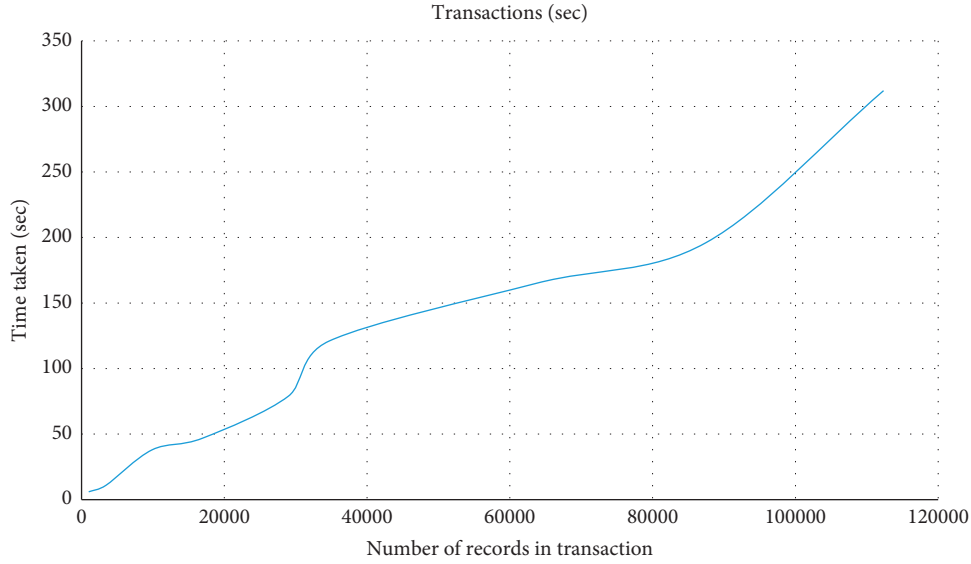

Figure 7: Transactions in row format.

with each other through private IP, and each of these instances has its own public IP which is visible to everyone. We then configure a cluster that will include both of these instances. Database software is installed on both of these instances, and the view of data is the same for both of them. So, the issue arises that each of these instances has its own IP, how will the user know to which IP I need to connect? We shall configure virtual IP for this purpose and that is configured for the cluster. The user who is connecting to the database will use the virtual IP instead of public IP of the specific instances and will connect to the cluster instead of connecting to the instances. Generally, the cluster decides to route the queries towards lesser loaded instances and If needed it remove or add more instances based on their availability. Traditionally if the cluster is configured, then both of the instances are supposed to be performing the same functionality, but in our case, the instances are not the same, and these are supposed to be performing the same functionality.

Keeping all these reasons in mind, we have to decide which database will we use for the implementation. The database should be a fully ACID complaint and should provide the clustering capabilities also, and its community version should be easily available for research. After analyzing multiple databases which includes MySQL, Postgresql, Microsoft SQL Server, and Oracle, we have decided to go with oracle as its clustering capabilities are better than the remaining and it makes the intelligent shared cluster. Our research focus is to decide the context of the query and type of user and then route the user to the suitable instance. So, how will we achieve this? Technically speaking, if a user has to connect to the database, he has to use a connection

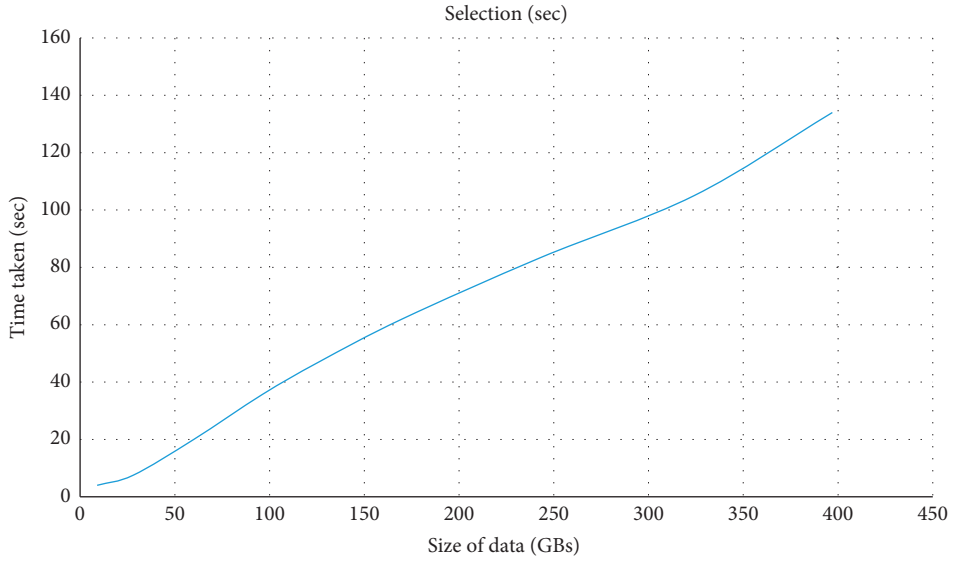

Figure 8: Selection in column format.

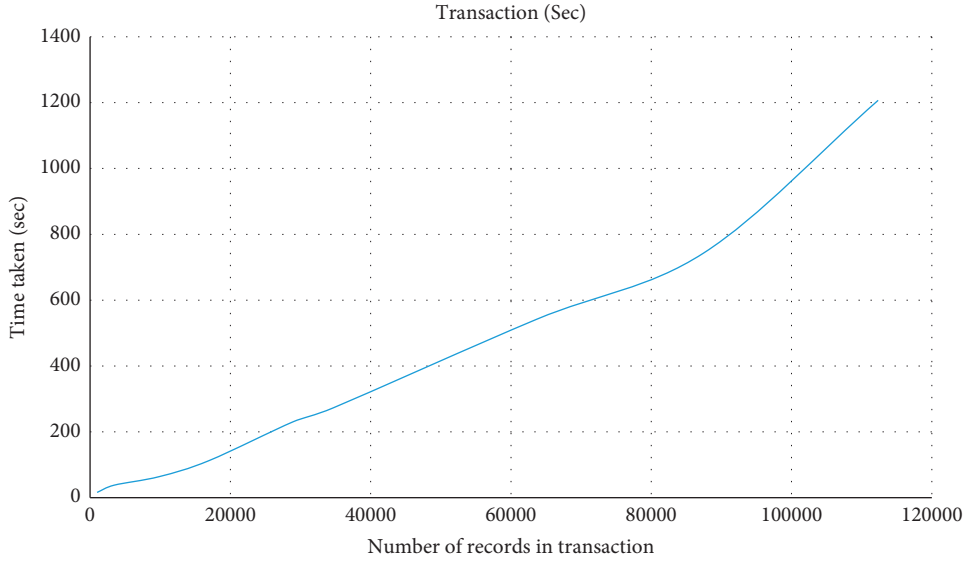

Figure 9: Transactions in column format.

string but has the IP of the database which will be virtual IP in our case. We will create services for the database instances and will use these services in the connection string so that database knows which user has to connect to which database instance. The connection strings are usually provided by Database Administrators to the relevant team users, so the only thing which has to be taken care of are the connection strings being provided to the users. We shall create 2 services for the database. One for the OLTP instance and another for the OLAP instance. The users who are in transaction-based teams will be provided with connection strings with service 1 and the users who are in analytics teams will be provided with connection strings with service 2. So, now since we have 2 different instances with shared storage and different services to be connected to them, we can now move on to implementing the main thing inside instances, i.e., layout definitions. We shall let instance 1 use its native layout which is row-based for transactions, and we shall configure instance 2 to store data in its memory in columnar format. Since the native layout is row-based both on disk and memory, we shall keep the size of the memory as per the organizational conventions. But since in instance 2, we will be keeping the data in a columnar format in memory, and we shall keep the size of the memory to a fairly large value which can be increased or decreased as we gain experience. This needs to be decided internally with the analytics team that which tables/partitions/views do they need at which time, and these can be loaded accordingly as per their needs into the memory of instance 2.

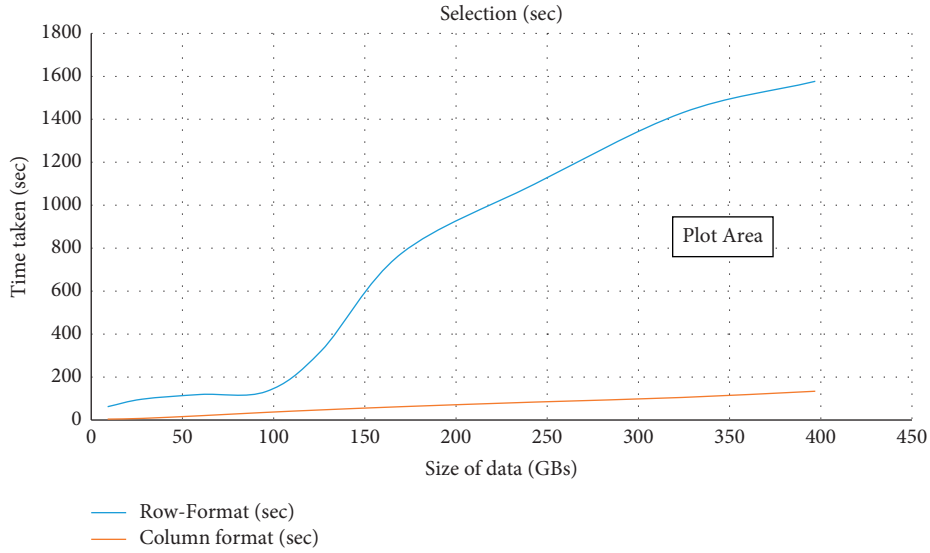

Figure 10: Selection in row format vs. column format.

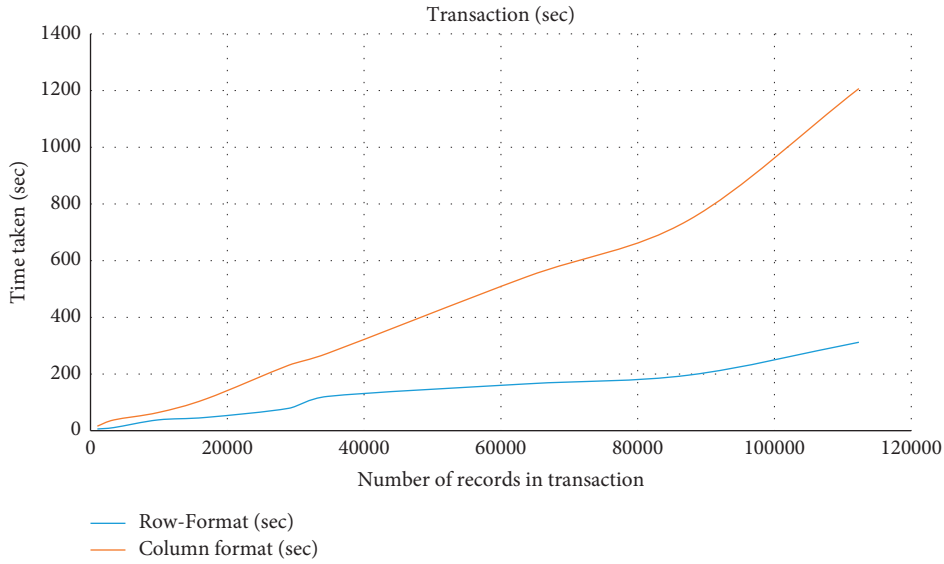

Figure 11: Transactions in row format vs. column format.

6. Experiments and Results

Now, we shall evaluate our proposed model and see how this can contribute to the overall idea. We will be validating a few things in our experiments which we claim to be the basis of the whole study, and these include that the row store works better for OLTP while column stores work better for OLAP, and these can co-locate in a database on their specified instances We shall start with looking at the conventional systems where the source databases are optimized for OLTP. For experimental purposes, we have acquired data from a corporate and table sizes range from 9 GBs to around 400.

GBs include partitioned as well nonpartitioned tables. Right now, we have a database with 2 instances, and both of the instances have their memory in a conventional layout which is the row format. For the start, we shall execute a series of select statements on tables of different sizes, and then will execute a series of transactions on the same database with a varying number of records to be updated.

Here, we can visualize in Figures 6 and 7 that the transactions are a bit faster as compared to selection as the database is yet in row format but at the moment we cannot accurately compare both because transactions are being evaluated on number of records while selection on the size of the database. The picture will keep on getting clearer as we shall proceed on the further cases.

Now, we shall convert the memory into the column store on both of the instances and will see how the system performs for selections and transactions. Ideally, this format should support selection or scan of records better.

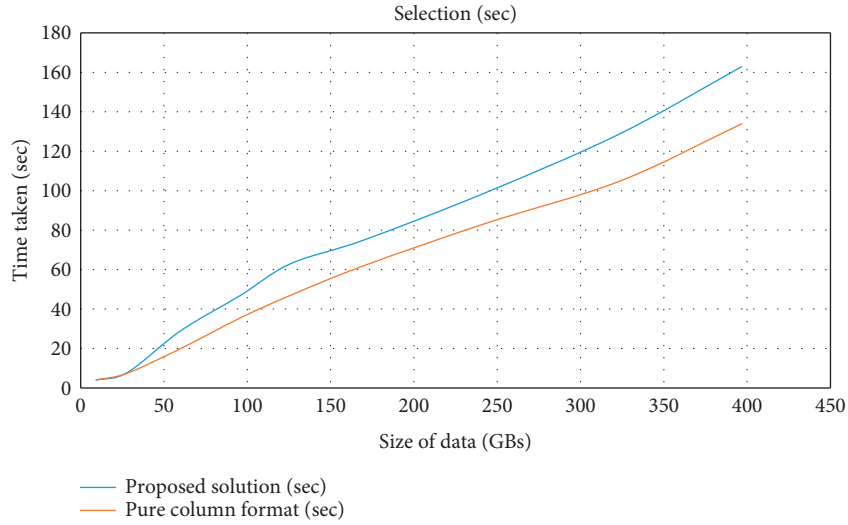

Figure 12: Selection when clustering in column format vs. proposed solution.

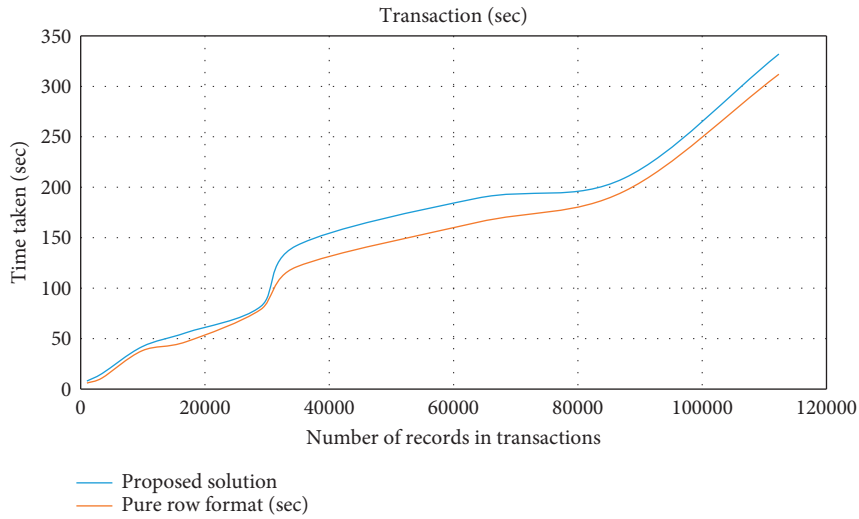

Figure 13: Transactions when clustering in row format vs. proposed solution.

Now, here we see that the selection of records seems to be faster and transactions seem slower as we can see in Figures 8 and 9. Here, we can establish that the row format is working better for the transactions while the column format is working better for selection of records. Now, let us combine the results from both of the experiments done above and see how much the format affects the speed of selection or transactions. Since the comparison is not between selection and transaction, this actually is selection in rowformat vs. selection in column format and similar transactions in row format vs. transactions in column format as shown in Figure 10.

We have seen that the same scan queries have performed a lot better in column format as compared to row format.

Now, let us see the comparisons between transactions performing in row format vs. transactions performing in column format Figure 11.

As expected, we can see that the transactions are performing better in row format as compared to column format. Since now, we have established the base that which type of queries work better in which format of data layout. In both the scenarios, we either had the data fully row formatted or column formatted. Now, we have to co-locate both of the formats. For this purpose, we shall now keep the one instance in native layout, i.e., row store while we shall change the other instance and will arrange its memory in column format, we shall keep the memory of the second node a little higher as we would expect that most of the data that the analytics team should be fetched into memory already. Here, we shall run the same queries which we ran earlier and will check their results to compare how well they perform in the proposed setup. Given below are the results, and the other lines in comparison are the results when the database was either fully row formatted or fully column formatted. In the selection, we shall compare the results of our proposed solution with the results in column format we checked earlier and plot it in Figure 12, and in transactions, we shall compare the results of our proposed solution with the results

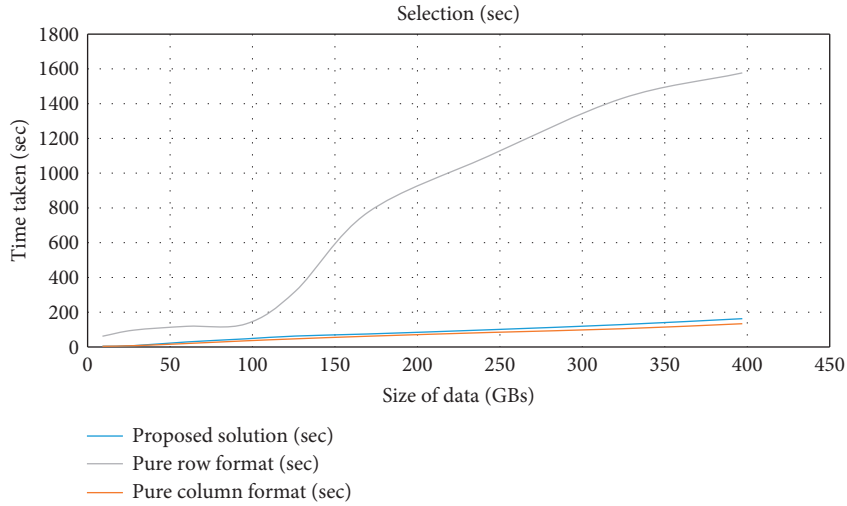

Figure 14: Selection in all scenarios combined.

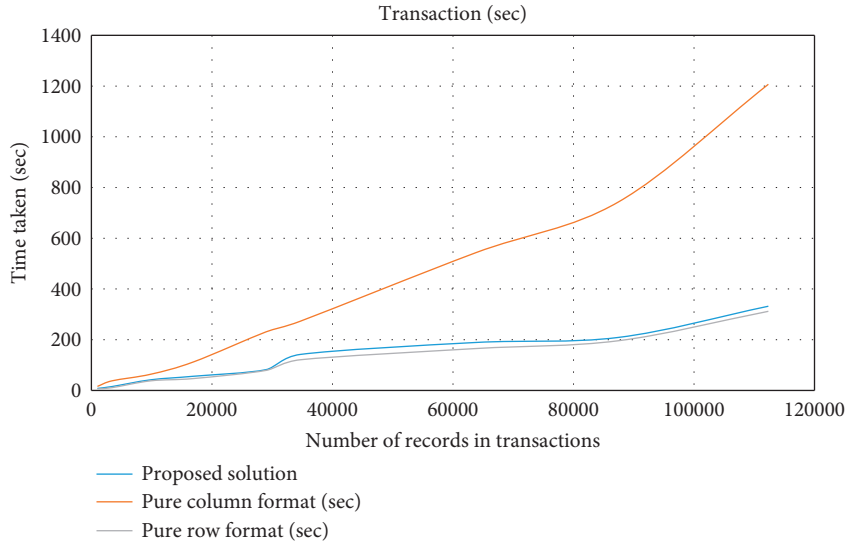

Figure 15: Transactions in all scenarios combined.

in row format which we also checked earlier and plot these in Figure 13.

Here, we can see that there is very slight performance degradation in our proposed solution as compared to when the database was in full row format or in full column format. The reason for this is that we have a cluster of 2 servers. When both nodes were in the same format, both of these servers were contributing to fetch the results so that computations were a bit faster; since now, we have one server in one format and we could see a slight slow. This is not very significant in this study, and we will not count it as a drawback because corporates have enough budgets so when they would implement the solution they can afford to keep 4 nodes in a cluster instead of 2 where 2 each will contribute for each type of format. Anyways the performance gain is very much as compared to the traditional approach. So, if we summarize the whole thing, we shall get the results as in Figures 14 and 15.

Here, the worst performing lines are either selection in a row format or transactions in column format. So, implementing the proposed solution brings is better of both worlds in our case.

7. Conclusion and Future Work

We have presented an architecture for the databases which can accommodate the needs for analytics and transactions in the same database. Also, this is going to be a necessary requirement for the organizations to have a view of the latest data at the earliest to fetch insights from it. This will allow the users to analyze the latest view of data without having the need for waiting for the latest data to be available through specialized jobs. The trade-offs we get from this approach is we need to have more memory at the column formatted instances, we may need to maintain more instances of the databases, and we have to look after segregation of users

carefully else they might land at the wrong layout which may worsen the situation for them. While in this paper, we knew the segregation of our users and we created different connection services for each of these which will route their queries to their respective engines; this is a good idea to work on making the system generic without having the need to tell the system which user is of which type and letting the system decide who the user is and route the query to the suitable engine.

In the future, instead of making the services, a parser can be developed, which will check whether the query is analytical or transactional and then make the suitable engine take over from there. Secondly, one can then embed a machine learning module which can track the behaviour of the system based on which queries are executed at which time by which user. This way we can better populate the query cache and buffer caches proactively. The focus will be to minimize the manual efforts and let the database do things for itself.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by the Technology Development Program (S2966549) funded by the Ministry of SMEs and Startups (MSS, Korea).

References

[1] V. Raman, G. Attaluri, R. Barber et al., "DB2 with BLU acceleration: so much more than just a column store," *Proceedings of the VLDB Endowment*, vol. 6, no. 11, pp. 1080–1091, 2013.

[2] U. Shafiq, M. K. Shahzad, M. Anwar, Q. Shaheen, M. Shiraz, and A. Gani, "Transfer Learning Auto-Encoder Neural Networks for Anomaly Detection of DDoS Generating IoT Devices," *Security and Communication Networks*, vol. 2022 Article ID 8221351, 13 pages, 2022.

[3] M. Anwar, A. H. Abdullah, A. Altameem et al., "Green communication for wireless body area networks: energy aware link efficient routing approach," *Sensors*, vol. 18, no. 10, p. 3237, 2018.

[4] M. Stonebraker and A. Weisberg, "The VoltDB Main Memory DBMS," *IEEE Data Eng. Bull.*vol. 36, no. 2, p. 2127, 2013.

[5] C. Diaconu, C. Freedman, E. Ismert et al., "Hekaton: SQL Server's memory- optimized OLTP engine," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, Article ID 12431254, New York, NY, USA, June 2013.

[6] MemSQL: http://www.memsql.com.

[7] S. Amjad, M. Younas, M. Anwar, Q. Shaheen, M. Shiraz, and A. Gani, "Data Mining Techniques to Analyze the Impact of Social Media on the Academic Performance of High School Students," *Wireless Communications and Mobile Computing*, vol. 2022, Article ID 9299115, 11 pages, 2022.

[8] A. Cassandra: http://apache.cassandra.org.

[9] RocksDB: http://rocksdb.org.

[10] M. Anwar, F. Masud, R. Aslam Butt, S. Mahdaliza Idrus, M. Nazir Ahmad, and M. Yazid Bajuri, "Traffic priority-aware medical data dissemination scheme for IoT based WBASN healthcare applications," *Computers, Materials & Continua*, vol. 71, no. 3, pp. 4443–4456, 2022.

[11] M. Kamran, M. Malik, M. W. Iqbal, M. Anwar, and M. Aqeel, "Web Simplification Prototype for Cognitive Disable Users," *Human Behaviour and Emergency Techonolgy*, vol. 2022, Article ID 5817410, 14 pages, 2022.

[12] S. Gray, F. Ozcan, H. Pereyra, B. Van der Linden, and A. Zubiri, "IBM Big SQL 3.0: SQL-On-Hadoop without Compromise," 2014.

[13] M. Kornacker, A. Behm, V. Bittorf et al., "Impala: A Modern, Open-Source SQL Engine for Hadoop," in *Proceedings of the 7th Biennial Conference on Innovative Data Systems Research (CIDR'15)*, Asilomar, CA, USA, January 2015.

[14] M. Armbrust, R. S. Xin, C. Lian et al., "Spark SQL: Relational Data Processing in Spark," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, Article ID 13831394, Melbourne, Victoria, Australia, May 2015.

[15] A. Kemper and T. Neumann, "HyPer A Hybrid OLTPOLAP Main Memory Database System Based on Virtual Memory Snapshots," in *Proceedings of the 2011 IEEE 27th International Conference on Data Engineering*, Article ID 195206, Hannover, Germany, April 2011.

[16] A. Pavlo, J. Arulraj, L. Ma et al., "Self-Driving Database Management Systems," in *Proceedings of the Biennial Conference on Innovative Data Systems Research (CIDR)*, Asilomar, CA, USA, 2017.

[17] L. Ma, D. Van Aken, A. Hefny, G. Mezerhane, A. Pavlo, and G. J. Gordon, "Query-based workload forecasting for self-driving database management systems," in *Proceedings of the 2018 International Conference on Management of Data*, Houston, TX, USA, May 2018.

[18] J. Arulraj, A. Pavlo, and P. Menon, "Bridging the archipelago between row-stores and column-stores for Hybrid workloads," in *Proceedings of the 2016 International Conference on Management of Data*, San Francisco, CA, USA, June 2016.