Reducing Workload On Real-Time Database Using Machine Learning Based Transaction Scheduling

Ashok Kumar Panda [1]* Jagannath Patel[2]
[1*] Department of Computer Science
Utkal University
Bhubaneswar-751004
Email: ashopanda@gmail.com
[2] PG Department of Mathematics
Utkal University
Email: jpatelmath@yahoo.co.in

Article History: Received: 11 January 2021; Revised: 12 February 2021; Accepted: 27 March 2021; Published online: 20 April 2021

Abstract: Reading and writing to relational databases requires accessing multiple tables for constraint & quality checks. In order to perform these checks, databases use transaction management, wherein index-based checking & validation is done, and data is committed to the database only when these checks are satisfied. In case of any validation violations, databases need to either fall back to previous data state, or activate violation rule engine and resolve the underlying conflicts. Performing these tasks for limited size databases doesn't compromise on system performance, but as database sizes increase, the number of checks increase exponentially, thereby reducing database system performance. In order to reduce the effect of database size on transaction scheduling performance, this work proposes a genetic algorithm inspired algorithm, which takes into consideration multiple performance parameters in order to optimize transaction performance. The underlying system is deployed on multiple relational databases, and a performance improvement of 10% in terms of scheduled transaction execution delay is observed. This performance is compared with recently proposed state-of-the-art systems, and it is observed that the proposed model is able to reduce execution delay by 5% across multiple implementations.

Keywords: Database, real-time, transaction, scheduling, machine, learning

1. Introduction

Transaction based scheduling requires databases to be checked on multiple indexing levels on a per transaction basis. These indexing levels include primary key checks, foreign key checks, cascading index checks, views update checks, etc. In order to effectively perform transaction-based scheduling, a set of mutually dependent operations must be performed effectively and in tandem. These operations are related to sequential query processing and are executed in the following order to obtain effective transaction scheduling performance,

- Evaluate the number of tables being evaluated by the query.
- Evaluate number of individual fields being evaluated from each of the table.
- Find out internal aggregate operations being carried out on each of the fields.
- Segregate the tables into reading & writing tables.
- Ensure atomicity of the transaction via these steps, and if the transaction is not atomic, then re-evaluate these steps.
- Allocate sufficient buffer space in order to ensure recoverability of the transactions.
- Provide these transactions to a scheduler in order to ensure serializability of queries to re-order requests.
- Lock the tables and execute the separated queries.
- Execute queries on the locked tables, and evaluate its correctness.
- In case of any errors, rollback transactions from the buffers and re-evaluate query division process.
- Continue this process, till the entire transaction is executed.

Based on these steps, each transaction scheduler can be in one of 5 states, which are active (when the transaction execution is in process), partially committed (when some transactions have been successfully executed), failed (when the transaction is not getting executed), committed (when the transaction is normally and actively getting executed) and aborted (when user aborts the transaction). State machine for transition between each of these stages must be effectively and clearly defined, so that the overall process of transaction scheduling can be executed with utmost efficiency. The state diagram for a typical transaction management system can be observed from figure 1,

wherein all these states can be seen. Based on the transition between these stages, different types of transaction schedulers are defined by researchers. These schedulers include,

- Serial schedulers, wherein all the queries in a transaction are executed in serial order.
- Conflict serializable schedulers, wherein serial schedulers are used in case of conflicts during execution.
- View serializable schedulers, wherein views are managed using serial execution.
- Cascading schedulers, wherein the scheduler tries to resolve and recover data using cascaded operations.
- Cascade-less schedulers, wherein parallel execution is preferred for recovering data.
- Strict schedulers, wherein scheduling is done based on strict rules, that might affect performance but improve execution efficiency.
- Non-recoverable schedulers, wherein any data lost during scheduling cannot be recovered, and transaction is rolled back.

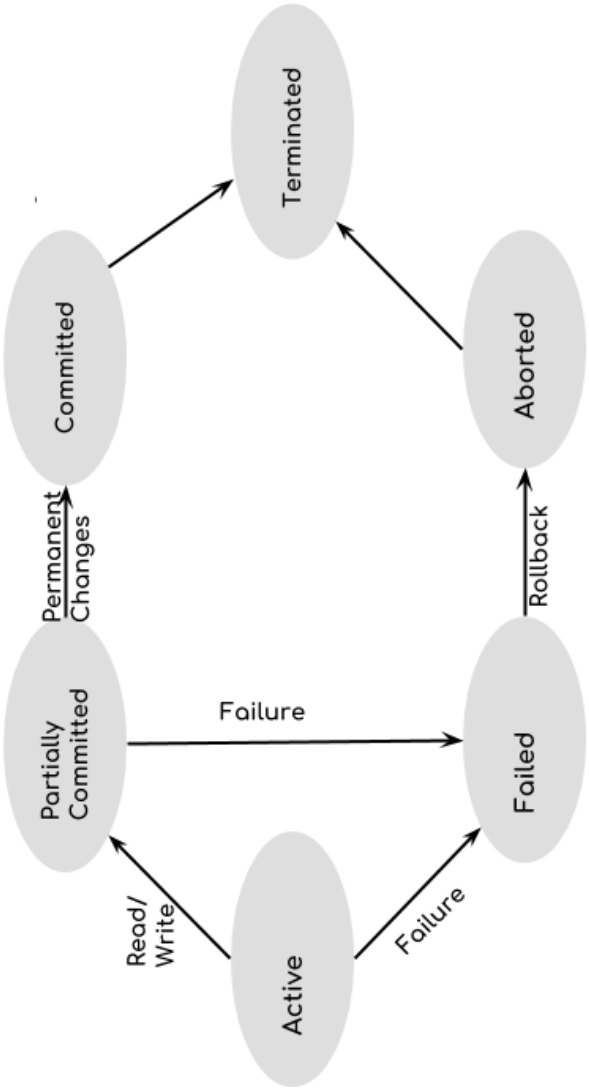

Figure 1. Different stages of a transaction execution system

Depending upon these scheduler types and their different states, a large variety of scheduling architectures are proposed by researchers over the past years. Thus, the next section reviews some of the recently proposed state-of-the-art models for transaction scheduling. This is followed by the proposed Genetic Algorithm based Model for effective transaction scheduling in the network, and its performance evaluation w.r.t. the reviewed models. Finally, this work concludes with some interesting observations about the proposed model, and recommends methods to improve their performance in terms of different performance metrics.

2. Literature Review

Due to the wide applicability of database deployments, relational database systems are used in every business operation irrespective of its size. Because of this wide application, unique transactional scenarios come up, which include, updating the same record at multiple different tables; when one table update is mutually dependent on others, extraction of data from multiple tables conditionally; based on the data contents of internal queried tables, etc. In order to handle such situations, hybrid transactional models are devised by researchers. One such model can be observed from [1], wherein a co-scheduling architecture is defined. This architecture uses a combination of efficient partitioning with hierarchical processing in order to devise an efficient partitioned scheduling model for hybrid transactions. This model has high acceptance ratio due to effective use of upper demand bound functions during transaction processing. It uses deadline-based dependency resolution of transactions in order to improve the transaction acceptance ratio by 15% when compared with other methods. The performance of this model can be improved via addition of load balancing architectures as suggested by [2] and [3], wherein firefly and grey wolf architectures are proposed. These architectures are based on stochastic modelling of the queries, and assist in improving overall execution performance using learn-once-use-continuously architecture. An application of these models on database systems via the use of Coloured Petri Nets is proposed in [4], wherein reliability inspired load balancing is performed by the system via active Markov Model (MM). The Coloured Petri Nets architecture can be observed from figure 2, wherein each query is marked as a work job, and is divided into sub work jobs. Each of these sub work jobs are given to data generators, schedulers, global resource management blocks, local resource management blocks and transaction managers in order to improve task modularity. Due to this, the overall transaction scheduling efficiency is improved by 15% when compared to a non CPN model.

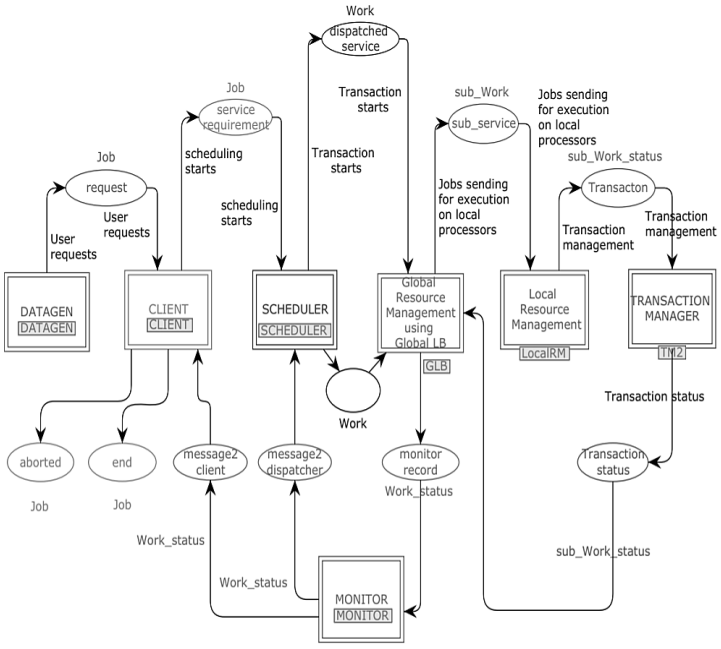

Figure 2. Coloured Petri Nets (CPN) for effective task scheduling [4]

This performance improvement is obtained due to the deadline aware nature of CPNs, which can be represented using equation 1, wherein reliability of query (R_{query}) is formulated.

$$R_{query} = \exp(-\partial * D * \left[\frac{1}{\varphi} + \left(\sum_{i=0}^{c-1} \frac{cp^i}{i!} + \frac{cp^c}{c!\,(1-p)}\right)^{-1} * \frac{p * cp^c}{c! * (c * \partial - \omega)(1-p)}\right] \ldots (1)$$

Where, ∂ is transaction arrival rate, 'D' is deadline of transaction, φ is transaction delay, 'cp' is transaction complexity, 'c' is cost of transaction, 'p' is probability of occurrence of transaction, ω is difference between different arrival rates of the transaction. From this factor, it can be observed that the reliability of query is inversely proportional to query delay, which is reduced by modularizing the query, thereby increasing it. These transactions can be verified using complex metrics like conflict detection rate, freeness ratios, conflict resolution rate, etc. A survey of these factors can be observed in [5], wherein conflict detection and resolution rate are considered as

primary metrics for performance improvement of the transaction scheduling systems. An example of such a high performance transaction management system that uses in-memory computations can be observed from [6], wherein online transaction processing (OLTP) operations are performed with high efficiency on database management systems (DBMS). In order to do this task, the system proposes different policies for conflict resolution, which include count & fraction, literal & canonical; which deals in fine grained processing and single & all; wherein different policies are applied to single transactions and multiple transactions. Due to these policies, the system's flexibility is improved, thereby making the system capable of adding multiple features during its internal performance optimization. The system is extended in [7], wherein different algorithms like weighted graph colouring, colouring-based schedule, complete graph scheduling and hypercube & related graphs are evaluated. These algorithms are applied to distributed transactional memory, and are combined together to form a distributed bucket algorithm. This algorithm is able to improve in memory performance by 18% when compared to individual approaches.

Approaches like neural networks for pattern analysis can also be used for transaction scheduling because of their pattern analysis capabilities. The work in [8] suggests use of radial basis function (RBF) based neural network design, that aims at reducing delay needed for transaction execution, and improve transaction dependency resolution using the following equation,

$$R(q_i, q_j) = \exp\left(-\frac{1}{2 * \partial^2} * \left|d_{q_i} - d_{q_j}\right|^2\right) \dots (2)$$

Where, 'q' represents given query, ∂ is variance in the query execution cost which is formulated in equation 3, and 'd' is the dependency level of query on sub queries.

$$\partial = \frac{1}{P} * \sum_{i=1}^{m} |d_i - q_i * c_i| \dots (3)$$

Where, 'P' is probability of occurrence of the query, 'c' is query cost, and 'm' is number of sub-queries for the given input query. Due to the use of this error reduction model, reliability of query execution improves by 8%, while the delay of subsequent execution reduces by 5%, which improves overall system performance. This performance can be further improved by clustering the queries based on location of their application, thereby executing queries which require immediate results, while holding queries that require deferred results. Work in [9] and [10] propose such models, wherein query performance-based clustering is done in order to improve overall system performance. This performance can also be improved via use of hybrid execution environments like the ones proposed in [11] and [12], wherein load balancing models like Shortest job scheduling algorithm, Throttled algorithm, Genetic Algorithm (GA), Modified Active Monitoring Load Balancer (MAMLB), etc. are defined and their combinations are studied for improving query execution performance. A survey of these algorithms suggests that transaction scheduling models that use load balancing can be broadly divided into static and dynamic models, and a list of these individual models can be observed from figure 3, wherein models like opportunistic load balancing (OLB) & Min–Min load balancing (LBMM) are defined. Out of these models the Genetic Algorithm (GA) and Ant Colony Optimization (ACO) provide optimum performance due to their stochastic properties. Both these techniques propose different fitness functions, all of which are aimed at reducing delay and increase reliability of query processing. A sample GA fitness function can be formulated using equation 4, wherein delay and reliability are considered for improved query execution performance.

$$F(q) = \frac{1}{\in * R_q + \partial * D_q} \dots (4)$$

Where, 'F' is the fitness, 'R' is query reliability, 'D' is query delay, $\in$ is reliability scaling factor, and ∂ is delay scaling factor, which are used to normalize the fitness value. Performance of these models can be improved via deploying them over the cloud with optimum load balancing models as discussed in [13] and [14]. Here machine learning and artificial intelligence models are deployed in order to pre-empt query execution, and thereby effectively dividing queries among cloud nodes for utmost efficiency.

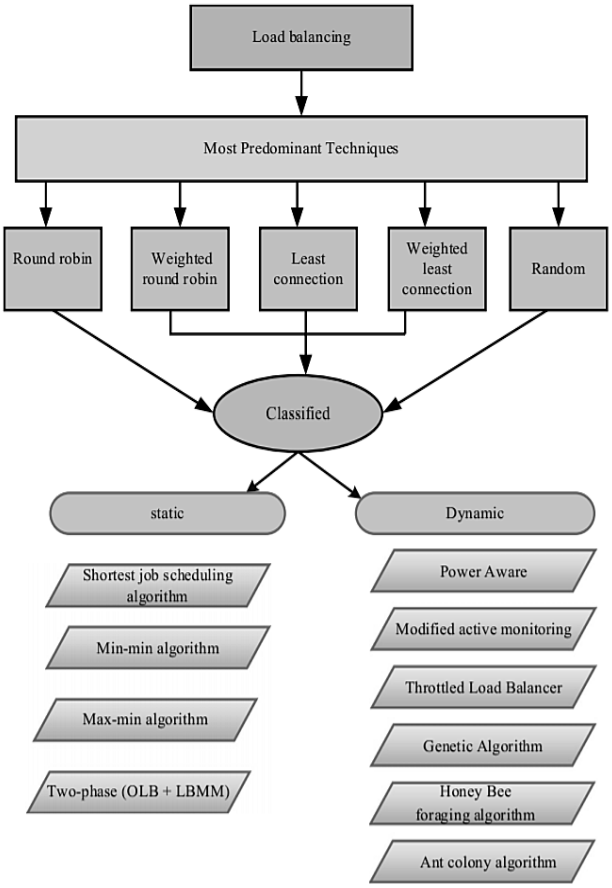

Figure 3. Load balancing models for transaction scheduling [12]

Query processing models can also use scheduling algorithms like earliest deadline first (EDF) and its variants in order to improve their performance. Work in [15] proposes use of semi-partitioned EDF model for splitting tasks for real-time dynamic workloads. Due to this query partitioning, overall delay of execution is reduced by 15%, while query execution reliability is improved by 9% when compared with a non-EDF system. This work can be used to enhance performance of memory-based scheduling via software transactional memory, which shows high atomicity performance, but the performance reduces exponentially due to query conflicts. These conflicts can be reduced via the use of partitioned EDF framework, that aims at reducing the queries into multiple sub queries, and executing them on software transactional memory for high performance. An application of this model can be observed from [17], wherein EDF is combined with in-memory execution. A similar EDF-based algorithm for reducing inconsistencies in delay and reliability during transaction scheduling is observed from [18]. This algorithm is named as jitter based EDF (JB EDF) and aims at reducing deadline & execution jitters using the following modified deadline equation,

$$J_d = \left[\frac{E_d}{E_d - \sum_{j=1}^{N} \frac{D_j}{N}} + \frac{R_d}{R_d - \sum_{j=1}^{N} \frac{R_j}{N}} \right]^{-1} \quad \ldots (5)$$

Where, ‘J’ is the performance jitter, ‘E’ is the EDF deadline, ‘D’ is delay, ‘N’ is number of queries to execute, and ‘R’ is reliability factor for executing this query that can be evaluated using equation 2. Due to minimization of jitter, overall delay of query execution is reduced by 5%, and reliability is improved by 9% when compared with a non-jitter-reduction algorithm. This performance can be further improved by executing queries over federated and high-speed clouds as suggested in [19] & [20]. Here, a performance enhancement of over 10% is achieved via increasing the processing capabilities of the query execution engine.

Researchers have worked towards improving query processing performance via use of static locking in distributed database. The work in [21] suggests such a novel model that uses a combination of Static Two-Phase Locking Protocol & dynamic two-phase locking protocol to form a hybrid locking protocol. This protocol locks

transactions pre-emptively, and doesn't allow any modifications to the buffer until all dependencies are resolved. Due to this high-end conflict resolution, a performance improvement of 15% is achieved in terms of query execution reliability, but delay is increased by 8% when compared to a single-phase locking protocol. This performance can be improved via the use of containers for query execution as suggested in [22] & [23], wherein different algorithms for speeding up query performance are discussed. One of these algorithms uses a concurrent architecture wherein parallel pools of query execution engines are deployed. Each pool is able to effectively resolve query dependencies using directed acyclic graphs, thereby improving overall system performance. A similar algorithm that aims at optimizing performance of join queries via fine-grained partitioning for skew data is mentioned in [24]. This model works on map reduce database, which makes it applicable for big data applications. Performance of these models can also be improved via use of efficient indexing, wherein indexes are created such that queries are easily separated from each other, and are processed at high speeds. Such a model that uses reinforcement learning for indexing is proposed in [25]. Due to the use of reinforcement learning, overall efficiency of query execution is improved by 15%, and thus can be used for real-time database deployments. The scalability of these algorithms can be improved via the use of machine learning models suggested in [26] and [27]. Due to use of machine learning models like Genetic Algorithm, high scalability in infrastructure, computing and application layers can be achieved. This architecture is Algorithm agnostic, which indicates dynamic flexibility of the system due to changes in data size, table dimensions, index size, etc. The architecture can also be used in pipelined mode as suggested in [28], due to which overall throughput of transaction scheduling is increased. A comprehensive review done in [29] also suggests use of bio-inspired models for improving efficiency of transaction scheduling in real-time databases. Based on this study, this text proposes a novel Genetic Algorithm based transaction scheduling model for real-time databases to achieve high transactional efficiency. This model is described in the next section, and is followed by its performance analysis and comparison.

3. Proposed Genetic Algorithm based model for improving transaction scheduling performance

Genetic algorithm comes from a family of stochastic optimization algorithms, wherein a large number of solutions are evaluated in order to solve any problem. A similar approach is undertaken in this text, wherein performance for a large number of query combinations is evaluated, and compared internally to form the best possible solution for transaction scheduling. The proposed Genetic Algorithm initially divides the entire query set into training and testing queries, with a ratio of 70:30, wherein 70% of queries are used for training the system, while remaining 30% queries are used for testing & optimization of system performance. Flow of algorithm can be observed using the following steps,

- Input,
  - Number of iterations (Ni)
  - Number of solutions (Ns)
  - Learning rate (Lr)
  - Minimum query result length (QLmin)
- Initially mark all solutions as 'to be mutated'
- For each iteration in 1 to Ni,
  - For each solution in 1 to Ns,
    - If the solution is marked as 'not to be mutated', then continue to next solution.
    - Else, divide the query into 'k' random parts, such that results of each of the 'k' parts is more than QLmin

$$k = random(QLmin, Qmax) \dots (6)$$

Where, Qmax is the max length of the query.

- Interleave the queries using 'm' percentage EDF algorithm, wherein deadline of the initial m% of queries is considered.

$$m = random(1, k) \dots (7)$$

- Evaluate the transitive dependencies of these queries.
- If the dependencies are not fulfilled, then re-arrange the query components till all dependencies are resolved by using different values of 'm'. This will evaluate new values of 'm' until atomicity of the query is achieved.
- Execute the query on real-time database, and evaluate fitness using the following equation,

$$f_i = \frac{\sum_{i=1}^{k} d_{q_i} * \left(\frac{\sum_{j=1}^{m} \frac{C_{r_j}}{C_j}}{m} \right)^{-1}}{k} \ldots (8)$$

Where, d_{q_i} is delay to execute the i^{th} sub-query, C_{r_j} is the number of conflicts resolved for the j^{th} sub-query, and C_j is total number of conflicts that occurred during this execution.

- Evaluate fitness for all solutions, then evaluate fitness threshold using the following equation,

$$f_{th} = \frac{\sum_{i=1}^{Ns} f_i}{Ns} * Lr \ldots (9)$$

- Mark all solutions as 'to be mutated', where the fitness value is more than threshold, and mark all others as 'not to be mutated'

- At the end of all iterations, select the solution with minimum fitness, and identify values of 'k' and 'm' for that particular solution.
- Use the testing set, to evaluate the efficiency of these 'k' and 'm' values.
- Efficiency is evaluated by using the values of 'k' and 'm' for each test query, and comparing its performance with the work in [4] and [21]
- If the performance is lower, then the queries which inhibit low performance are added to the training set, and the entire GA is repeated for this new set.
- Once the new GA has produced results, then they are again tested on the entire test set to optimize performance.

Flow of this algorithm can be observed from figure 4, wherein multiple GA executions are seen depending upon current performance. The main aim of this algorithm to select the values of 'k' and 'm' such that overall query execution performance is improved. This also indicates that the system is now able to resolve all dependencies in the underlying queries and is ready to schedule real-time queries and execute them effectively on the database.

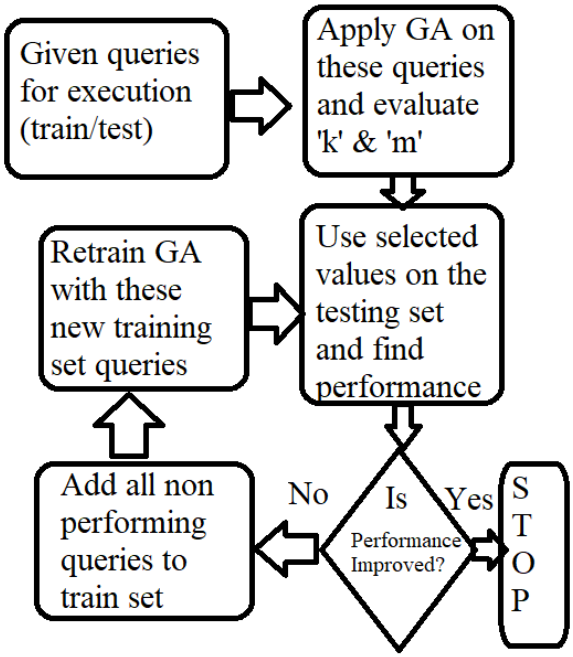

Figure 4. Flow of the proposed GA based model

Result evaluation & comparative of this proposed model is done on Google's Firebase and Apache's MongoDB databases, and is compared with the implementations provided by [4] and [21]. This result evaluation can be observed from the next section.

4. Result and analysis

In order to evaluate performance of the proposed model, the primary parameters; which are; execution delay, query throughput and conflict resolution accuracy are considered. These parameters are evaluated for 2 real-time databases, which are Google's Firebase and Apache's MongoDB. Number of transactions on each database were varied between 1000 to 500k, and a mix of read, write and access requests were made to the database. Average values of execution delay (D), query throughput (T), and conflict resolution accuracy (R) are evaluated using the following equations,

$$D = \frac{\sum_{i=1}^{N} T_{out_i} - T_{in_i}}{N} \dots (6)$$

$$T = \frac{\sum_{i=1}^{N} B_{r_i} + B_{w_i}}{N} \dots (7)$$

$$R = \frac{\sum_{i=1}^{N} \frac{N_{cr_i}}{N_{c_i}}}{N} \dots (8)$$

Where, T_{out_i} and T_{in_i} are the time instances at which query result is output and query is input to the system respectively, B_{r_i} & B_{w_i} are the number of bytes read & number of bytes written by the query respectively, N_{cr_i} & N_{c_i} are number of conflicts resolved, & total number of conflicts in the i^{th} query, and 'N' are total number of queries which are executed by the system. Results are compared with [4] and [21], due to their applicability and high performance of query execution. The results for delay can be observed from table 1, wherein different number of transactions are varied and its delay performance is evaluated.

Num. Trans.	**Delay [4]. (s)**	**Delay [21] (s)**	**Delay [Proposed]. (s)**
1000	31.20	65.50	26.56
2000	39.20	82.30	33.36
5000	48.40	101.65	41.20
10k	52.95	111.20	45.08
20k	56.80	119.30	48.36
50k	63.95	134.30	54.44
100k	67.85	142.50	57.76
200k	69.95	146.90	59.56
300k	72.85	153.00	62.04

400k	81.63	171.46	69.51
500k	86.78	182.27	73.90

Table 1. Delay performance for different transactions

From the delay performance it can be observed that the proposed model is 10% to 25% more effective than state-of-the-art methods. This can also be observed from figure 5, wherein the delay is visualized against number of communications.

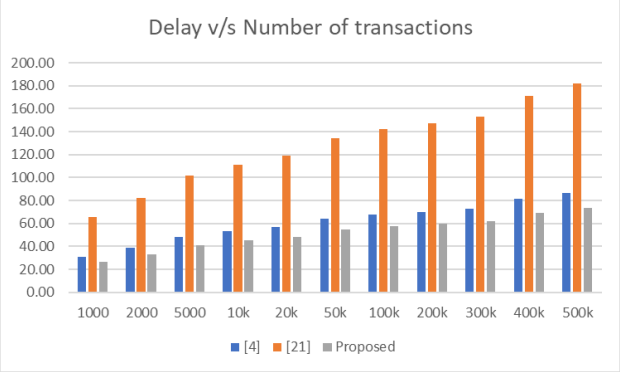

Figure 5. Delay v/s Number of transactions

Similarly, the results for throughput can be observed from table 2, wherein different number of transactions are varied and its average throughput performance is evaluated.

Num. Trans.	**T [4]. (kbps)**	**T [21] (kbps)**	**T [Proposed]. (kbps)**
1000	23.81	12.48	32.75
2000	29.91	15.68	41.15
5000	36.94	19.36	50.83
10k	40.41	21.18	55.60
20k	43.35	22.72	59.65
50k	48.81	25.58	67.15
100k	51.78	27.14	71.25
200k	53.38	27.98	73.45
300k	55.60	29.14	76.50
400k	62.31	32.65	85.73

500k	66.23	34.71	91.13

Table 2. Throughput performance for different transactions

From the throughput performance it can be observed that the proposed model is 15% to 20% more effective than state-of-the-art methods. This can also be observed from figure 6, wherein the throughput is visualized against number of communications.

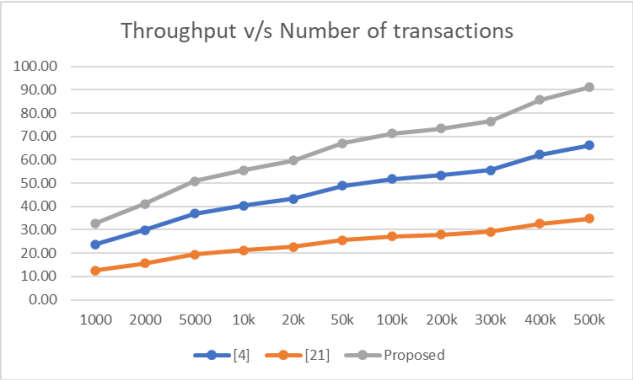

Figure 6. Throughput v/s Number of transactions

Finally, the results for conflict resolution accuracy can be observed from table 3, wherein different number of transactions are varied and its average conflict resolution performance is evaluated.

Num. Trans.	**R [4]. (%)**	**R [21] (%)**	**R [Proposed]. (%)**
1000	79.59	79.59	83.78
2000	80.97	80.99	85.24
5000	91.41	91.41	96.22
10k	93.21	93.22	98.12
20k	88.83	88.82	93.50
50k	94.25	94.25	99.21
100k	97.00	97.00	99.10
200k	96.01	96.02	99.40
300k	89.24	89.24	93.93
400k	94.07	94.07	99.02
500k	95.60	97.30	99.30

Table 3. Conflict resolution performance for different transactions

From the conflict resolution performance, it can be observed that the proposed model is 4% higher effective than state-of-the-art methods. This can also be observed from figure 6, wherein the average conflict resolution is visualized for the given number of communications.

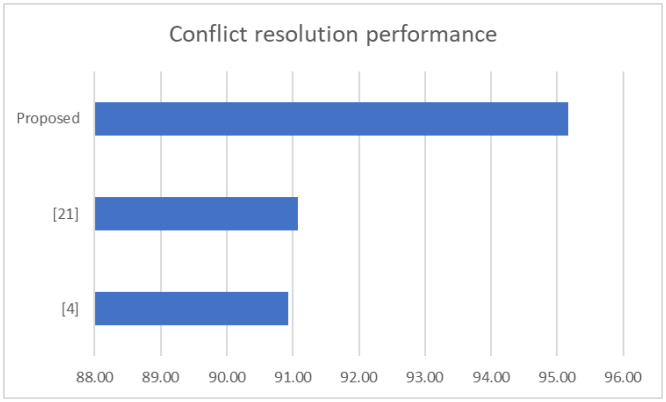

Figure 6. Average conflict resolution performance

From these results it is observed that the proposed model is superior in terms of execution delay, throughput and overall conflict resolution performance, when compared to the state-of-the art models.

5. Conclusion and future scope

Due to use of Genetic Algorithm for query analysis, the overall system performance of transaction scheduling is improved. Genetic algorithm analyzes training set queries and uses that information on testing set queries for highly efficient predictive analysis. Due to which the system is able to reduce execution delay and estimate conflict resolutions with high performance. This performance is compared in terms of overall query execution delay, system throughput while executing these queries, and conflict resolution accuracy. It is observed that the proposed model is 15% efficient in terms of overall throughput, atleast 10% efficient in terms of query execution delay, and 4% effective in terms of conflict resolution when compared with Coloured Petri Nets and EDF based implementations. Performance of this model can be further improved via use of convolutional neural networks for query analysis. Models like long-short-term-memory (LSTM) & gated recurrent units (GRUs) can be used for further improvement of delay and throughput performance.

References

1. C. Deng, G. Li, Q. Zhou and J. Li, "Co-Scheduling of Hybrid Transactions on Multiprocessor Real-Time Database Systems," in IEEE Access, vol. 7, pp. 109506-109517, 2019, doi: 10.1109/ACCESS.2019.2932799.
2. Yong, H. (2020), Load balancing strategy for medical big data based on low delay cloud network. J. Eng., 2020: 799-804. https://doi.org/10.1049/joe.2020.0126
3. Xingjun, L, Zhiwei, S, Hongping, C, Mohammed, BO. A new fuzzy-based method for load balancing in the cloud-based Internet of things using a grey wolf optimization algorithm. *Int J Commun Syst*. 2020; 33:e4370. https://doi.org/10.1002/dac.4370
4. D. P. Mahato and J. K. Sandhu, "Modeling of Load Balanced Scheduling and Reliability Evaluation for On-demand Computing Based Transaction Processing System," 2018 IEEE 14th International Conference on e-Science (e-Science), 2018, pp. 390-391, doi: 10.1109/eScience.2018.00114.
5. C. Xu, X. Wu, H. Zhu and M. Popovic, "Modeling and Verifying Transaction Scheduling for Software Transactional Memory using CSP," 2019 International Symposium on Theoretical Aspects of Software Engineering (TASE), 2019, pp. 240-247, doi: 10.1109/TASE.2019.00009.
6. T. Zhang, A. Tomasic, Y. Sheng and A. Pavlo, "Performance of OLTP via Intelligent Scheduling," 2018 IEEE 34th International Conference on Data Engineering (ICDE), 2018, pp. 1288-1291, doi: 10.1109/ICDE.2018.00132.
7. C. Busch, M. Herlihy, M. Popovic and G. Sharma, "Dynamic Scheduling in Distributed Transactional Memory," 2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2020, pp. 874-883, doi: 10.1109/IPDPS47924.2020.00094.

8. J. Hu, X. Wei, M. Yang, B. Tang, K. Lin and Y. Zhong, "A Practical RBF Framework for Database Load Balancing Prediction," 2020 3rd International Conference on Artificial Intelligence and Big Data (ICAIBD), 2020, pp. 83-86, doi: 10.1109/ICAIBD49809.2020.9137481.
9. S. Subramaniam and G. Krishnamurthi, "Load balancing location management," ICC 2001. IEEE International Conference on Communications. Conference Record (Cat. No.01CH37240), 2001, pp. 2835-2839 vol.9, doi: 10.1109/ICC.2001.936667.
10. Weihua Gong and Yuanzhen Wang, "Load balancing of OLTP on heterogeneous database cluster," 2006 8th International Conference Advanced Communication Technology, 2006, pp. 6 pp.-2045, doi: 10.1109/ICACT.2006.206398.
11. Chen JB., Pao TL., Lee KD. (2009) Effect of Database Server Arrangement to the Performance of Load Balancing Systems. In: Hua A., Chang SL. (eds) Algorithms and Architectures for Parallel Processing. ICA3PP 2009. Lecture Notes in Computer Science, vol 5574. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-03095-6_15
12. Jyoti, A., Shrimali, M., Tiwari, S. *et al.* Cloud computing using load balancing and service broker policy for IT service: a taxonomy and survey. *J Ambient Intell Human Comput* **11,** 4785–4814 (2020). https://doi.org/10.1007/s12652-020-01747-z
13. Gundu, S.R., Panem, C.A. & Thimmapuram, A. Real-Time Cloud-Based Load Balance Algorithms and an Analysis. *SN COMPUT. SCI.* **1,** 187 (2020). https://doi.org/10.1007/s42979-020-00199-8
14. Cai, W., Zhu, J., Bai, W. *et al.* A cost saving and load balancing task scheduling model for computational biology in heterogeneous cloud datacenters. *J Supercomput* **76,** 6113–6139 (2020). https://doi.org/10.1007/s11227-020-03305-y
15. D. Casini, A. Biondi and G. C. Buttazzo, "Task Splitting and Load Balancing of Dynamic Real-Time Workloads for Semi-Partitioned EDF," in IEEE Transactions on Computers, doi: 10.1109/TC.2020.3038286.
16. P. Di Sanzo, A. Pellegrini, M. Sannicandro, B. Ciciani and F. Quaglia, "Adaptive Model-Based Scheduling in Software Transactional Memory," in IEEE Transactions on Computers, vol. 69, no. 5, pp. 621-632, 1 May 2020, doi: 10.1109/TC.2019.2954139.
17. C. Deng, G. Li, Q. Zhou and J. Li, "Guarantee the Quality-of-Service of Control Transactions in Real-Time Database Systems," in IEEE Access, vol. 8, pp. 110511-110522, 2020, doi: 10.1109/ACCESS.2020.3002335.
18. G. Li, c. zhou, J. Li and B. Guo, "Maintaining Data Freshness in Distributed Cyber-Physical Systems," in IEEE Transactions on Computers, vol. 68, no. 7, pp. 1077-1090, 1 July 2019, doi: 10.1109/TC.2018.2889456.
19. Zhang, Y., Zhang, Y., Lu, J. *et al.* One size does not fit all: accelerating OLAP workloads with GPUs. *Distrib Parallel Databases* **38,** 995–1037 (2020). https://doi.org/10.1007/s10619-020-07304-z
20. Wang, B., Wang, C., Song, Y. *et al.* A survey and taxonomy on workload scheduling and resource provisioning in hybrid clouds. *Cluster Comput* **23,** 2809–2834 (2020). https://doi.org/10.1007/s10586-020-03048-8
21. Lam, KY., Hung, SL. & Son, S.H. On Using Real-Time Static Locking Protocols for Distributed Real-Time Databases. *Real-Time Systems* **13,** 141–166 (1997). https://doi.org/10.1023/A:1007981523223
22. Imdoukh, M., Ahmad, I. & Alfailakawi, M.G. Machine learning-based auto-scaling for containerized applications. *Neural Comput & Applic* **32,** 9745–9760 (2020). https://doi.org/10.1007/s00521-019-04507-z
23. Masdari, M., Khoshnevis, A. A survey and classification of the workload forecasting methods in cloud computing. *Cluster Comput* **23,** 2399–2424 (2020). https://doi.org/10.1007/s10586-019-03010-3
24. Gavagsaz, E., Rezaee, A. & Haj Seyyed Javadi, H. Load balancing in join algorithms for skewed data in MapReduce systems. *J Supercomput* **75,** 228–254 (2019). https://doi.org/10.1007/s11227-018-2578-0
25. Paludo Licks, G., Colleoni Couto, J., de Fátima Miehe, P. *et al.* SMARTIX: A database indexing agent based on reinforcement learning. *Appl Intell* **50,** 2575–2588 (2020). https://doi.org/10.1007/s10489-020-01674-8
26. Hu, H., Wen, Y., Chua, T., & Li, X. (2014). Toward Scalable Systems for Big Data Analytics: A Technology Tutorial. *IEEE Access*, *2*, 652-687.
27. D. E. Diamantis and D. K. Iakovidis, "ASML: Algorithm-Agnostic Architecture for Scalable Machine Learning," in IEEE Access, vol. 9, pp. 51970-51982, 2021, doi: 10.1109/ACCESS.2021.3069857.
28. W. Chen, W. Li and F. Yu, "Modular Pipeline Architecture for Accelerating Join Operation in RDBMS," in IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 67, no. 11, pp. 2662-2666, Nov. 2020, doi: 10.1109/TCSII.2020.2968499.
29. Y. Jiang, "A Survey of Task Allocation and Load Balancing in Distributed Systems," in IEEE Transactions on Parallel and Distributed Systems, vol. 27, no. 2, pp. 585-599, 1 Feb. 2016, doi: 10.1109/TPDS.2015.2407900.