International Journal of Engineering and Advanced Technology (IJEAT)
ISSN: 2249-8958 (Online), Volume-8 Issue-5, June 2019

Predicting Processing Time of Real Time Transaction in Replicated DRTDBS via Middleware

Pratik Shrivastava, Udai Shanker

Abstract*: *The performance of Replicated Distributed Real Time Database System depends on the processing time of Real Time Transaction (RTT) and mutual consistency between replicated data item. Existing research works are kernel based and primarily concentrates on maintaining mutual consistency between replicated real time and non-real time data item. However, research towards predicting the processing time of RTT is still in preliminary stage. Predicting the processing time of RTT involves different factors such as resources/data conflicts between different RTTs, the data requirements of RTT, conventional recovery algorithm, interaction with indeterministic sub-system, and mutual consistency between replicated data. The objective of this paper is to predicts the processing time of RTT with maintaining mutual consistency between replicated data item via middleware. An existing middleware [15] is extended with prediction layer that efficiently predicts the processing time of RTTs and maintains replica consistency. This proposed solution is well suited for different types of real time applications where simultaneously mutual consistency and timeliness demand of RTT must be satisfied. The experimental results show that our proposed approach is beneficial for the RDRTDBS.

Index Terms*: *Mutual Consistency, Prediction, RDRTDBS, RTT.

I. INTRODUCTION

The performance of RDRTDBS depends on the timeliness and mutual consistency [1], and there is always a tradeoff between timely completion of RTT and maintaining mutual consistency. Thus, it is necessary to improve the processing time of RTT such that performance in terms of transaction miss ratio (TMR) get improved. In the current state of art, extensive research work [2-17] is mainly concentrated on the designing and development of an effective and efficient replication protocol. Replication protocol is an algorithm that maintains the mutual consistency between different replicated data item (i.e. real time data or non-real time data) placed in different replica sites. The primary reason behind such research work is to allow admitted RTT to get processed locally because distributed execution of RTT and existence of single version data object cause in majority of times admitted RTT to miss their deadline. Although replication technique increases the performance of the system but utilizing such technique in the system requires judgmental decision. There are different factors that need to be considered before equipping replication technique in the system such as replica size, replica frequency, number of replicas, and soon. Thus, based on application requirement RPL is designed. Although promising work has been done towards maintaining mutual consistency but research towards predicting processing of RTT in RDRTDBS is still in preliminary stage. Very little work [18] has been towards improving processing time of RTT in RDRTDBS. In the current paper, our objective is to propose a processing plan that operates from the middleware to improve the processing time of RTT such that TMR can be improved. Kernel based predictable processing plan of RTT in the RDRTDBS is proposed in [18]. However, such work is unable to work on heterogeneous environment of real time database system and kernel can be optimized up to a certain level. Thus, in place of kernel, our focus is to propose the solution that predictably identify the processing time of RTT in the middleware. This makes the master site to invest all its system resources in processing the RTT such that majority of RTT get competed with in deadline. In paper [15], middleware based RPL is proposed that maintains the mutual consistency of replicated data from the middleware. In the current paper, we extend such middleware with our proposed prediction sublayer that predictably identifies the processing time of RTT and maintains the mutual consistency between replicated data item. Predicting the processing time of RTT involves various factors such as resources/data conflicts between different RTT, the data requirements of RTT, traditional recovery algorithm, interaction with indeterministic sub-system, and mutual consistency between replicated data [18]. Among all such factors, our focus is on two factors: resources/data conflicts between different RTT and traditional recovery algorithm. Resource/data conflict occurs due to presence of single version data objects and during occurrence of conflict between different RTTs, master site cause majority of low priority RTT to miss their deadline. Thus, it is necessary to propose a solution that prevents such issue. In the recently published article [19], a solution is proposed for solving such issue via dynamically creating and deleting multiple version of data objects. This solution is appropriate for RDRTDBS because during occurrence of conflict between different RTTs, multiple versions of data object get created such that simultaneously low priority RTTs with demand of weaker consistency criteria and high priority RTT with demand of strict consistency criteria get completed within their deadline.

Manuscript published on 30 June 2019.
* Correspondence Author (s)
Pratik Shrivastava, Department of CSE, M.M.M.U.T, Gorakhpur, India.
Udai Shanker, Department of CSE, M.M.M.U.T, Gorakhpur, India.

After completion of RTTs these all created multiple versions get converged to a single value such that for future coming RTT single value can be provided by the master site. This finalized value belongs to high priority RTT. Although this solution is better than dual version and multi version data item, but this solution operates in the partially RDRTDBS and is not easy to configure it in the fully replicated RDRTDBS. In fully replicated RDRTDBS it raises unnecessary bandwidth utilization and unnecessary replica updation. Thus, adapting such solution in a system needs judgmental decision. Conventional recovery technique is another issue that really makes the system hard in predicting the processing time of RTT. Thus, in the current paper, our secondary objective is to propose the solution for recovery algorithm that predicts the processing time of RTT more easily. The recovery algorithm proposed in [18] is well suited for RDRTDBS that simultaneously allows the recovering and processing of RTT. In this algorithm master site update the data object whose updated value is urgently needed by the admitted RTT such that admitted RTT is completed within its deadline. Though this algorithm is beneficial, but it operates from the kernel. thus, in this paper we placed this algorithm in the middleware which makes the site to process only RTT. Additionally, recovering load is distributed among available healthy master sites. Healthy master site prepares the partial log and communicate it to the recovering sites such that recovering master site can simultaneously process the RTT and recovers the system from inconsistent state to consistent state. Apart from these issues, our master sites consist of main memory database thus issue of interaction with indeterministic subsystem does not exist in our system. Similarly, issues w.r.t data requirement of RTT is solved via existing solution [20]. At last the issue of maintaining mutual consistency between replicated data item is solved via existing system model [15]. In this system model, the place location for processing different types of RTT is separated in distinct location. For instance, master site processes only update and write RTT whereas slave site processes only process read RTT, and complexity w.r.t to replica consistency management is done via middleware. Middleware itself consists of three sublayer that process different task to maintain mutual consistency. Client submit the request to the middleware, middleware decomposes the RTT into set of operations, check for conflict between newly admitted RTT and existing RTT, and propagates the RTT to the master site. If RTT is of update or write types, then master site will process such RTTs. Apart from update and write RTT, if the RTT is of read type, then master site will forward it to the slave site to get processed. Response generated from slave site or master site is forwarded to the client via middleware. In the current paper, we integrate our proposed prediction layer in the middleware such that predictable processing of RTT can be done in the middleware. The rest of the chapter is organized as follows. Section 2 discuss the factors affecting the predictability of RTT. Section 3 consist of extended system model that predictably predicts the processing time of RTT. In Section 4, solution for unpredictability factors is presented and section 5 outlines the cost function for predicting the processing time of RTT. Experimental result is discussed in section 6 with section 7 concludes the paper.

II. FACTORS AFFECTING PREDICTABILITY

As already mentioned, predictable processing of RTT can be achieved via solving different issues such as resources and data conflicts between RTT, the data requirements of RTT, traditional recovery mechanism, interaction with indeterministic sub-system, and mutual consistency between replicated data. However, our focus is on recovery algorithm and data/resource conflict. For more information on unpredictability factors, authors are requested to read the paper [18].

A. *Recovery Algorithm*

Predicting the processing time of RTT in recovering site is more typical than predicting in healthy sites. Existing recovery algorithm does not allow the processing of RTT until such recovering site get back to the consistent state. Thus, during recovering period admitted RTT must wait until recovering site regains the consistent state. This waiting ultimately causes RTT to miss their deadline which may cause heavy economical loss in the system. Thus, it is necessary to propose a solution which simultaneously allows processing of admitted RTT and recovery algorithm. In paper [18], author has proposed a recovery solution that distribute the load of recovery among existing healthy site such that healthy site prepares the partial log and forward to the recovering site for recovery. In the current paper, an existing middleware is extended with such an algorithm such that middleware identifies the number of healthy sites, distribute the load of partial log among healthy sites, and healthy site propagate their prepared log records to the recovering site.

B. *Resource/Data Conflict Between Different RTT*

In RDRTDBS, the performance of the system depends on timely completion of RTT. However, during occurrence of conflict, conflict resolution algorithm favors for high priority RTT which ultimately cause low priority RTT to miss their deadline. This abortion of low RTT cause wastage of resources. Thus, it is necessary to propose some mechanism that simultaneously allows the processing of high and low priority RTT. In the recently published paper [19], a solution is proposed that dynamically creates different version of data objects. This dynamic version of data objects allows low priority RTT with weaker consistency criteria to get processed. Similarly, high priority RTT also get processed using a particular version of data object. In the current paper, slave site is equipped with such a solution such that RTT processing in slave site get completed within the deadline. In addition to this, dynamic version of data object gets converge to single value which will ultimately cause efficient utilization of the storage space.

III. EXTENDED SYSTEM MODEL

Existing RPLs [2-17] have been designed and tested in their respected simulation environment. The arrival of RTT in their respective simulation environment follows the Poisson stream. In addition to this, these admitted RTTs consist of a set of read and write operations that processes the real time and non-real time data item.

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication (BEIESP)

However, these simulation environment lacks from predicting the processing time of RTTs. In the current paper, an existing system model [15] is extended with our proposed prediction layer such that processing time of update/write/read RTTs can be easily predicted.

A. *Data Model*

Our extended system model consists of two types of data items: real time data item and non-real data item. Real time data is linked with a periodic interval whose value remains consistent until the periodic interval get expired. Thus, update RTTs are executed periodically to maintain the temporal consistency of such data item. Similarly, write RTTs are executed to update the non-real time data item. This RTTs are executed periodically or non-periodically to maintain consistency of non-real data item. Read RTT is different from update and write RTT. This RTT is used to access the value of real time and non-real data item.

B. *RTT Model*

Our extended system model processes three types of RTTs: update RTT, write RTT, and read RTT. These RTTs are categorized based on their property value and information availability. The properties of RTT are in the form of arrival time, priority value, response time, execution time, value function, slack value, data requirement, and circulation (i.e. periodic, non-periodic, or random). Based on property value, RTT is given by.

1. Update RTT: Update RTTs are periodically executed and have strict requirement to meet deadline. In order to fulfill the strict requirement of meeting deadline, these RTTs are always assigned high priority value such that among different types of RTTs, update RTTs are always processed first. In addition to this, the data requirements of such RTTs are predefined and are used to maintain the temporal consistency of real time data item. However, during occurrence of conflict, low priority always miss deadline. Thus, it is necessary in RDRTDBS to design efficient conflict detection and resolution mechanism. In our extended system model, this issue is solved via separating the place location of processing write/update RTT and read RTT in master site and slave site respectively.

2. Write RTT: Write RTTs are used to update non-real time data items and their arrival pattern is periodic or non-periodic in nature. The data requirement of such RTT is also predefined. These RTTs does not have a strict requirement to strict consistency. However, during missing the deadline of write RTT, it leaves no value for the RDRTDBS. Thus, it is immediately aborted by the system such that waiting RTT get the opportunity to meet their deadline.

3. Read RTT: Read RTTs are initiated by the user to read the value of real time and non-real time data item. The arrival pattern of read RTT is periodic or non-periodic in nature. Due to the limitation of system resources, these RTTs are assigned low priority value as compare to update RTT.

IV. PROPOSED SOLUTION

As already mentioned in our previous section, that predicting the processing time of RTT depends heavily on recovery algorithm and data/resource conflict. Thus, in this our objective is to propose the solution for such mentioned issue.

A. *Proposed Recovery Algorithm*

In RDRTDBS, database site gets failed from any type of failure such as media failure, system failure, and transaction failure. To recovery from such type of failures, recovery algorithm is processed to regain the system from inconsistent state to consistent state. However, during processing of recovery algorithm, recovering site does not allows processing of admitted RTT which will ultimately cause admitted RTT to miss their deadline. Thus, it is necessary to design an efficient and effective recovery algorithm that simultaneously allow processing of both admitted RTT and recovery algorithm. In our extended system model, recovery algorithm operates from the middleware. Primarily, recovery algorithm identifies the number of failed sites and healthy sites. After identifying the number of failed and healthy sites, recovery algorithm distributed the load of preparing the partial log among existing healthy sites. Secondary, healthy site will prepare partial log and communicate to the recovering site. Recovering site initiates the process of recovery from urgently needed log such that admitted RTT with high priority get completed within its deadline. The main reason for distributing the recovery load among existing healthy sites is because existing recovery techniques are one to one process that engage both the recoverer site and recovering site. This will ultimately cease the processing of admitted RTTs which will ultimately cause heavy loss in the system. That's why, our proposed recovery algorithm distributed the load of recovery among existing healthy sites such that the load of recovery distributes among the healthy sites. The flow chart of processing recovery algorithm is given in Fig. 1.

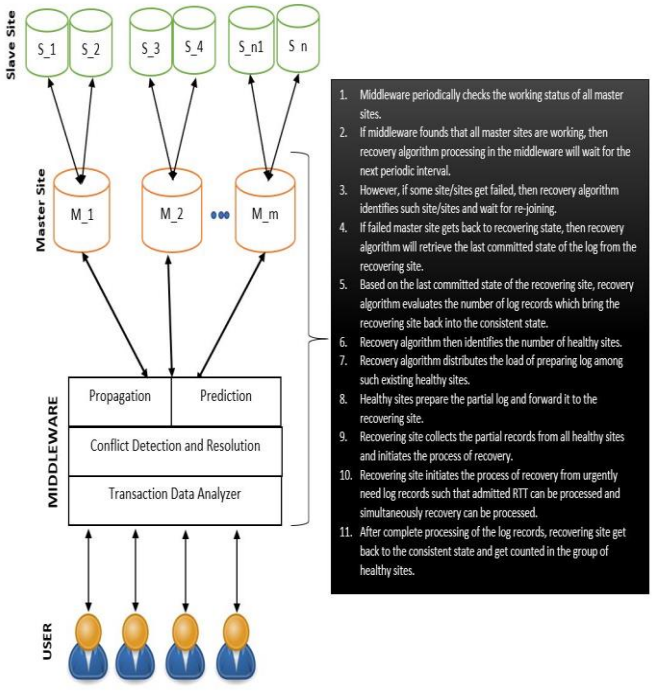

Figure 1. Flow Chart of Recovery Algorithm

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication (BEIESP)

B. *Solution for Data/Resource Conflict*

In RDRTDBS, RTTs are processed serially or concurrently. Serial processing of RTTs allows only one RTT to get processed. Thus, occurrence of conflict between different RTTs is NIL. Although serial is free from conflict, but it is not suitable for real time environment where RTTs are of short duration and missing the deadline of RTT cause heavy loss. In compare to serial processing of RTTs, concurrent processing of RTTs allows simultaneous processing of different RTTs. This concurrent processing of different RTTs increase the performance of the system as compared to RTTs processed in serial manner. Although concurrent processing of RTTs is beneficial for real time environment but it suffers from the issue of conflict. Conflict is the condition where more than one RTTs compete for the same resource or data at a time. During such an instance only one RTT can processed and another RTT must get abort. This abortion is not acceptable in the real time environment. Thus, it is necessary to develop a solution that solves the issue of data/resource conflict more efficiently and effectively. In existing paper [3], a solution is proposed to solve the issue of data/resource conflict via using dual version data object. Dual version data object stores two values (i.e. before value and after value). This dual version concept allows two RTT to get processed simultaneously at a time. Similarly, a concept of multiversion data object is proposed in [21]. This multiversion concept allows the processing of multiple RTTs simultaneously at a time. Although multiversion concept is better than dual version data object but is not space efficient. Recently in paper [19], a concept of dynamic version data object is proposed which dynamically creates the different version of data object and finally converge to a single value. This dynamically creation of different versions of data object is processed in the master site despite of middleware. Let us consider a scenario to understand the use of dynamic version of data objects in master site. Suppose one read RTT (i.e. T_{R1}), one update RTT (i.e. T_U), and one write RTT (i.e. T_W) is admitted in the master site to get processed and their set of operation is given by.

T_{R1}: R(RT1), R(NRT1), R(RT2).
T_U: R(RT1), W(RT1).
T_W: R(NRT1), W(NRT1).

Where R and W stands for read and write operation respectively. Now at time instant t1, T_{R1} and T_U is admitted in the master site to get processed. Master site will forward T_{R1} to the least loaded slave site such that T_{R1} get meet its timeliness property. During processing of T_{R1}, if T_{R1} follows weaker consistency criteria, then T_{R1} get processed via creating their new version of data object and get commit. However, if T_{R1} follows strict consistency criteria, then in such a case T_{R1} is processed, but the outcome of T_{R1} (i.e. commit or abort) will depends on the high priority RTT T_U whom is also updating the same data member. If this high priority RTT T_U is committed, then as per strict correctness criteria T_{R1} get abort. However, if exceptionally T_U get abort then T_{R1} get the chance to get commit. Overall, via using dynamic versioning of data object in the RDTDBS, wastage of resources or data object can be prevented.

V. ESTIMATION OF COST FOR PREDICTING THE PROCESSING TIME OF RTT

As already mentioned, that in the current paper, our main objective is to predict the processing time of different RTTs in the middleware. Thus, to predict the processing time of RTT, cost of processing RTT in the master site is evaluated. If the decision parameter predicts that the processing time of admitted RTT cross the defined limit, then such admitted RTT get immediately aborted. however, if such admitted RTT does not cross the defined limit, then such admitted RTT can get proceed for the processing in the master site. The cost evaluation of different RTT is given by.

A. *Cost Prediction for Update RTT*

The data requirement of update RTT is predefined and is always assigned high priority value such that update RTT get completed within its deadline. The reason for assigning high priority value is because missing the deadline of update RTT cause heavy disaster in the system. The cost evaluation for update RTT is done from the far distance master site. The reason for selecting such master site is because if the evaluated cost of the update RTT is satisfied, then it must satisfy for all master sites which are at shorter distance. The formula for cost evaluation is given by.

$$C_{TOTAL} = (T_{INIT} + TSDC_{COST} * M + T_{CLOSE}) + NWC_{COST} \quad (1)$$

Where C_{TOTAL} is the total computation cost, T_{INIT} is the RTT initialization cost, $TSDC_{COST}$ is the read/write operation cost, M is the total number of operations, T_{CLOSE} is the closing time of RTT, and NWC_{COST} is the communication cost. This cost is evaluated for each RTT. If the cost of admitted update RTT crosses the absolute validity interval of real time data item, then in such a case update RTT is aborted. In order to further improve the issue of aborting update RTT, we leave this work for the future researchers.

B. *Cost Prediction for Write RTT*

The data requirement of write RTT are also predefined and missing the deadline of such RTT leaves no value for the system. Thus, this type of RTT get immediately aborted on missing its deadline. Scheduler assign the low priority value to this type of RTT as compared to update RTT. The cost evaluation for write RTT is given by.

$$C_{TOTAL} = (T_{INIT} + TSDC_{COST} * M + T_{CLOSE}) + NWC_{COST} \quad (2)$$

C. *Cost Prediction for Read RTT*

Read RTT is used to access the data value of temporal and non-temporal data item. The data requirement of such RTT is unknown and these type of RTTs is processed by the slave site. Master receives read RTT from the middleware and forwards it to the least loaded slave site such that timeliness property of read RTT get easily satisfied. On missing the deadline of such RTT leaves some value for the system. Thus, this type of RTT is not immediately aborted. the cost evaluation of read RTT is given by.

$$C_{TOTAL} = (T_{INIT} + TSDC_{COST} * M + T_{CLOSE}) + MC_{COST} + MS_{COST} \quad (3)$$

Where MC_{COST} and MS_{COST} stands for communication cost from middleware to the master site and communication from master site to the slave site respectively.

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication (BEIESP)

If the CTOTAL crosses the time limit of read RTT, then read RTT is not allowed to proceed to get processed. However, if this is not the case, then read RTT is forward from the master site to the slave site to get processed.

VI. EXPERIMENTAL RESULT

In order to check the performance of our proposed solution for predicting the processing time of RTT, we have implemented the proposed solution of prediction in the java programming language and compared it with the other processing plan proposed in [18]. This existing solution [18] is based on locking and locking allows only one RTT to get processed. Remaining RTTs must wait to get the lock for accessing the data value. The primary performance metric used to check the performance of the system is TMR. TMR is termed as the percentage of RTTs that have not successfully meet their deadline. This is the traditional performance metric and is mostly used by all the researchers to check the performance of the system. TMR is given by.

$$TMR = DM_{RTT} / (SC_{RTT} + DM_{RTT}) * 100 \quad (4)$$

Where DM_{RTT} stands for RTTs that have missed their deadline and SC_{RTT} stands for RTTs that have successfully meet their deadline.

A. *Cost for Predicting the Processing Time of Update RTT*

Fig. 2 and Fig. 3 shows the experimental result of update and write RTTs respectively. These RTTs are scheduled by different scheduling techniques. Additionally, Fig. 2 also contains the result of conventional scheduling algorithm such as Spare Capacity Finding (SCF)/Guaranteed algorithm, EDF/Blind algorithm.

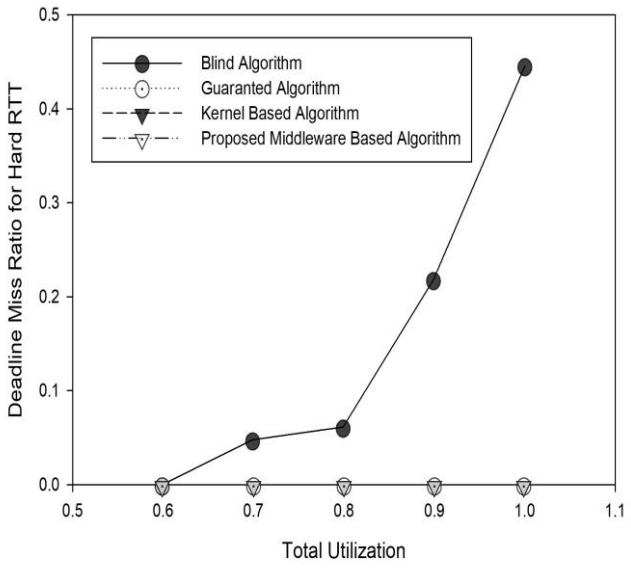

Figure 2. TMR for Update RTT.

B. *Cost for Predicting the Processing Time of Write RTT*

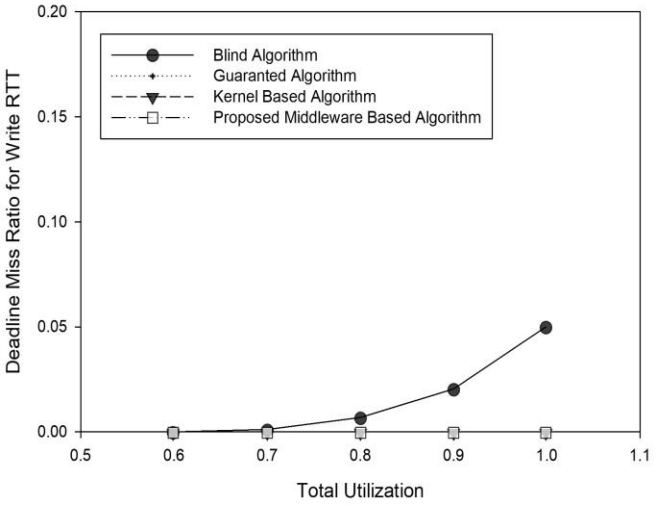

Figure 3. TMR for write RTT.

C. *Cost for Predicting the Processing Time of Read RTT*

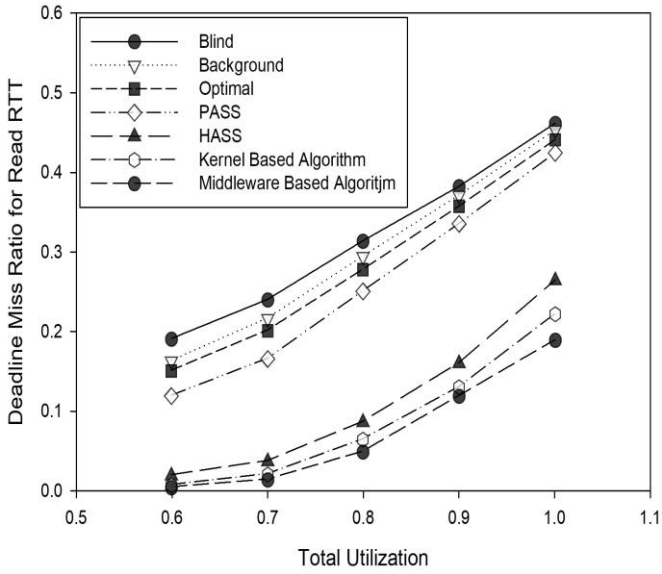

Figure 1. TMR for Read RTT

Fig. 3 shows the TMR of read RTT which is processed in the slave sites. In Fig. 3 experimental result of exiting algorithms are included for clear understanding. From the experimental result, a conclusion can be drawn that predicting processing of read RTT in our extended system model is better than kernel-based solution.

VII. CONCLUSION

In the current paper, our primary objective is to predict the processing time of RTT in the RDRTBDS. This objective is accomplished via integrating our proposed prediction layer in the existing middleware. Our proposed prediction layer involves the solution of two unpredictability factors: conventional recovery algorithm and data/resource conflict between different RTTs. The decision parameter for deciding that whether admitted RTT can be processed or not is also discussed in the cost evaluation section.

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication (BEIESP)

Experiments are conducted to verify that our proposed approach is better than existing approach and the result confirm that middleware-based solution is better than kernel-based solution. Although our proposed approach is better, but there is path in which research is more needed such that performance of the system can get further improved.

REFERENCES

1. Yu, Philip S., et al. "On real-time databases: Concurrency control and scheduling." Proceedings of the IEEE 82.1 (1994): 140-157.
2. Son, Sang Hyuk. "Using Replication for High Performance Database Support in Distributed Real-Time Systems." RTSS. 1987.
3. Son, Sang H., and Spiros Kouloumbis. "A token-based synchronization scheme for distributed real-time databases." Information Systems 18.6 (1993): 375-389.
4. Son, Sang Hyuk, and Fengjie Zhang. "Real-Time Replication Control for Distributed Database Systems: Algorithms and Their Performance." DASFAA. Vol. 11. 1995.
5. Son, Sang H., Fengjie Zhang, and Buhyun Hwang. "Concurrency control for replicated data in distributed real-time systems." Journal of Database Management (JDM) 7.2 (1996): 12-23.
6. Kim, Young-Kuk. "Towards real-time performance in a scalable, continuously available telecom DBMS." (1996).
7. Xiong, Ming, et al. "MIRROR: A state-conscious concurrency control protocol for replicated real-time databases." Information systems 27.4 (2002): 277-297.
8. Peddi, Praveen, and Lisa Cingiser DiPippo. "A replication strategy for distributed real-time object-oriented databases." Proceedings Fifth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing. ISIRC 2002. IEEE, 2002.
9. Gustavsson, Sanny, and Sten F. Andler. "Real-time conflict management in replicated databases." Proceedings of the Fourth Conference for the Promotion of Research in IT at New Universities and University Colleges in Sweden (PROMOTE IT 2004), Karlstad, Sweden. Vol. 2. 2004.
10. Gustavsson, Sanny, and S. R. Andler. "Continuous consistency management in distributed real-time databases with multiple writers of replicated data." 19th IEEE International Parallel and Distributed Processing Symposium. IEEE, 2005.
11. Syberfeldt, Sanny. Optimistic replication with forward conflict resolution in distributed real-time databases. Diss. Institutionen för datavetenskap, 2007.
12. Haj Said A., Sadeg B., Amanton L. and Ayeb B. "A Protocol to Control Replication in Dis-tributed Real Time Database Systems". In Proceedings of the Tenth International Confer-ence on Enterprise Information Systems Volume 1: ICEIS (2008), ISBN 978 989 8111 36 4, pages 501 504.
13. El-Bakry, Hazem M., and Torky Sultan. "Design of replicated real-time database simulator." Proceedings of the 6th WSEAS International Conference on Computer Engineering and Applications, and Proceedings of the 2012 American conference on Applied Mathematics. World Scientific and Engineering Academy and Society (WSEAS). 2012.
14. Mathiason, Gunnar, Sten F. Andler, and Sang H. Son. "Virtual full replication by adaptive segmentation." 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2007). IEEE, 2007.
15. Shrivastava, Pratik, and Udai Shanker. "Replica control following 1SR in DRTDBS through best case of transaction execution." Advances in Data and Information Sciences. Springer, Singapore, 2018. 139-150.
16. Shrivastava, Pratik, and Udai Shanker. "Real time transaction management in replicated DRTDBS." Australasian Database Conference. Springer, Cham, 2019.
17. Shrivastava, P., and U. Shanker. "Replica update technique in RDRTDBS: issues & challenges." Proceedings of the 24th International Conference on Advanced Computing and Communications (ADCOM-2018), Ph. D. Forum, Bangalore, India. 2018.
18. Shrivastava, Pratik, and Udai Shanker. "Supporting transaction predictability in replicated DRTDBS." International Conference on Distributed Computing and Internet Technology. Springer, Cham, 2019.
19. Shrivastava, Pratik, and Udai Shanker. "Replication protocol based on dynamic versioning of data object for replicated DRTDBS." International Journal of Computational Intelligence & IoT 1.2 (2018).
20. O'Neil, Patrick E., Krithi Ramamritham, and Calton Pu. "A Two-Phase Approach to Predictably Scheduling Real-Time Transactions." (1996): 494-522.
21. Son, Sang Hyuk. "Synchronization of replicated data in distributed systems." Information Systems 12.2 (1987): 191-202.