An Evaluation Of Cs-Rtdbs Over Ce-Rtdbs And Improving Its Performance

Amit Sinha
Ph.D(CS/IT) Scholar, SHAITS
(Formerly Allahabad Agricultural Institute)
Deemed University, Allahabad

Dr. R.K. Isaac
Professor, CET, SHAITS
(Formerly Allahabad Agricultural Institute)
Deemed University, Allahabad

ABSTRACT

In this paper, we reviewed the behavior of response time in various situation of real time transaction. The centralized real time databases (CE-RTDB) and client server real time databases (CS-RTDB) are the two areas on which real time transactions can be taken place. Till date, the real time transaction with time constraint has been analyzed according to scheduling of the transaction. Here we studied and then have some experiments for real time transactions on the basis of CPU usage, processor consumption and number of transaction performed background and foreground of the system. Our objective is to investigate the dependence of efficiency of real time processing in centralized database and then in client server database. The graphical representation of the experiments performed on centralized database can give an idea of behavior of the real time transaction and hence can be used to reduce the response time.

1. INTRODUCTION

In real-time databases, a transaction, also called sometimes real time transaction, completes successfully only if it finishes its execution within a pre-specified deadline. Deadlines are a mandatory part which is used to satisfy quality of service requirements or control the operation of physical systems. Therefore, the key measure of performance is the percentage of all transactions that complete within their deadlines rather than the average transaction response time or throughput.

Alternatively, the key issue of performance in a system is response time. In any transaction, good response time implies sub second response time. The reduced response time is required for good performance of a system. It depends on a variety of factors including system architecture, transaction configuration, user requirements and system reliability etc. It is necessary to note that set of constraints performing good performance for one system may be poor performance for another system.

If, for example, a transaction requires 200 accesses to a database, a response time of two to four seconds may be considered to be quite good. On the other hand, another application requires only ten accesses and result into a response time of three to six seconds. So accesses would need to be investigated. Response times, however, depend on the speed of the processor, and on the nature of the application being run on the production system.

A generic (i.e. hard or soft) RTS can be describes as consisting of three principle subsystems as shown below

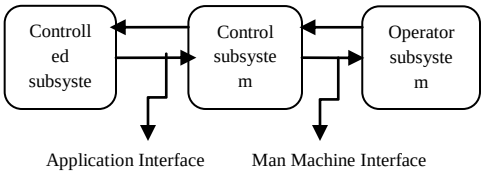

Figure1: a RTS Organization

The controlled subsystem represents the application or environment which indicates the real time requirements; the control subsystem controls some computing and communication equipment for use from the controlled subsystem; the operator subsystem initiates and monitors the entire system activity. The interface between the controlled and the control subsystems consists of devices such as sensors and actuators. The interface between the control subsystem and the operator consists of a man machine interface.
Successful completion of an operation in a RTS depends upon the correct and timely operation of the system. So, we need to design the hardware and software that is to be used in the system in such a way so that it might meet the RT requirements of the entire system. The designers need to focus very early on the RT response requirements.

Each database transaction usually involves to several data items, using which it carries out necessary processing. The transactions that are active at any time are called concurrent transactions. For the purpose to achieve non-interference among transactions, concurrency control schemes normally restrict concurrent transactions to be serialzable. Hence, a RTS must be designed in such a way that concurrency protocols should allow several transactions to access the database concurrently, but leave the database consistent by enforcing seraliazability. Techniques to provide real-time transaction scheduling include priority-driven techniques, earliest-deadline-first techniques [12], and hybrid scheduling techniques [13]. Concurrency control techniques must also be adjusted for the requirements of real-time. Typically these techniques relax serializability as the concurrency control correctness criteria through lock-based techniques [14], optimistic techniques [15], and semantic concurrency control [16]. Real-time database test beds have been developed for military applications, and in academic settings, but are not currently used widely in commercial applications.

2. RELATED WORK

The real time processing is time based and it depends on various constraints such as the processor speed, size of the RAM, network congestion and bus bandwidth.

Some results have been observed [1] that a Client Server Real Time Data Base system (CS-RTDBS) can be more efficient than a centralized system in the presence of the following conditions:

(i) if there is a reasonable amount of spatial and temporal locality in client data access patterns, and
(ii) if the percentage of data accesses with respect to updating is low.

The important requirement of a traditional DB system is to provide fast average response time while Real-time database systems (RTDBS) may be evaluated based on the number of transactions miss their deadline [2], the average late transactions, the cost incurred in transactions missing their deadlines. Therefore, in RTDBS, transactions should be scheduled according to their criticalness and the tightness of their deadlines, even if this means sacrificing fairness and system throughput. Also it always must guarantee preceding process of a high priority transaction (HPT) than low priority transaction (LPT). If HPT is eliminated in a system because of its deadline missing, an unnecessary aborting or blocking of LPT is occurred. To resolve the problem, AVCC (alternate version concurrency control) algorithm was proposed. However, AVCC must always create the alternative version and have additionally a technique to manage complex alternative versions. A new and efficient scheduling algorithm, called Multi-level EFDF (earliest feasible deadline first) that combines EFDF and Multilevel Queue scheduling algorithm very ably and achieves good performance over the other existing methods proposed earlier.

Earliest deadline first (EDF) is one of the main scheduling policies used in real-time database systems (RTDBSs) for transactions processing. With EDF, prioritized transactions are not necessarily the most important in the system. Moreover, it is well-known that EDF is not efficient in overload conditions. [3] Another notion of transaction importance and a new priority assignment technique based on both transactions importance and deadlines is present. This assignment policy leads to a new scheduling policy, called generalized earliest deadline first (GEDF).

Although, there has not been much research into this specific area, there has been considerable interest in two closely related areas, viz, client server information and database systems, and active rule based systems. Delis and Roussopoulos [5] examine the problem of managing server updates that affect client cached data. They introduce five possible strategies. In the simplest strategy, updates are sent to clients only on demand while in more complicated ones the server maintains a catalog of binding information which designates the specific areas of the database that each client has cached. Keller and Basu [6] introduce the concept of predicate-based client-side caching. Client queries are executed at the server and the results are used to load the client cache. The contents of client caches are described by means of predicates. If a client determines from its local cache description that a new query is not computable locally then the query (or a part of it) is sent to the server for execution. Otherwise, the query is executed on the cached local data. All the above described protocols deal with the issue of providing relatively up-to-date data accesses while providing shorter query response times, lower data transfer costs and reduced network contention. The other important related area is Active Database Systems which allow database designers the ability to define rules that make the DBMS.

3. ISSUES IN CE-RTDBS and CS-RTDBS

In the CS-RTDBS, the object-server executes one thread per client. Each such thread maintains two persistent socket connections with the client for the entire duration of the experiment. One connection is used exclusively for messages, while the other is used for the transfer of database objects to and from the client. Here too, like the CE-RTDBS, each transaction is executed as a separate thread, but the distinction is that transactions at each client are scheduled locally, independent of the server. The clients also make use of the short and the long term memory available to them [4]. Data object / locks that have been fetched from the server are maintained in the local cache so that the future requests on the cached data can be satisfied without interaction with the server.

Sometimes no change in existing working system (Centralized RTDBs) is required. However, it needs to increase its efficiency. Also few situations may occur where centralized RTDBs give reasonable performance. *For example*, If the environment requires a frequent and larger update then centralized systems demonstrate better performance than their client-server counterparts. This is because a large volume of updates increases the overhead incurred in coordinating distributed concurrency and shipping objects among sites significantly. Thus, transactions at client sites are forced to block for long periods of time waiting for their required data objects and locks to become available.

3.1 STUDY OF RESPONSE TIME WITH INCREASING LOAD TRANSACTION

A graph can be plotted between response time and increasing load of transactions. However this dependency is also possible in centralized

data bases, we have analyzed in CS-RTDBS. The response time in the system, obviously, may vary with increasing transaction rate. But the study shown by the curve exhibits it increases gradually at first then deteriorates rapidly (i.e. response time increases suddenly). The typical curve shows a sharp change when the response time increases dramatically for a relatively small increase in the transaction rate.

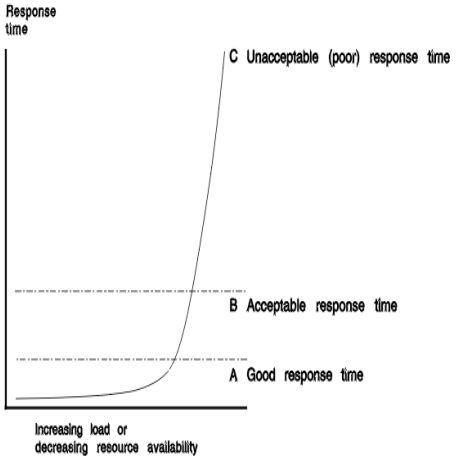

Figure2: Effect of response time versus increasing transaction load

4. TECHNIQUES FOR IMPROVING REAL-TIME PROCESSING

This section describes the techniques that we have used in our load-sharing algorithm such as transaction-shipping [9], transaction decomposition [19], object migration, and object request scheduling.

In a non real-time environment, the order in which client object requests are served is often First-Come First-Serve. However, in a real-time system, object requests can be prioritized according to the deadlines of the requesting transactions. Object requests by clients can convey the deadline information of the requesting transactions. Requests by transactions with earlier deadlines are satisfied before others. If a client transaction has already missed its deadline then the server can unilaterally decide not to ship the object to that client. In fact, if reasonably accurate estimates of transaction execution times were available, the server could decide whether to satisfy certain object requests at all viz. requests by transactions that are expected to miss their deadlines.

In this paper, we have taken the case of object request scheduling. Various object requests by clients can come across to the server and accordingly these requests can be satisfied. In a real time system, these objects also carry the deadline information of the requesting transactions. Since the RT- processes are ensured to finish its job within its time constraints imposed from outside the computer therefore these processes are used to handle time-driven materials, e.g., audio, video, games, robot controllers and so far.

4.1 PROPOSED ALGORITHM FOR OBJECT REQUEST SCHEDULING

We have taken CS-RTDBS as a sample case where the transactions can be processed through the objects available at client side (local service) or at the server side (global service). The following algorithm gives an idea of handling of various clients' requests at the same time. We have not considered some of the problems like network congestion and BUS bandwidth.

Step-1: Real time transaction T is initiated at a client side.

Step2: *If*
T can be processed at local host with a reasonable chance of meeting its deadline
Then
T is appended in local queue i.e. local cache / disc memory and processed accordingly.

else
(T needs to be processed at server end)
follow either step-3 or step-4

Step-3: *If*
server can provide objects to T
Then
these objects are locked appropriately w.r.t. the concerned client and process it.

Step-4: *If*
server is unable to process T immediately
Then
T will be appended in server queue and picked another client's request that is in better position.

4.2 EXPERIMENT RESULT

We have taken a case of Intel core ™ 2 duo CPU T5670 1.80 GHz at the server side and PIV processor 631 3.0 GHz at the client side. We have considered one transaction from one client. The total consumption of processor at server side is measured with the same transaction repeatedly few times.

Case

The request size is 5KB

CPU consumption before the transaction = 2%

Processor consumption before the transaction = 52%

Table1: Processor utilization w.r.t. to the same transaction several times

Turn of the same transaction	Processor consumption	
	Before the service	After the service
1	93%	300%
2	109%	353%
3	109%	262%
4	125%	615%
5	109%	300%
6	109%	276%
7	94%	239%
8	78%	274%
9	93%	388%
10	109%	282%

The above values are taken as a particular case and specific purpose of study. The values may be changed in different scenarios.
The equivalence graphic notation is shown in figure2.

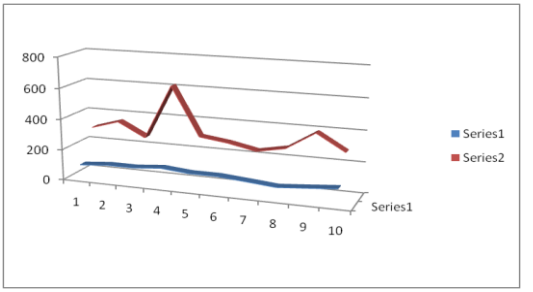

Figure3:Processor utilization before& after the execution of the transaction

The graph states that there is one highest value, say P_{max} (max. consumption of processor). The value of P_{max} can help in evaluating the performance of CS-RTDBS. If the consumption of processor is below P_{max} , some real time transactions can be allocated to processor to improve the performance.

5. CONCLUSION

In this paper, we studied various issues relating to RTS. Two different databases CE-RTDBS and CS-RTDBS are associated with real time transactions. A graphical representation shows the importance of these two in different scenarios. We propose the usage of the client-server database paradigm for handling requests with time constraints. The resulting configuration takes advantage of the available client resources and schedules requests locally. The proposed algorithm can explain the allocation of processor to the client's requests available in waiting queue.

The experimental result also gives an idea of maximum value of processor utilization. This P_{max} can be used in performance evaluation.

REFERENCES

1. V. Kanitkar and A. Delis, A Case for Real-Time Client-Server Databases, In *Proceedings of the 2nd International Workshop on Real-Time Databases*, September 1997, p121-132.
2. Seok Jae Lee Jae Ryong Shin Seok Il, Song Jae Soo YooKi Hyung Cho, A concurrency control algorithm for firm real-time database systems, Proceeding ICCS'03 Proceedings of the 2003 international conference on Computational science Springer-Verlag Berlin, Heidelberg 2003, p431-438.
3. Samy Semghouni, Laurent Amanton, Bruno Sadeg, Alexandre Berred, On new scheduling policy for the improvement of firm RTDBSs performances, Journal Data & Knowledge Engineering, Elsevier Science Publishers B. V. Amsterdam, The Netherlands, Volume 63, Issue 2, p176-188, November, 2007.
4. A Delis and N. Roussopoulos, "Performance Comparison of Three Modern DBMS Arcitectures", IEEE Trans. Software Eng., volume 19, No.2, pp. 120-138, Feb. 1993.
5. A. Delis and N. Roussopoulos, Management of Updates in the Enhanced Client–Server DBMS, In *Proceedings of the 14th IEEE International Conference on Distributed Computing Systems*, June 1994.
6. A. Keller and J. Basu. A Predicate–based Caching Scheme for Client-Server Database Architectures. *The VLDB Journal*, 5(1), 1996.
7. R. Abbott and H. Garcia-Molina. Scheduling Real–Time Transactions: A Performance Evaluation. *ACM–Transactions on Database Systems*, 17(3), 1992.
8. R. Alonso, D. Barbara, and H. Garcia-Molina. Data Caching Issues in an Information Retrieval System. *ACM–Transactions on Database Systems*, 15(3):359–384, September 1990.
9. V. Ballingam, K. Christensen, and F. Noel. Analysis of Client/Server Transaction Delay through a Local Area Network Switch. In *Proceedings of SOUTHEASTCON 1996*, pages 571–577, Tampa, FL, USA, April 1996.
10. M. Carey, M. Franklin, M. Livny, and E. Shekita. Data Caching Tradeoffs in Client–Server DBMS Architecture. In
11. *ACM–SIGMOD–Conference on the Management of Data*, Denver, CO, May 1991.
12. Harista ,J; Livny, M; Carey, M. Earliest deadline scheduling for real-time database systems. *Proceedings of the IEEE Real-Time Systems Symposium* (1990) 11, 94--103.
13. Abbott, R., Garcia-Molina, H. Scheduling real-time transactions: A performance evaluation. *Proceedings of the Very Large Database Conference* (1988), p 14, 203-210.
14. Sha, L; Rajkumar, R., Son, S; & Chang,C. A Real-Time Locking Protocol. *IEEE Transactions on Computers*, 40(7), p793-800. (1991).
15. Huang, J., Stankovic, J; Towsley, D; Ramamritham,K (1991). Experimental Evaluation of Real-Time Optimistic Concurrency Control Schemes. *Proceedings of the Very Large Database Conference*, 17, 110-118.
16. Garcia-Molina, H. Using Semantic Knowledge For Transaction Processing in a Distributed Database System *ACM Transactions on Database Systems*, 1993, 8(2), p186-213.