Journal of Computer Science 1(4): 495-499, 2005
ISSN 1549-3636
© 2005 Science Publications

Adaptive Update Policy for Proactive Management of Deadline Miss Ratio and Data Freshness in Real-Time Database Models

Sanjay Tanwani and Ashwani Kumar Ramani
School of Computer Science, Devi Ahilya University, Indore, India

Abstract: Important applications, like e-commerce, online stock trading, traffic control demand real-time data services. Conventional database perform poor at these applications. A database for real-time data services has to support timing constraints and temporal consistency in addition to supporting characteristics of a conventional database system. In other words, it is desirable to execute transactions within its deadline using updated data reflecting the current world status. In order to achieve these objectives, the Quality of Service (QoS) metrics like miss ratio, data freshness, perceived freshness and Quality of Data (QoD) management schemes along with architectures for effective QoS management are introduced in Literature. These architectures accommodate the unpredictable change in user transactions and maintain QoS metrics by changing the update pattern of data items, which are periodically updated to reflect the real world status. In this study, we present architecture with an improved update policy and propose an algorithm to change the update pattern in such a manner that the system is able to maintain QoS metrics even under busty traffic conditions. The simulation results show that the proposed model can keep QoS metrics, like miss ratio and perceived freshness within limits under busty traffic conditions [1].

Key words: Real-time database, miss ratio, perceived freshness

INTRODUCTION

A real time database is used in important applications, like e-commerce, online stock trading and traffic control requiring stringent timing constraints for completion of tasks. It is difficult for the conventional databases to support timing constraints associated with transactions in addition to concurrency, atomicity and consistency properties. The transaction timing constraints can be like, completion deadlines, start times, periodic invocations and so on. Real-time data such as, sensor data, stock market prices and location of moving objects are valid for a specified interval. A real-time database system has not only to be fast, but also has to execute transactions under specified timing constraints maintaining the value(s) of data item(s), reflecting the current real world state and performing transactions within their deadlines[2].

In a fixed and predictable environment, executing transactions within deadline and maintaining data items in database reflecting the real-world state is feasible with off-line scheduling. The challenge is to maintain this correctness in unpredictable and varying load conditions. Examples include air traffic control, autonomous vehicles and missile control systems, which operate in nondeterministic and fault-inducing environments under severe time constraints. This requires dynamic solutions and robust real-time databases delivering real-time performance under unpredictable and bursty traffic conditions.

The application of QoS aware approaches in systems can improve the performance in a cost-effective manner. Vast literature is available in QoS related research as well as database research separately. However, limited work is done on QoS issues in Databases. This is a serious problem as approximately $420 million was lost due to late non real-time e-commerce transaction processing in 1999[3]. Work has been done in the area of auto tuning & auto configuration so as to improve performance of database systems. Research at IBM is aimed to support auto configuration at database through physical database design[4,5]. Microsoft is working on a database self-tuning project, aimed to reduce the cost of manual database tuning needed to support the required database performance for specific applications[6]. However, these approaches do not consider QoS guarantee issues in terms of both timeliness and freshness.

An architecture for QoS management under varying and unpredictable load conditions is proposed by Kang *et al.*[7]. The important QoS metrics, like, miss ratio, perceived freshness and CPU utilization are monitored at regular intervals. If the user transaction load increases and any of the QoS metrics is not found within acceptable limits, a feedback mechanism is used to reduce the update load by changing the update policy. One of the update policy changes the workload

Corresponding Author: Sanjay Tanwani, School of Computer Science, Devi Ahilya University, Indore, India

by updating few less frequently accessed data items only on demand. The problem with this approach is that the perceived freshness of the system may be affected. Another update policy incrementally changes the update period of less frequently accessed data items. The scheme works well for regular traffic patterns. However, the incremental change in update period fails to reduce update workload and accommodate abrupt change in traffic conditions in order to maintain QoS metrics within limits. Therefore, as pointed by the author[7], these policies fail for busty traffic conditions.

In this study, a modified architecture is proposed for maintaining QoS metrics within limits under unpredictable and varying load conditions, to improve the transient performance and therefore eliminating the problems mentioned above. Along with monitoring the metrics, like miss ratio and perceived freshness, the weight of transactions (summation of estimated execution times) in queue is also monitored[8]. An increase in queue size is indicative of load increase and may increase the miss ratio in near future. If the weight of transactions in queue exceeds a threshold limit, feedback controller is activated to change the update policy. We propose an improved update policy, where increase in workload monitored by CPU utilization is appropriately mapped to the required change in update period rather than incrementally changing the update period as proposed[7]. By calculating the required change in update period, QoS metrics are maintained within limits even under busty traffic resulting in significant reduction in number of deadline misses and freshness violations. The effectiveness of our approach is demonstrated with the help of simulation studies.

REAL-TIME DATABASE MODEL

An array of approximately 1000 data items stored in memory represents the real-time database model. CPU is the main system resource. The transactions in the model are classified as: User and Update transactions.

User transactions: The user transactions refer to data items in the main memory and perform some arithmetic and logical operations over these data items. A user transaction may perform some I/O operation in between. A random delay is introduced during execution of user transaction to simulate I/O operation. However, for the current research, it is assumed that, all the user transactions are CPU bound without waiting for I/O operation. A deadline is associated with these transactions. The transaction adds value to the system only when executed within its deadline. All the transactions are assumed to have soft deadlines. With soft real-time transactions, it is not necessary that all the transactions meet their deadlines all the time. The goal is to maximize the number of transactions that meet their deadlines. A transaction is allowed to continue, even if, it misses its deadline. For example, a web site tries its best to service customers as per the Service Level Agreement (SLA). Most of the times, most of the users experience acceptable response times. But, a transaction is not aborted, if the service time exceeds that acceptable limit. Solutions for meeting soft deadlines are dealt in the literature[9-12].

Update transactions: An update transaction updates a data item regularly as per its update interval. It is assumed that the update transactions maintain the data item reflecting the real-world state on a regular basis. For example, the data coming from sensors in an embedded system is captured and the data items are updated periodically as per the sensor state. A validity interval is associated with every data item. The update transactions are generated for various data items so that they reflect the real-world state within their validity intervals. An update transaction adds value to the system, only if, it is executed within the validity interval. Else, it is aborted. A user transaction, while referring to a data item must get a fresh copy (recently updated and reflecting the current world state) of data item. A transaction referring to stale data is blocked and waits for the data item to update before continuing further.

Data access issues: The data items are shared between user and update transactions. Therefore, the access to these data items is made through critical section. The data items are classified into two categories: Cold and Hot. Cold data items are the data items, which are accessed less frequently as compared to update frequency. The hot data items are the data items, which are frequently accessed as compared to update frequency. Hot data items are immediately updated. The update policy of cold data items is changed from immediate to on-demand, when the load of user transactions is increased. A term, called Access Update Ratio is introduced to categorize hot & cold data items. Access Update Ratio (AUR) of a data item is defined as the ratio of Number of References/ Number of Updations made to the data item in one sec. In our study, data items with AUR < 1 are termed as cold data items.

Architecture: The QoS management architecture proposed[7] is modified and is shown in Fig. 1. The user and update transactions are executed concurrently in a manner that the miss ratio (ratio of number of transactions missing their deadlines to the total number of transactions) is kept below the acceptable/predefined limits. Also, whenever, the miss ratio exceeds the acceptable limits (overshoot), it is brought back to the acceptable limits with the help of feedback controller within an acceptable time (settling time).

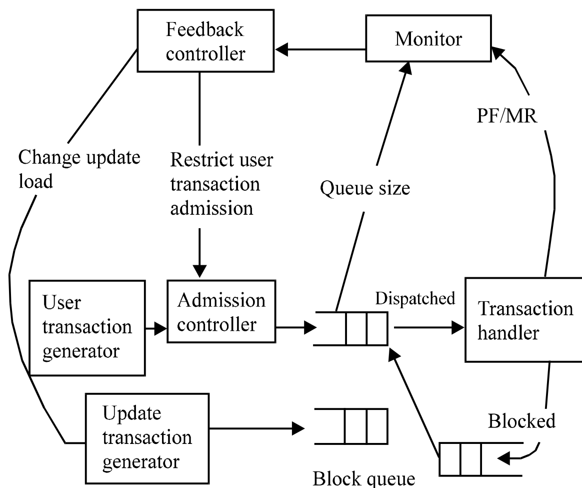

Fig. 1: Architecture for QoS management

An independent thread monitors the current miss ratio, queue size of transactions ready for execution and perceived freshness at predefined sampling interval. The miss ratio and current CPU utilization along with weighted queue size compute the required CPU utilization adjustment. Considering the current performance error such as the miss ratio overshoot or CPU underutilization, the QoD manager adapts the update policy or period (according to the selected freshness management scheme) to reduce the update workload, if necessary. The admission controller enforces the remaining utilization adjustment, if the reduction in the update workload does not result in keeping miss ratio within acceptable limits.

Feedback controller: Feedback Controller activates when

i. The user transaction workload is increased
ii. Transactions are unable to meet their deadlines leading to increase in miss ratio beyond the acceptable limits or/and
iii. Transactions getting stale data due to missed update operation. The module calculates the required CPU utilization adjustment using a proportional and integral controller and performs necessary action to bring QoS metrics under acceptable limits. The action is in the form of activating the adaptive update policy so as to reduce the update workload. This action permits the user transactions to meet their deadlines.

Transaction handler: An important component of our architecture, called transaction handler consists of a predefined number of worker threads. Each worker thread through a critical section picks up the next transaction from the user transaction queue and starts the execution. Every transaction during its execution refers to data items. The reference to data items by the user transactions is made in a manner that the defined access frequency of a data item is maintained. The shared data items are updated and referred through a critical section. The freshness of a data item is checked before accessing a data item using the corresponding update interval and validity period. A user transaction is placed in blocked queue, if a stale data item is accessed. It waits for the updation to perform before continuing further.

Modified architecture: The model proposed[7] is modified with an adaptive update policy and includes an additional parameter: weighted-queue-size. The overload condition is detected by monitoring the CPU utilization as well as the queue size. If the weighted queue size exceeds its threshold, then feedback controller is activated to reduce the update workload. The advantage of our approach is that, we can predict the load increase and take corrective action before the CPU utilization, miss ratio, perceived freshness and other QoS metrics exceed their threshold limit. It may be late, before the feedback controller starts and brings back QoS metrics back within their desired limits. Also, the queue size can be easily maintained with every insertion and deletion of transactions from the queue. An increase in queue size is an indicator of likely increase in CPU utilization, whereas, a decrease in queue size is an indicator of likely decrease in CPU utilization. The adaptive update policy reduces the update transaction workload in such a manner that the number of user transactions meeting their deadlines is increased.

Miss ratio adjustment: When the miss ratio exceeds the predefined acceptable limits, it is required to bring back the miss ratio within limits by reducing the update workload. Increasing the update period of few cold data items reduces the update workload. Assume that, the required reduction in CPU utilization is Y and deltaX is reduction in CPU utilization because of eliminating one update transaction. So, the required number of update eliminations, say X is calculated by the formula given below:

$$X = Y / \text{deltaX} \tag{1}$$

The required reduction in the number of update transactions is mapped to increase in update period of few cold data item. If the feedback controller suggests reducing CPU utilization by Y% to maintain miss ratio and perceived freshness within limits, it is found that, by how much factor, the update period of few cold data items is to be increased. Given below is the equation describing the increment factor of update period. ΣupdateFrequency denotes the sum of update frequency of all the cold data items.

$$\text{incrementFactor}=(1.00-((\Sigma\text{updateFrequency}-X)/\Sigma\text{updateFrequency}) \tag{2}$$

All the cold data items' update period is then incremented by incrementFactor. When the QoS metrics is not found within acceptable limits even after changing the update policy of all the cold data items, admission control is then applied.

QoD upgrade: Freshness adjustment: When the perceived freshness limit[7] (ratio of temporally valid data references as compared to total number of data references) is violated, the update period of few cold data items is decreased so as to maintain the data items reflecting their status closer to real-world status. Given below is the equation describing the calculation of decrementFactor for update period of few cold data items based on the required increase in CPU utilization and required increase in number of update transactions. The data items in the order of relatively high AUR value are selected for this purpose. However, the policy upgrade starts only when the CPU utilization reaches below a specified threshold value of freshness constraints are violated. ΣupdateFrequency denotes the sum of update frequency of all the cold data items.

$$decrementFactor=((\Sigma updateFrequency-X)/\Sigma updateFrequency) \quad (3)$$

Modified update policy: We present an adaptive update policy, which modifies the regular update pattern of data items to reflect the real-world status in such a manner that the system is able to maintain QoS within limits even under unpredictable and busty traffic conditions. Under abrupt change in workload, the cold data item's update interval is changed depending upon the required workload adjustment. We estimate the workload based on weighted queue size and existing CPU utilization and then mapping it to appropriate changes in update workload so as to accommodate increased user workload and still maintaining QoS within limits. The detailed algorithm is explained below:

```
for I = 1 to numberOfDataItems
{
{filter frequently accessed hot data items. Hot data items update interval is kept unchanged even under overloaded condition)
let Y = expected reduction in workload
{Y is estimated based on existing workload and the weighted queue size}
let Δx = reduction in workload because of eliminating one update transaction
Total number of update eliminations required by changing the update policy
    X = Y/Δx
let lfaDataItems = less frequently accessed data items with updatePeriod < settlingTime
let sumUpdateFrequency represents the sum of update frequency of all the lfaDataItems
for I = 1 TO lfaDataItems
if (updatePeriod_i <= settlingTime)
{
if (X > 0)
{
 incrementFactor = (1.00 - ((sumUpdateFrequency - X)

                                / sumUpdateFrequency));
updatePeriod_i  /= incrementFactor;
} else
{
decrementFactor = ((sumUpdateFrequency - X)

                    / sumUpdateFrequency);
updatePeriod_i  /= decrementFactor;
}
}
```

Performance evaluation: The simulator is designed to implement the proposed real-time database memory model and demonstrate the effectiveness of the proposed architecture. It simulates concurrent execution of two sets of transactions. User & Update transactions are generated through two separate streams. User transaction arrival rate is random and follows an exponential distribution. Update transactions are generated as per the predefined update intervals. The access to data items by user transactions is random, but controlled in a manner that the predefined Access Update Ratio is maintained.

EXPERIMENTAL RESULTS AND DISCUSSIONS

Simulator varies the application load by changing the inter-arrival rate of user transactions. The load of update transaction was fixed at approximately 50% of normal load, depending upon update interval of data items. The simulation run was carried out by varying the load from 10-200% at an interval of 10 sec each. Overall, simulation run was 200 sec per experimentation. The experimentation was repeated with identical load conditions and keeping all other tunable parameters fixed.

As shown in the Fig. 2, it is observed that, in the absence of adaptive update policy through feedback control, miss ratio exceeds the acceptable/predefined limits. However, by monitoring the queue size in addition to CPU utilization, miss ratio, perceived freshness, load increase can be predicted and accordingly, feedback controller is activated. The overall objective is to improve the transient performance, reduce the settling time and minimize number of transactions missing their deadlines during the transient period.

In the second experiment, abrupt load spikes for short duration were introduced. Load was increased to 100%-150% for a short duration of 1-5 sec. It is evident from the Fig. 3 given below, that the total number of

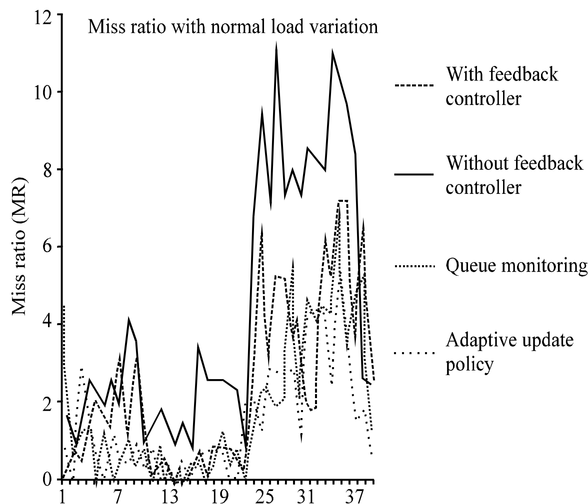

Fig. 2: Application load vs miss ratio

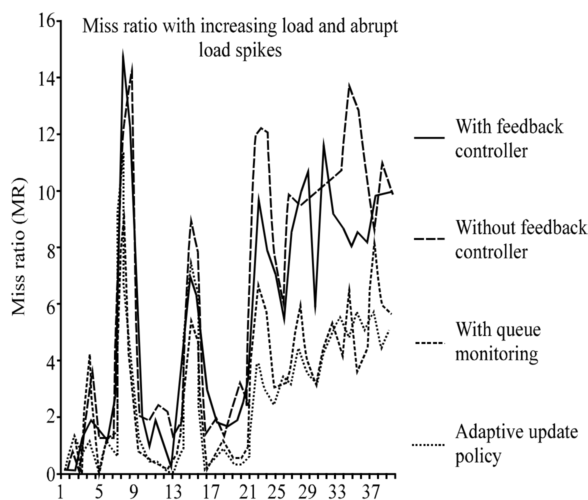

Fig. 3: Application load vs miss ratio (with bursty load variation)

transactions missing their deadlines was notably reduced as compared to the total number of missed transactions without feedback controller. The results clearly show that for abrupt load increase, the miss ratio reduction with adaptive update policy along with queue monitoring is considerably higher than that observed without using it, which shows the effectiveness of the proposed architecture.

CONCLUSIONS

Kang *et al*.[7] have studied the trade-off issues between timeliness and data freshness. They have proposed a model to provide performance guarantee in terms of either miss ratio or data freshness. We have extended that model and proposed a modified update policy so as to maintain miss ratio and data freshness even under busty traffic conditions. The QoS management architecture proposes a modified update policy, which adjusts the update workload as per changes in use transactions workload and tries to bring back QoS metrics within limits so as to reduce overshoot and latency time. With the proposed update policy, which maps the increased workload to required change in update period, QoS metrics were maintained within limits even under busty traffic. Further, by our improved adaptive update policy, significant reduction in number of deadline misses and freshness violations was observed. The results obtained through simulation show that the miss ratio can be better controlled with proposed approach during busty load variation. The proposed architecture is being further extended to associate weight with every transaction based on its CPU or I/O bound nature and compute weighted-queue-size so as to accurately predict load increase and accordingly change the update pattern, thereby further improving the miss ratio.

REFERENCES

1. Bestavros, A., K-J Lin and S. Son (eds.), 1997. Real-time database systems: Issues and applications. Kluwer Academic, Boston.
2. Stankovic, J., S. Son and J. Hansson, 1999. Misconceptions about real-time databases. IEEE Comp., 32: 29-36.
3. ActivMedia Research. Real Numbers behind ’Net Profits. http://www.activmediaresearch.com/.
4. Chaudhuri, S. and G. Weikum, 2002. Rethinking database system architecture: Towards a self-tuning, RISCstyle database system. In: Very Large Databases.
5. Schiefer, B. and G. Valentin, 1999. DB2 universal database performance tuning. IEEE Data Eng. Bull., 22:12-19.
6. Weikum, G., A. Moenkeberg, C. Hasse and P. Zabback, 2002. Self-tuning database technology and information services: from wishful thinking to viable engineering. In: VLDB Conference.
7. Kang, K.D., S.H. Son, J.A. Stankovic and T.F. Abdelzaher, 2002. A QoS-sensitive approach for timeliness and freshness guarantees in real-time databases. In the 14th Euromicro Conference on Real-Time Systems, (With modifications to appear in July 2004, IEEE Transactions on Knowledge and Data Engineering)
8. Tanwani, S., A.K. Ramani. 2004. Proactive management of deadline miss ratio and data freshness in real-time databases. CIT 2004 7th International Conference on Information Technology, IDRBT, Hyderabad, India Dec. 20-23.
9. Huang, J., J.A. Stankovic, K. Ramamritham and D. Towsley, 1991. On using priority inheritance in real-time databases. Proc. Real-Time Systems Symposium.
10. Son, S.H., Y. Lin and R.P. Cook, 1991. Concurrency control in real-time database systems. In: Foundations of Real-Time Computing: Scheduling and Resource Management. Edited by Andre van Tilborg and Gary Koob. Kluwer Academic Publishers, pp: 185-202.
11. Stankovic, J.A., K. Ramamritham and D. Towsley, 1991. Scheduling in real-time transaction systems. In: Foundations of Real-Time Computing: Scheduling and Resource Management. Edited by Andre van Tilborg and Gary Koob. Kluwer Academic Publishers, pp: 157-184.
12. Stankovic, J., C. Lu, S. Son and G. Tao, 1999. The case for feedback control real-time scheduling. EuroMicro Conference on Real-Time Systems.