Maintaining Data Freshness in Distributed Real-Time Databases [*]

Yuan Wei Sang H. Son John A. Stankovic
Department of Computer Science
University of Virginia
Charlottesville, Virginia, 22904-4740, USA
E-mail: {yw3f, son, stankovic}@cs.virginia.edu

Abstract

Many real-time systems need to maintain fresh views which are derived from shared data that are distributed among multiple sites. When a base data item changes, all derived views that are based on it need to be re-computed. There are two major derived data re-computation strategies - immediate update and on-demand update. However, they both have their advantages and limitations. In this paper, we study the performance of derived data update using immediate and on-demand strategies in distributed real-time databases and identify several criteria for choosing proper update policies. Based on these criteria, we propose a derived data update algorithm. In our algorithm, the update policy of a particular derived data item is determined dynamically by its access frequency, current transaction miss ratio and the system utilization. A thorough simulation study shows that our algorithm outperforms immediate and on-demand update in most cases.

1 Introduction

Real-time database systems are often employed to monitor and interact with dynamic environments. They process incoming application transactions and ensure that they meet their time constraints. In addition to usual application transactions, real-time database systems also need to handle temporal data updates and derived data re-computations. In stock trading, for example, the system must process incoming user transactions and at the same time, keep temporal data and derived data fresh. If temporal data are not refreshed, they will gradually become stale and useless. *Derived data* are maintained in real-time databases to summarize the base data items which reflect the real world states. Derived data may be out-of-date if they are not recomputed when their base data items change.

In real-time database systems, the workload of temporal data update can be very high (e.g., over 500 updates/sec [6]). Because multiple derived data items may be derived from one base data item and they all need to be re-computed if that base data item changes, the workload for derived data re-computation can be even higher. Given the fact that the temporal data update, derived data re-computation and application transactions share the system resources at run time, trade-offs need to be carefully studied. If insufficient system resources are given to temporal data updates and derived data re-computations, the application transactions will not get the fresh data they need; if too many system resources are given, there will be insufficient system resources left for application transactions.

In a previous study [3], *on-demand* update is shown to have the best overall performance in terms of transaction miss ratios and returned value in centralized real-time database systems. However, this conclusion dose not hold in distributed environments because the on-demand updates in distributed systems may take much longer than those in centralized systems. For example, consider that an application transaction in a distributed real-time database needs to access a stale derived data item. The transaction first gets blocked, waiting for the derived data to be updated by the re-computation transaction. However, in a distributed environment, some base data items may come from remote sites. The on-demand derived data re-computation now takes much more time because the system needs to exchange messages to request the base data items from remote sites. If the slack time of the incoming application transaction is short, the original transaction is likely to miss its deadline. From this example, it is obvious that the system designers need to be very careful when using an on-demand update strategy in distributed real-time database systems.

There is clearly a need for understanding the effects of stale data blocking and its impact on transaction timeliness in a distributed environment. Most previous research only focused on maintaining the freshness of temporal base data, while issues about maintaining the freshness of derived views in a distributed environment have never been studied. As many systems need

[*]This work was supported, in part, by NSF grants IIS-0208758, CCR-0329609 and CCR-0098269.

to operate on both temporal base data and derived views, previous research work does not seem to be sufficient. In this paper, we study the effects of derived data update policies on meeting transaction timing requirements in distributed real-time databases. To the best of our knowledge, this work is the first paper that systematically investigates this issue in distributed real-time database systems. In the paper, we summarize the observations from our simulation study and use them as the basis for the adaptation of the derived data update policy. Based on these observations, we propose a novel update policy adaptation algorithm that dynamically selects update policies for derived data items according to the system conditions. The simulation study shows that our algorithm utilizes the system resources more efficiently and provides better transaction miss ratios than immediate and on-demand update policies under different system conditions.

2 System Model

In this paper, we study a distributed real-time database system which consists of a group of main memory real-time databases connected by a Local Area Network (LAN). Main memory databases have been increasingly used for real-time data management because of the high performance of memory accesses and the decreasing main memory cost. In the system, a firm real-time database model is used. Tardy transactions (transactions that have missed their deadlines) are aborted upon their deadline misses.

In our system, transactions are divided into two types, *system update transactions* and *user transactions*. The system update transactions consist of temporal base data (sensor data) update transactions and derived data re-computation transactions. The user transactions are queries or updates from applications.

Transactions are represented as a sequence of *operations* on the data objects. The operations of temporal base data update transactions are always *write*. The derived data re-computation transactions read from a set of base data items, recompute the results and then update the corresponding derived data items. For user transactions, operations on temporal data are read only (temporal data are updated only by the system update transactions) while operations on non-temporal data objects are either *read* or write. Operations within one transaction are executed in *sequential* fashion. One operation can not be executed unless all previous operations are finished. Once all operations of one transaction are finished, the transaction enters the *validation* stage. At the validation stage, the system checks the freshness of the accessed data. If the accessed data are not fresh anymore, the transaction will be restarted. After the validation stage, the transaction enters commit stage during which the logging and commitment operations are performed.

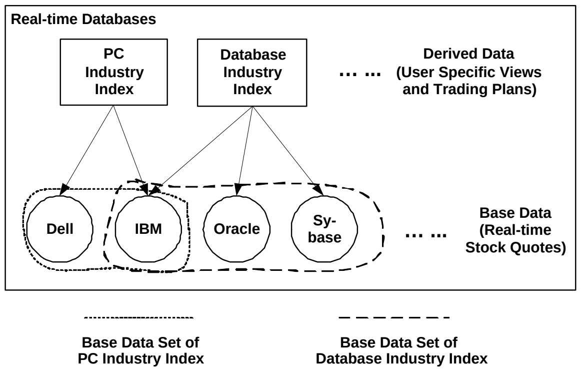

Real-time Databases
PC Industry Index
Database Industry Index
... ...
Derived Data (User Specific Views and Trading Plans)
Dell
IBM
Oracle
Sy-base
... ...
Base Data (Real-time Stock Quotes)
Base Data Set of PC Industry Index
Base Data Set of Database Industry Index

Figure 1. Derived Data and Base Data Set

If a transaction needs to access data items at another site, it sends out the data service requests to set up cohorts at remote sites. The cohorts read or write data items on behalf of the transaction. At the validation stage, the transaction sends out the commit preparation message to all the cohorts. As in the 2-phase commit protocol, the transaction commits only if all the cohorts agree to commit.

3 Data Model

3.1 Data Classification

In our data model, a data item is called *base data* if its value does not depend on any other data items in the database; a data item is called *derived data* if it is computed from a set of data items in the database and its value depends on the values of those data items. A derived data item can be derived from another derived data as long as there is no cycle in the derivation relation. The set of base data items on which a derived data item depends is called the *Base Data Set*. Examples of the base data set are given in Figure 1. In the figure, two derived data items, PC industry index and database industry index, are derived from four base data items. The base data sets of different derived data items can overlap. For example, as shown in Figure 1, the stock quote of IBM might be used in computing both the index for the PC industry and the index for the database industry.

A data item is *temporal* if it models an external environment variable and its value changes with that variable. The temporal data items are divided into two types: *temporal base data* and *temporal derived data*. A data item is called *temporal base data* if it comes directly from the external environment and does not depend on any other data item in the database. The values of temporal base data items reflect the states of a dynamically changing environment, such as the sensor readings of physical variables or the quotes of stocks. Usually, the temporal base data items are updated periodically to reflect the changes of the environment variables. A data item is called *temporal derived data* if it is derived from a set of data items in the database and there is at least one temporal data item in its base data set. The *non-temporal* data items are those data in the database that do not change with the external environment.

3.2 Temporal Consistency

The temporal consistency measures how well the temporal data items reflect the actual state of the environment. We use the absolute temporal consistency model that is defined in [13].

A temporal base data item is defined as

$$TBD : (value, avi, timestamp) \qquad (1)$$

where TBD_{value} denotes the current state of the temporal base data. $TBD_{timestamp}$ is the time when this temporal base data is sampled from the physical world. TBD_{avi} is the time interval following $TBD_{timestamp}$ when the temporal base data item is considered to be *absolutely valid* (temporally consistent). A temporal base data item is valid iff

$$(current\_time - TBD_{timestamp}) \leq TBD_{avi} \qquad (2)$$

Assume that a temporal derived data TDD is derived from a *base data set* R, which has a *relative validity interval* denoted by R_{rvi}. TDD is relatively consistent iff
$\forall TBD \in R$

- Absolute Consistency: $(current\_time - TBD_{timestamp}) \leq TBD_{avi}$
- Relative Consistency: $\forall TBD^{'} \in R, \mid TBD_{timestamp} - TBD^{'}_{timestamp}| \leq R_{rvi}$, where TBD and $TBD^{'}$ are two temporal base data items in the base data set R.

In this paper, we assume that the *rvi* of a derived data item is larger than the *avi* of any of its base temporal data items. Under this assumption, the *rvi* of a derived data item is maintained as long as all temporal base data items from which it derives are absolutely consistent based on their *avi*. This assumption is reasonable in the sense that if the derived data are accessed by incoming transactions, the relative consistency requirements of the derived data items can be considered as requirements of incoming transactions. As the *avi* of a temporal base data item should meet the temporal requirements of all incoming transactions, it should also meet the requirements of the derived data *rvi*.

4 Update Policies and Scheduling

There are two commonly-used update policies for temporal base data update: *immediate* update and *on-demand* update. With the immediate update policy, a temporal base data item is updated once a new value is available. With the on-demand update policy, the base data items are not updated until they are accessed by incoming application transactions.

However, the derived data items and their base data items become inconsistent if the derived data items are not recomputed when their base data item change. In real-time databases, derived data could be derived from both temporal and non-temporal data items. In this paper, we mainly study the re-computation incurred by the updates of temporal base data items. The derived data items are updated immediately when their non-temporal base data items are updated. However, depending on the derived data update policy used, the updates of derived data could be deferred when their temporal base data item gets updated.

In this paper, the updates of the base data and the updates of the derived data are decoupled. They are not put in the same transaction because that would make the update transaction run longer and possibly block other transactions. Similar results have been observed in active database systems where decoupling the *event*s and the *action*s can greatly improve the response time of the transactions [5].

We assume that accessing fresh temporal data is important. Based on previous research results [3], the system schedules the temporal base data update transactions first, temporal derived data update transactions second, and incoming application transactions last. Within the same type, transactions are scheduled based on their deadlines using Earliest Deadline First (EDF) scheduling algorithm [11].

5 Update Policy Adaptation

5.1 Observations from Simulation Study

Choosing a proper update policy is not simple. Many factors affect the decision, including the transaction slack time, transaction access patterns, current system workload, on-demand update cost and others. In this section, we summarize the observations from our simulation study as the basis for the adaptation of the derived data update policy. Our simulation study indicates the following:

- The performance of the immediate update policy is fairly good and is almost not affected by other system parameters when the system workload is low. Thus it is the best choice when the system workload is low.
- When the incoming transaction workload is low, the immediate update policy gives a better performance; when incoming transaction workload is high, the on-demand update policy delivers a better performance. As the slack factor gets shorter, the performance gaps between two update policies at the low and the high workload ends become larger.
- When the transactions access patterns gets less skewed, the on-demand update policy can not save much CPU time, but will incur a serious transaction response time delay. Therefore, the system should refrain from using the on-demand update policy under that circumstance.
- The system should refrain from using the on-demand update policy when the network operation delay is long or when many derived data items are derived from base data items residing at remote sites.

5.2 Dynamic Adaptation of Derived Data Updates

To guide the update policy selection process, we define a term called *Access Update Ratio* (AUR), for a data object O_i as follows:

$$AUR[i] = \frac{AccessFrequency[i]}{UpdateFrequency[i]} \tag{3}$$

The notion of AUR has several interesting features. It can be a guideline for selecting a proper update policy for a certain data item. It can also be a cost/benefit indicator for the immediate update of a data object. A pictorial description is given in Figure 2, which is a snapshot of a real-time database. In the figure, the derived data objects in the database are ordered by non-increasing values of AUR. If a data item is on or above the horizontal line of AUR $= 1$, it is accessed at least as frequently as it is updated. For simplicity, we call the data with $AUR \geq 1$ *hot* data, and the data with $AUR < 1$ *cold* data, respectively. It is clear that the hot data should be updated in an aggressive manner. If a hot data item is out-of-date when accessed, potentially multiple transactions may miss their deadlines waiting for the update. For cold data, the on-demand update policy can be used to reduce the CPU utilization when the system is overloaded.

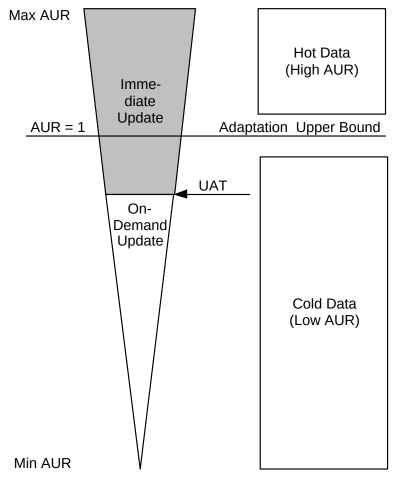


Figure 2. Database Snapshot Sorted by Access Update Ratio

In order to minimize the transaction response delay resulting from the on-demand update policy, the system needs to use the immediate update policy on as many data items as possible, as long as the desired transaction miss ratio is maintained. As shown in Figure 2, we set a value *update adaptation threshold* (UAT) for selecting a right update policy for a data item. A derived data item will be immediately updated if it has a AUR not less than UAT; otherwise, it will be updated using on-demand update. By controlling the value of UAT, we can control the number of the derived data items that are updated by the on-demand update policy and thus the CPU time that is saved by dropping the immediate updates on them. The UAT is set to a number between zero and one. Note that the upper bound is set at one because moving the UAT beyond one would not necessarily save CPU time, since hot data items are, by definition, accessed more often than they are updated.

When the system is seriously overloaded, there may not be enough CPU time for incoming transactions even when the UAT is set to one. Under those conditions, we need to use admission control to limit the incoming transaction workload.

5.3 Implementation

In order to implement our derived data update adaptation algorithm, the system performance needs to be closely monitored. To implement the derived data adaptation algorithm, we need to add a moni-

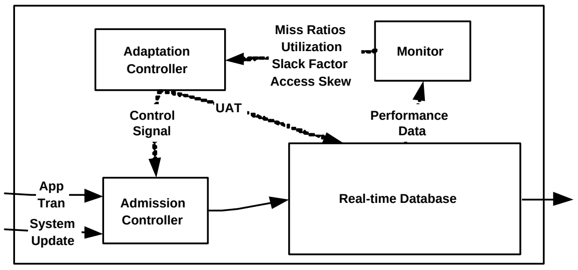


Figure 3. System Architecture

tor, an adaptation controller and an admission controller. The system architecture is shown in Figure 3. In the system, the monitor periodically collects the system performance data and sends them to the adaptation controller. The adaption controller calculates the appropriate UAT value and sends it to the real-time database. The UAT value is used at the next sampling period to control the update policy adaptation process. If the UAT value has already reached 1, adapting derived data update policy alone is not enough to meet the desired transaction miss ratio. In that case, the adaptation controller sends a control signal to the admission controller to reduce the number of the admitted transactions.

The monitor module checks the system performance periodically. The CPU utilization, miss ratio and slack factor can be sampled directly from the system. The AURs of derived data item are obtained by recording the numbers of read and update operations and dividing the read operation number by the write operation number. In our algorithm, the transaction access skew is measured by the standard deviation of AURs. The larger the standard deviation of AURs, the more skewed are the transaction accesses.

Based on the observations from our simulation study, we identify the CPU utilization, transaction missed ratio, transaction access skew and slack factor as the most important system parameters that affect the performance of the derived data update algorithms. Adaptation controller takes these parameters into consideration and makes the adaptation decisions. Once the system performance data are sent to the adaptation controller, the adaptation controller inspects whether the miss ratio specifications are met. If it is met, the controller increases the ratio of transaction admission or applies the immediate update on more data. If it is not met, the controller either decreases the number of data items that are updated by the immediate update, or decreases the number of admitted transactions. When the system is overloaded, the adaptation controller tests whether the on-demand update policy is appropriate. The on-demand update is appropriate only when transaction access pattern is fairly skewed and the system utilization is high. If the transaction access pattern is not skewed, from our experimental data, the on-demand update policy is not appropriate. If the utilization is not high while the transaction miss ratio is high, the CPU is not the bottleneck of the system. Thus on-demand update policy is not appropriate.

Parameter	Value
Node Number	8
End to End Transmission Delay	(50 - 1200) micro seconds
Temporal Data #	1000 /Node
Temporal Data Size	Uniform (1 - 128) bytes
Temporal Data AVI	Uniform (1 - 2) seconds
Non-temporal Data #	10,000 /Node
Non-temporal Data Size	Uniform (1 - 1024) bytes
Derived Data #	500/Node
Derived Data Size	2 - 6 base data items
Base Data Remote Data Ratio	0.2
Base Data Temporal Data Ratio	0.75

Table 1. System Parameter Settings

Parameter	Value
Operation Time	0.2 - 2 ms
Temporal Data OP #	1 - 4 /tran
Non-temporal Data OP #	1 - 2/tran
Derived Data OP #	1 - 2/tran
Transaction Slack Factor	5
Temporal Data Access Skew	20%
Non-temporal Data Access Skew	10%
Derived Data Access Skew	10%
Remote Data Ratio	20%
On-demand Update Cost	0.1 ms

Table 2. Transaction Parameter Settings

6 Performance Evaluation

In this section, we describe our simulation settings and present the results of the performance evaluation. There are two objectives for our simulation study. The first objective is to study the performance of the immediate and on-demand derived data update policies under different system settings and find the proper conditions for using them. The second objective is to test whether our algorithm can deliver better system performance compared to the immediate or on-demand update policies under different system conditions.

6.1 Simulation Setting

For the simulations, we have chosen values that are, in general, representative of online stock trading systems [7]. We have also chosen other system parameter values that are typical of today's technology capabilities, e.g., network delays. The general system parameter settings are given in Table 1. The size of the base data set of a derived data item ranges from 2 to 6. For derived data items, 20 percent of base data items come from remote sites and 75 percent of them are temporal. In the simulation study, we only study one-level derivation relation, i.e., no derived data item is derived from the other derived data items. The network delays are modelled by calculating the end-to-end transmission delay for each packet. Depending on the packet size (64 - 1500 bytes for Ethernet), the end-to-end delay ranges from 50 microseconds to 1.2 milliseconds. If the data size exceeds one packet size, the data is put into separate packets and the transmission delay is the sum of delays for those packets. In the simulation, the overhead of running adaptation controller is not modelled. The reason is that the controller computation procedure is not invoked very often (in our case, once every 5 seconds) and its computation cost is negligible compared to the CPU requirements of incoming transactions.

The settings for the user transaction workload are given in Table 2. A user transaction consists of operations on all three types of data objects. The execution time for one operation is between 200 microseconds to 2000 microseconds. The slack factor for transactions is set to 5. To model the transaction access patterns, we introduce *Temporal Data Access Skew*, *Non-temporal Data Access Skew* and *Derived Data Access Skew*. A 20% access skew means that 80 percent of all transaction operations access 20 percent of that type of data objects. The *Remote Data Ratio* is the ratio of the number of remote data operations (operations that access data hosted by other sites) to that of all data operations. The remote data ratio is set to 20%, which means 20 percent of all operations are remote data operations. The on-demand update cost (the time to get a fresh value for a temporal base data item) is set as 100 microseconds. At each node, the user transactions arrive according to a Poisson distribution and the average arrival rate is shown. All simulation results are based on at least ten runs and the confidence intervals are less than 10% of the mean values.

6.2 Performance with Immediate and On-Demand Update

To better understand the performance data of both update algorithms, we implement an *oracle* algorithm, which knows everything about the transactions and only updates those derived data items that will be accessed by incoming transactions. The performance of the oracle algorithm is the best performance the system can achieve. The system performance results are shown in Figure 4. In the figure, it is clear that the performance of the immediate update policy is better than that of on-demand update policy with low system workload. For example, when transaction arrival rate is 160 tran/s, the miss ratio using immediate update is around 2% while the miss ratio of on-demand update is around 10%. When the system workload is high, the performance of on-demand update is better. For example, when the transaction arrival rate is 720 tran/s, the miss ratio using on-demand update is at 12% while the miss ratio using immediate update is as high as 38%. We can see clearly that there are miss ratio gaps between the oracle algorithm and these two algorithms, which means that there is room for improvement.

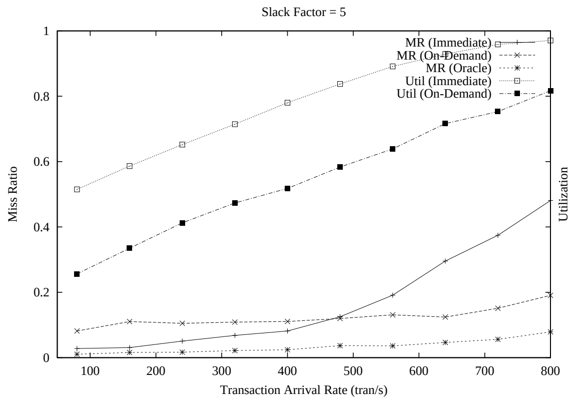
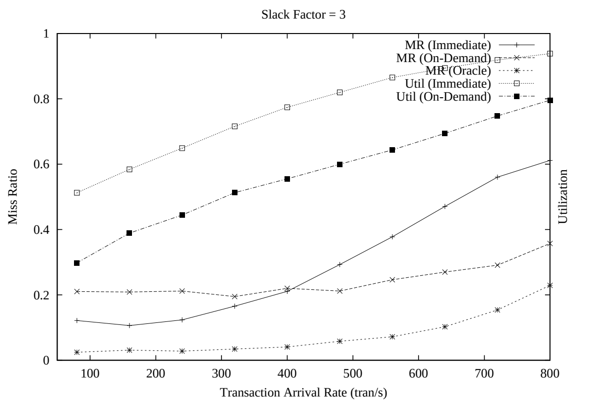


Figure 4. Normal System Performance

In the remaining parts of this section, we compare the performance of the immediate derived data update policy with on-demand derived data policy update under different system parameter settings, including different transaction slack factors, different derived data access skews, different on-demand update costs and different remote base data ratios. The system performance results under different parameters are compared against the results shown in Figure 4.

6.2.1 Transaction Slack Factor

The transaction slack time has major impact on the system performance. When the transaction slack time is long, the system can be more aggressive in deferring the derived data updates and saving system resources by using on-demand update. However, if the transaction slack time is short, lots of transactions will miss their deadlines while waiting for the on-demand updates. The simulation results of different slack factors are given in Figure 5. From the figure, we can see that when the incoming transaction workload is low, the immediate update policy gives better performance; when the incoming transaction rate is high, the on-demand update policy performs better. The performance differences between the two update policies at the low and high workloads become more evident as the slack



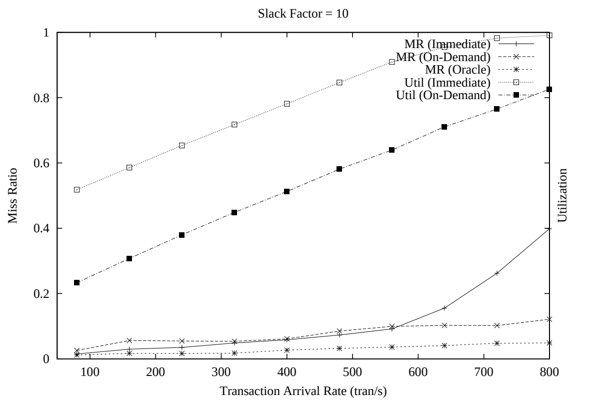
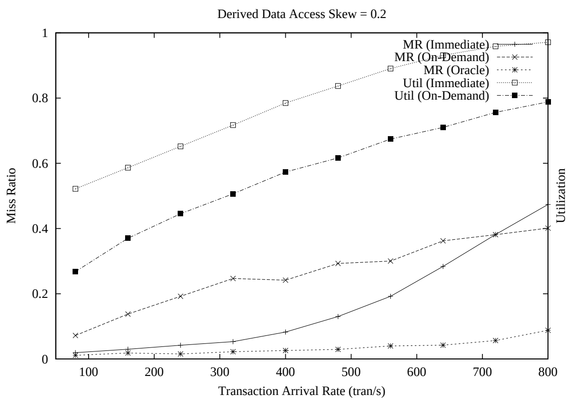


Figure 5. Performance with Different Slack Factors

factor gets shorter.

To determine the conditions for using different update policies, we define the term *switching point* as the incoming transaction arrival rate where miss ratios of the two update policies cross each other. As shown in Figure 4, when the slack factor equals to 5, the switching point is around 470 tran/s. Comparing Figure 5 with Figure 4, we can see that as the slack factor gets shorter, instead of moving right, the switching point moves left, which is contrary to our initial expectation. Our initial expectation was that when the slack factor gets shorter, the miss ratio using on-demand update will increase and the switching point will move right. The simulation results show something different. The miss ratio using on-demand update policy does move up but the miss ratio of immediate update moves up even more. The reason is that a transaction is interrupted more often by temporal and derived data re-computations in the system using immediate update, while the transactions do not have enough time for blocking when the slack factor is short. Therefore, a lot of transactions miss their deadlines. Comparing the performance of the two algorithms with that of the oracle algorithm, we notice that when the transaction slack factor decreases, the performance gaps between immediate, on-demand algorithms and the oracle algorithm become larger.

6.2.2 Transaction Access Skew



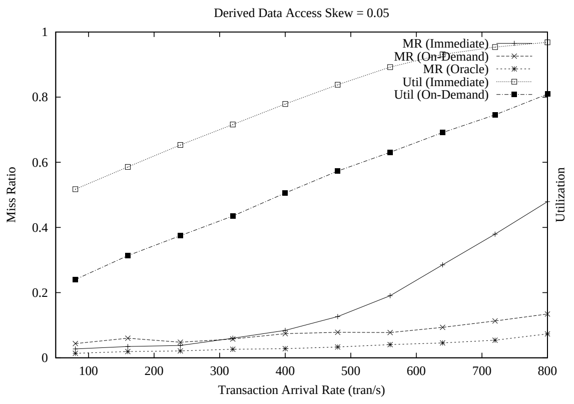


Figure 6. Performance with Different Derived Data Access Skews

The application access pattern is another major factor that affects the system performance. The system performance under different derived data access skew is given in Figure 6. Note that when access skew equals 0.2, it means 80% of accesses to the derived data go to 20% of the derived data items. Thus, the smaller the access skew value, the more skewed are the transaction data access patterns. Comparing Figure 6 with Figure 4, we find out that the performance of the immediate update policy almost does not change with the derived data access skew while the performance of the on-demand update changes dramatically. As shown in Figures 4 and 6, when the transaction derived data access changes from 0.1 to 0.2, the miss ratio of the on-demand update policy changes from 10% to 22 % at 400 tran/s. As a general rule, if the transaction access pattern is skewed, i.e., the majority of transactions access a small number of hot data items, on-demand update has more performance advantages and the switching point moves to the left. As the derived data access pattern gets less skewed, the switching point moves to the right. If the derived data access skew gets larger than 20%, the on-demand update policy has no performance advantage at all (Note that this conclusion is made under the condition that the slack factor equals 5. If the slack factor gets larger, the demarcation point can be higher). Comparing the performance of the two algorithms with that of the oracle algorithm, we notice that the performance gap between the immediate update algorithm and the oracle algorithm does not change while the performance gap between the on-demand update algorithm and the oracle algorithm changes dramatically. As the transaction access pattern gets more skewed, the performance gap between the on-demand update and the oracle algorithm becomes substantially smaller. As shown in the figure, when the derived data access skew is 0.2, the miss ratio gap between the on-demand update and the oracle algorithm is between 4% to 15% while the miss ratio gap is only between 1% to 3% when derived data access skew is 0.05. The reason is that when the transaction accesses get more skewed, transactions access a smaller set of hot data items. As a result, it becomes more likely that when a data item is needed, it may have already been updated by the on-demand updates from previous transactions.

6.2.3 On-demand Update Cost and Remote Base Data Ratio

We also perform simulations with different on-demand update costs and remote base ratio. With higher on-demand update cost or higher remote base data ratio, the miss ratios of systems using on-demand update rise and the switching point moves to the right.

6.3 Performance with Update Policy Adaptation

In this subsection, we evaluate the performance of our algorithm under different system conditions. Four derived data update algorithms are compared with each other. They are immediate update, on-demand update, our update adaptation algorithm (*Adaptation*) and the oracle. We use only these three algorithms to compare against our algorithm because to the best of our knowledge, there is no other algorithm specifically designed for the derived data update management in distributed real-time databases. To make a fair comparison between our algorithm and the others, we do not use admission control when the system workload is high. As we described in the previous section, admission control should be used when the incoming transaction miss ratio is higher than the requirements and the derived data adaptation can not save enough CPU time. However, we should not use admission control when comparing the system performance because the baseline algorithms do not use admission control as a measure to achieve better transaction miss ratios. The system performance with the slack factor of 5 and the derived data access skew of 0.1 is shown in Figure 7. As we can see in the figure, our algorithm delivers better system performance than immediate and on-demand update policies on all transaction workloads. With a low system workload (transaction arrival rate less than 400/s), our algorithm performs just like the immediate update policy, which delivers very good performance at this transaction workload range. With a high system workload (transaction arrival rate greater than 400/s), our algorithm consistently outperforms both immediate and on-demand update policies. The miss ratio of our algorithm is 3% to 5% lower than the on-demand update policy. The reason for this performance gain is that our algorithm updates hot data using immediate update policy to minimize the transaction data access delay.

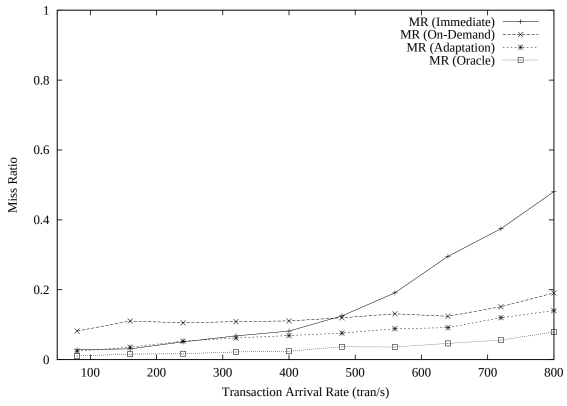
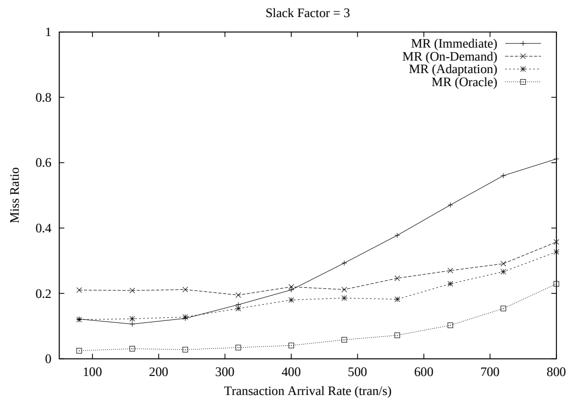


Figure 7. Performance with Update Adaptation

6.3.1 Impact of Slack Factor

As we have shown in the previous section, the transaction slack factor has a significant impact on the performance of the derived data update algorithms. To better understand the performance of our algorithm, we perform simulations on different transaction slack factors. The simulation results are shown in Figure 8. Comparing Figure 8 with Figure 7, we observe that as the slack factor gets shorter, the miss ratios of our algorithm rise just like other algorithms. But our algorithm still delivers the best transaction miss ratios compared to the immediate and on-demand update policies. However, the gap between our algorithm and the oracle algorithm gets larger as the slack factor get smaller. As we see from the figure, the miss ratio difference between our algorithm and the oracle algorithm changes from around 5% to around 10% when the transaction slack factor changes from 5 to 3. As the transaction slack factor gets larger, the miss ratios of all algorithms drift lower and the gaps between our algorithm, the on-demand update algorithm and the oracle algorithm become smaller.



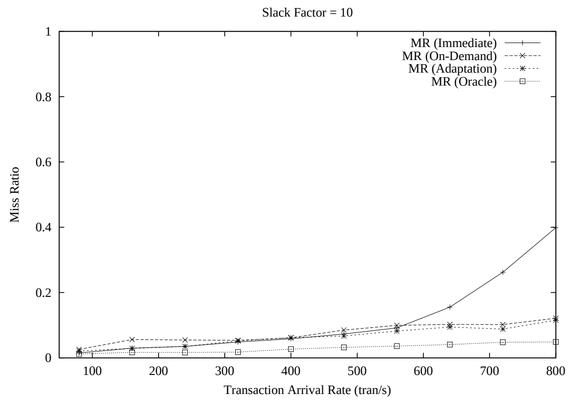
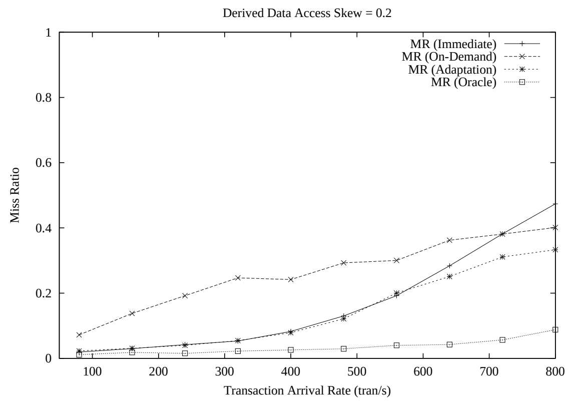


Figure 8. Impact of Slack Factor

6.3.2 Impact of Access Skew

Another parameter that significantly affects the performance of the derived data update algorithms is the derived data access skew. As shown in Figure 9, our algorithm delivers the best performance in almost all situations. The reason is that our algorithm estimates the access pattern of incoming transactions by calculating the standard deviation of the AUR of derived data items and makes decisions whether a derived data item should be updated using immediate update or on-demand update. When the transaction access pattern is skewed, our algorithm keeps immediate updates only on hot derived data items and drops updates on cold derived data items. By doing that, our algorithm saves enough CPU time for user transaction processing. We can also see that the gap between our algorithm and the oracle algorithm gets smaller when the derived data access skew gets smaller. The reason is that when transaction access skew is smaller, our algorithm is more accurate in finding hot derived data items.



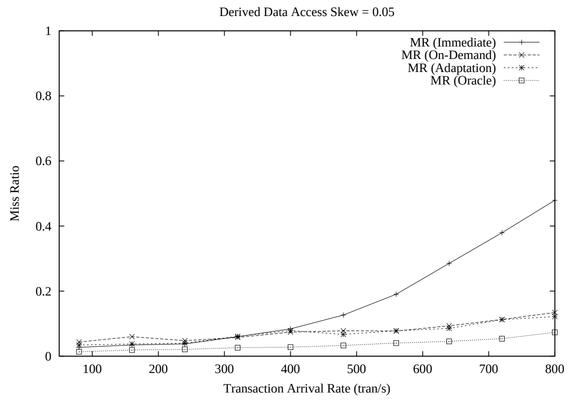


Figure 9. Impact of Slack Factor

Due to the space limitation, we can not show the simulation results under other system conditions, such as with different remote base data ratios and different on-demand update costs. They have a smaller impact on the performance of the derived data update algorithms compared to the slack factor and transaction access skew. In systems under these different conditions, we observe similar patterns in which our algorithm outperforms both the immediate and the on-demand update algorithms in terms of transaction miss ratios.

7 Related Work

Since the major publications by Abbott [1] and Stankovic [14] in 1988, real-time databases research has received a lot of attention. There have been several research papers addressing temporal data update policies in recent years. In [3], the load balancing issues between updates and transactions in a real-time database are studied. In the paper, it is shown that the *on-demand* strategy, with which updates are only applied when the data are needed by transactions, gives the best overall performance. In this paper, we show that this conclusion is not valid in the distributed environments and we propose an adaptive derived data update algorithm to maintain the derived data freshness in distributed real-time databases. In [4][2], Adelberg et. al. discussed the rule system implemented as part of the STanford Real-time Information Processor(STRIP). Kao et. al. defined the concept of temporal consistency from the perspective of transactions and discussed the properties of updates, re-computation and data accesses associated with derived data [9]. Their research focused on the derived data management in centralized real-time database systems. How to enforce the temporal consistency of derived data in the distributed environment was not addressed.

The *Quality-of-Data*(QoD or Data Freshness) is an important issue in database research. In [12], the issues on replicated data freshness in distributed databases are studied. In the paper, the mutual consistency is relaxed and data freshness is used to measure the deviation between replica copies. Labrinidis et. al. [10] discuss the design issues on QoD of dynamically generated data on the web. In [8], Kang et. al. present a QoS management scheme to support guarantees on both transaction deadline miss ratios and specified data freshness requirements.

8 Conclusion and Future Work

In this paper, we study the issues of maintaining derived data freshness in distributed real-time databases. We carefully study the performance of two basic derived data update algorithms, immediate and on-demand update algorithms. From our study, we find unlike that in centralized systems, on-demand update is not the best derived data update policy in distributed environments. In the paper, we also identify the proper conditions for using different update policies. Based on those conditions, we propose a derived data update algorithm that dynamically adapts the derived data update policies to the current system conditions. A thorough simulation study shows that our algorithm outperforms the immediate and on-demand update policies in most cases and performs no worse than any of them in worse-case scenarios.

There are several directions to expand this work. One direction is to study more aggressive on-demand update algorithms, in which on-demand update is used on temporal base data items. Another direction is to study the impact of replication and replica update policies on the system performance. The third direction is to study the trade-offs between the data freshness and the system throughput. We expect to address these issues in future papers.

References

[1] R. Abbott and H. Garcia-Molina. Scheduling real-time transactions. *SIGMOD Record*, 17(1):71 – 81, 1988.

[2] B. Adelberg. *STRIP: A soft real-time main memory database for open systems.* PhD thesis, Computer Science Dept. Stanford University, 1997.

[3] B. Adelberg, H. Garcia-Molina, and B. Kao. Applying update streams in a soft real-time database system. In *ACM SIGMOD*, 1995.

[4] B. Adelberg, H. Garcia-Molina, and J. Widom. The STRIP rule system for efficiently maintaining derived data. In *Proc. The ACM SIGMOD International Conference on Management of Data*, volume 26,2 of *SIGMOD Record*, pages 147–158, New York, May 13–15 1997. ACM Press.

[5] M. Carey, R. Jauhari, and M. Livny. On transaction boundaries in active databases: A performance perspective. In *IEEE Transactions on Knowledge and Data Engineering*, 1991.

[6] M. Cochinwala and J. Bradley. A multidatabase system for tracking and retrieval of financial data. In *Proceedings of 20th International Conference on Very Large Data Bases*, 1994.

[7] K. Kang, S. Son, and J. Stankovic. Differentiated real-time data services for e-commerce applications. *Electronic Commerce Research, Special Issue on Business Process Integration and E-Commerce Infrastructure*, 2003.

[8] K. Kang, S. Son, J. Stankovic, and T. Abdelzaher. A qos-sensitive approach for miss ratio and freshness guarantees in real-time databases. In *Proc. 14th Euromicro Conference on Real-Time Systems*, Vienna, Austria, 2002.

[9] B. Kao, K. Lam, B. Adelberg, R. Cheng, and T. Lee. Updates and view maintenance in soft real-time database systems. In *Proc. 8th International Conference on Information Knowledgement (CIKM 99)*, pages 300–307, New York, Nov. 2–6 2000. ACM Press.

[10] A. Labrinidis and N. Roussopoulos. Update propagation strategies for improving the quality of data on the web. In *Proc. 27th International Conference on Very Large Data Bases (VLDB 01)*, pages 391–400, Roma, Italy, Sept. 2001. Morgan Kaufmann Publishers.

[11] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1), 1973.

[12] E. Pacitti and E. Simon. Update propagation strategies to improve freshness in lazy master replicated databases. *VLDB Journal: Very Large Data Bases*, 8(3 - 4):305 – 318, 2000.

[13] K. Ramamritham. Real-time databases. *International Journal of Distributed and Parallel Databases*, 1(2):199–226, Apr. 1993.

[14] J. Stankovic and W. Zhao. On real-time transactions. *ACM Sigmod Record*, 17(1):4 – 18, 1988.