Mutual Consistency in Real-Time Databases

Abhay Kumar Jha
IIT Bombay
abhaykj@cse.iitb.ac.in

Ming Xiong
Lucent Bell Laboratories
xiong@research.bell-labs.com

Krithi Ramamritham
IIT Bombay
krithi@cse.iitb.ac.in

Abstract

A real-time database is composed of real-time objects whose values remain valid only within their validity intervals. Each object in the database models a real world entity. The freshness of these objects is maintained by update transactions that sample the real world entities. The literature proposes various ways to derive a schedule of transactions that preserves the freshness (also known as absolute consistency*) of these objects. But these approaches do not take care of the mutual consistency of the objects, i.e., whether together they represent a logical state of the system. We investigate the problem of checking whether, given an update transaction schedule, a periodic query would be able to read mutually consistent values. We propose solutions for both single- and multiple-query cases in the presence of non-preemptable query executions. Specifically, we first investigate formulas that give the maximal value of mutual gaps among a set of data read at a certain point in time. (A mutual gap for two object values read from the database refers to the difference between the times at which the two objects were updated.) We then propose design approaches to (1) decide the period and relative deadline of a query so that it would guarantee mutual consistency; (2) decide if a given set of queries with relative deadlines and periods can guarantee mutual consistency. Finally, we suggest ways of reducing the complexity of our proposed approaches for both harmonic periods and general cases.*

1 Introduction

A real-time database is composed of real-time objects which are updated by periodic sensor transactions. An object in the database models a real world entity like the position of an aircraft, whose state may become invalid with the passage of time. Associated with the state is a temporal validity interval. It is the responsibility of the update transactions to maintain the freshness of the objects by refreshing them before their temporal validity interval expires. There are many approaches proposed in literature to determine a schedule for the update transactions that maintains the freshness of objects [22, 16, 9, 23, 24]. But these schedules just concentrate on maintaining the individual freshness of the objects without paying any regard to their relative freshness (also known as mutual consistency). This might result in the presentation of an incoherent view of the system.

The need for mutual consistency is motivated by the observation that many objects may be related to one another and that the system should present a logically consistent view of the objects to any query. The need for mutual consistency is apparent in the web domain, for example, on newspaper websites carrying breaking news stories that consist of text objects accompanied by embedded images and audio/video clips. Since such stories are updated frequently (as additional information becomes available), the source serving the request has to ensure that any cached version of the text is consistent with the embedded objects. Similarly, while delivering up-to-the-minute sports information, it has to be ensured that the cached objects are mutually consistent with each other. In real-time databases, the values of the objects sampled by different sensors may be related to one another. Thus, it is important to preserve the relative freshness of these objects in addition to their individual freshness. This motivates the work reported here.

We address the question of mutual consistency between objects in this paper. Instead of proposing a schedule that preserves both individual as well as mutual consistency we look at a related problem of determining whether a given schedule of queries and updates maintains the mutual consistency among the objects.

There are already various approaches proposed to maintain mutual consistency in the web domain, e.g., see [20]. Let us first review the mutual consistency semantics. For simplicity, the definitions are given for two objects but they can be generalized for n objects. Consider two objects a and b that are related to each other and updated by transactions 1 and 2, respectively. The database versions of the objects a and b at time t, i.e., D_t^a and D_t^b, are defined to be mutually consistent in the time domain if the following condition holds

$$if\ D_t^a = A_{t_1}^a\ and\ D_t^b = A_{t_2}^b\ then\ |t_1 - t_2| \leq \delta$$

where A_t^a refers to the correct version of the object a at time t. For example, say the object is the altitude of an aircraft and at time t_1 the aircraft was at an altitude x. It is possible that at time t ($t_1 < t$) the database version reads x. So $D_t^a = A_{t_1}^a$ means that the version of the object in the database at time t is actually the correct version at time t_1. We refer to $|t_1 - t_2|$ as the mutual gap between the two transactions 1, 2. Assuming that the transactions update the database with the current version, for each object a, $D_t^a = A_{t_1}^a$ where t_1 is the time at which the value of a was last updated. So the mutual gap between the two objects a and b is just the difference in the time when they were last updated by their respective update transactions.

In this paper, we propose novel solutions for guaranteeing mutual consistency in the presence of *non-preemptable* query executions. The main contributions of the paper are:

- Formulas that give the maximal value of mutual gaps among a set of data reads.
- A design approach to decide the period and relative deadline of a query so that it would guarantee mutual consistency.
- A design approach to decide if a given set of queries with relative deadlines and periods can guarantee mutual consistency.
- Methods of reducing the complexity of our proposed approaches for both harmonic periods and general cases.

In what follows, we show the solution approach to the single query case in Section 2 and also investigate it empirically through various experiments. Section 3 looks at a solution for the multiple-query case and in Section 4 we see how harmonic periods can improve our solution. We then look at some related work in Section 5 and present conclusions in Section 6.

2 Checking Single Query for Feasibility

We begin by formally stating the problem along with the assumptions made. Let $\{X_i\}_{i=1}^m$ be a set of real-time data. The validity of each $X_i\,(1 \leq i \leq m)$ is maintained by an update transaction τ_i. Let D_i, $P_i, C_i, (1 \leq i \leq m)$ denote the relative deadline, period, actual execution time of transaction τ_i, respectively. Assume that $D_i \leq P_i$ according to the *More-Less* approach in [22]. The D_i's and P_i's together represent the schedule of the transactions.

The problem: Given a periodic query with relative deadline D, period P, and execution time C, determine if a given schedule of update transactions can preserve the mutual consistency requirements of that query.

The assumptions made to derive a solution for the problem are listed below:

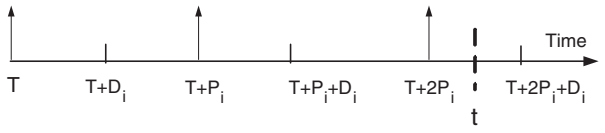

Figure 1. If we are reading the object at time t, the *earliest possible time* of update in this example would be $T + P_i$

- There is a single valid version of the objects and the queries only read valid values.
- All the data are read by an instance of the query at its start time.
- It takes no time for queries to read the values.
- An object is sampled at the time of updating the object in the database.
- The query and the transactions run on separate CPUs.
- The queries considered are non-preemptable.

2.1 Intuition

Since the schedule of the update transactions in the form of periods and relative deadlines is already known to us, our strategy is to derive a formula that gives the maximum value of the mutual gap ($\mu\,(t)$) at all points on the timeline i.e., at all instances of time beginning with 0. Time 0 is the instant at which all the transactions commence synchronously. Then we check the value of P and D to see if our query can avoid reading data at those points where $\mu\,(t)$ is greater than the mutual consistency requirement (δ). The same strategy can be modified to choose a suitable P and D for the query.

2.2 Computing $\mu\,(t)$, the Mutual Gap

Supposing the objects X_1 and X_2 are related and there is a mutual consistency requirement on them. We are interested in the $\mu\,(t)$ at any time t, between X_1 and X_2. To do that, note that we only need to know the earliest and latest time before t, when the current values being read could have been updated. For that we first look at two possible cases of "time of update". Case I assumes that the time of update of objects is the commit time of transaction. Case II assumes that the objects could be updated before the commit time of transactions; Assuming that the update will commit, there is no risk of a dirty read and hence we also assume that the query can read the updated object before the corresponding update transaction has committed. We look at the cases in order.

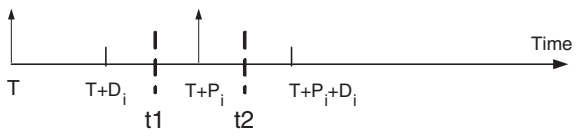

Figure 2. If we are reading the object at time t_1, the *latest possible time* of update would be $T + D_i$, in case of t_2 it would be t or $T + D_i$ respectively depending upon whether $T + P_i + C_i \leq t_2$ or not.

Case I

Here we assume that the value cannot be updated earlier than the commit time of the transaction. Consider Figure 1 for example, if the data is read at t, it could not have been written earlier than $T + P_i + C_i$, where T, P_i, C_i are the starting time, period and actual execution time of the transaction τ_i, respectively.

Basically if there is an instance of a periodic transaction that has its deadline before t, you cannot have the earliest update of the data available to the query at time t, started before the time of beginning of that instance, $T + P_i$ in Figure 1. For values of t that appear before the commitment of the first instance of an update transaction, there is no previous version of the data for a query to read. Hence it should be easy to see that e_i, the *earliest possible time* of update for X_i, would be

$$e_i = P_i \cdot \lfloor \frac{t - D_i}{P_i} \rfloor + C_i;$$

if $e_i \leq 0$ then $\{$
$e_i = e_i + P_i$;
if $e_i > t$ then $e_i =$ undefined.$\}$

For deriving l_i, the *latest possible time* of update for X_i, consider Figure 2. We just want to know the time when last update transaction committed before t and if it had at least C_i time before t so that the update was over. Hence,
if $P_i \cdot \lfloor \frac{t}{P_i} \rfloor + D_i \leq t$ then $l_i = P_i \cdot \lfloor \frac{t}{P_i} \rfloor + D_i$;
else if $P_i \cdot \lfloor \frac{t}{P_i} \rfloor + C_i \leq t$ then $l_i = t$;
else $\{$
$l_i = P_i \cdot \lfloor \frac{t}{P_i} \rfloor - P_i + D_i$;
if $l_i < 0$ then $l_i =$ undefined. $\}$ □

Case II

After reading the derivation of $\mu(t)$ for the above case, it should not be difficult to do it again. In this case, the only difference is that the time of update is not constrained to be at the commit time, the value could have been updated at any time during the transaction execution. So, in the situation of Figure 1, the *earliest possible time* becomes $T + P_i$ instead of $T + Pi + C_i$. In general,

$$e_i = P_i \cdot \lfloor \frac{t - D_i}{P_i} \rfloor;$$

if $e_i \leq 0$ then $\{$
$e_i = e_i + P_i$;
if $e_i > t$ then $e_i =$ undefined.$\}$

To derive the *latest possible time*, unlike in the previous case, we do not have to worry whether the last transaction of interest had enough time to commit before time t. Thus,

$$l_i = min\left(P_i \cdot \lfloor \frac{t}{P_i} \rfloor + D_i, t\right). \quad \square$$

Hence the maximum mutual gap ($\mu(t)$) for X_1 and X_2 would be

$$\mu(t) = max\left\{|e_1 - l2|, |e_2 - l_1|\right\}.$$

Table 1 shows the $\mu(t)$ for $P_1 = 10, D_1 = 5, C_1 =$

$t\%30$	$\mu(t)$	$t\%30$	$\mu(t)$	$t\%30$	$\mu(t)$
0	8	10	8	20	13
1	8	11	8	21	13
2	8	12	8	22	13
3	16	13	11	23	21
4	17	14	12	24	22
5	18	15	13	25	8
6	18	16	13	26	8
7	18	17	13	27	8
8	18	18	13	28	8
9	18	19	13	29	8

Table 1. The $\mu(t)$ values given periodically mod 30, for $P_1 = 10, D_1 = 5, C_1 = 2, P_2 = 15, D_2 = 10, C_2 = 2$, derived using Case I.

$t\%30$	$\mu(t)$	$t\%30$	$\mu(t)$	$t\%30$	$\mu(t)$
0	15	10	10	20	20
1	16	11	11	21	21
2	17	12	12	22	22
3	18	13	13	23	23
4	19	14	14	24	24
5	20	15	15	25	10
6	21	16	15	26	10
7	22	17	15	27	10
8	23	18	15	28	10
9	24	19	15	29	10

Table 2. The $\mu(t)$ values given periodically mod 30, for $P_1 = 10, D_1 = 5, C_1 = 2, P_2 = 15, D_2 = 10, C_2 = 2$, calculated using Case II.

$2, P_2 = 15, D_2 = 10, C_2 = 2$ derived using Case I. Note that only values mod 30 ($= LCM\,(10, 15)$) are shown, because they repeat with period 30, except for some small values of t, that come before the commitment of even the first instance of the transaction, in which case $\mu\,(t)$ wont be defined. The experiments reported throughout the rest of the paper calculate $\mu\,(t)$ using Case I. $t\%30$ stands for t(mod 30). Assume $t \geq 30$ because for t in (0,29), $\mu\,(t)$ might be undefined as explained above and discussed in Case I.

Table 2 shows the same thing for Case II. It is also assumed that $t \geq 30$ for the same reasons as discussed for Table 1.

In case the mutual consistency is specified on a set, say $X_1, \ldots, X_k$ rather than just a pair, we will have the set of earliest and latest possible times of updates, $e_1, \ldots, e_k$ and $l_1, \ldots, l_k$, respectively. Let e_{max}, e_{min} be the maximum and minimum elements respectively in the earliest possible time set and similarly define l_{max}, l_{min}. Then it should be easy to see that

$$\mu\,(t) = max\,\{|e_{max} - l_{min}|, |e_{min} - l_{max}|\}$$

2.3 Checking for Mutual Consistency

We have to check the value of P,D,C (the period, relative deadline, execution time of the query) to see that all the mutual consistency constraints are satisfied. Now any constraint would be applied on a set of data $X_1, X_2, \ldots, X_k$. Note that a constraint in this section means the mutual consistency constraint unless it is specified otherwise. Let us first look at the case when D, C are 0: the query would read the values periodically at only the points divisible by $gcd\,(P, L)$, where $L = LCM\,\left(\{P_i\}_{i=1}^{k}\right)$. Here we assume that all data are read by the query at one instant and it takes no time to read them. Therefore we only need to check for the mutual consistency at such points on a timeline of L by verifying that $\mu\,(t) \leq \delta$. This checking can be done separately across all other constraints as well since checking for one constraint is independent of the other. For example in the previous example shown as in the Table 1 a query with period 15 can satisfy a mutual consistency requirement of 13 between X_1 and X_2. Now if $D \neq 0$ and $C < D$, then the query has a "slack" to execute, meaning there is a set of points before the deadline when the query can start unlike the previous case. These are the points where $\mu\,(t) < \delta$ and also there should be enough time for the query to finish execution by the deadline. The point chosen should be such that the other constraints are also satisfied, so in this case the checking of all the constraints has to be done together on one timeline which would be the LCM of the individual timelines of all the mutual consistency constraints. This way on the same timeline we could write all the $\mu\,(t)$ values necessary and then check if all the constraints are satisfied.

2.4 Choosing a Suitable P or D

Besides checking for feasibility given P and D, we can also examine how good a given value of P is by calculating the minimum value of D for which the query satisfies the mutual consistency constraints. This can be done just as in the feasibility checking, instead of checking if there is one point in the given slack, you now check for the minimum slack at each point where the query could arrive and the maximum of all of this is the D necessary.
It might also be useful to know the shortest P that will work given the relative deadline D. To do this just calculate the points on the timeline which are intolerable in terms of the mutual consistency constraints. Then find out those points, using the value of D, at which the query should not start, i.e., starting from those points there is no point within D slack that can satisfy mutual consistency. They appear as a set of intervals viz. $(a_1, b_1), (a_2, b_2), \ldots, (a_k, b_k)$. We can find the minimum P by finding the least integer $P > F$ such that $gcd(P, L)$ does not divide any element in the above set of intervals, where $F = \max_{i=1}^{k}\{b_i - a_i\}$. To see why $P > F$, note that if $P \leq F$ then P will necessarily divide at least one number in the interval (a_i, b_i), where $b_i - a_i = F$, and hence will $gcd(P, L)$, a contradiction. To illustrate the procedure let us again consider the scenario in Table 1 and suppose the mutual consistency constraint δ is of 11. Assume 0 execution time for the query. Then the points at which the mutual consistency is satisfied are $0, 1, 2, 10, 11, 12, 13, 25, 26, 27, 28, 29$. Let $D = 5$, then the periodic instances of query should not start at the time points 3, 4, 14, 19. Now the minimum $P > 5$ that does not divide any integers in $[3, 4]$, $[14, 19]$ is 10, hence $P = 10$.

2.5 Empirical Evaluation

To evaluate our approach we used the real-world traces from [20] containing information about update frequency of various newspaper websites as shown in Table 3. We label them respectively as transactions 1,2,3,4 in the order they appeared in Table 3. Let the schedule be represented as $P_1 = 26, D_1 = 5, C_1 = 2, P_2 = 12, D_2 = 5, C_2 = 2, P_3 = 21, D_3 = 5, C_3 = 2, P_4 = 5, D_4 = 3, C_4 = 1$, the deadlines for all except the last transaction have been chosen to be 5 for the sake of simplicity. Now clearly 2 and 3 are related, so we try to impose a mutual consistency requirement on them and then look for the feasible values of P and D. We find out the minimum slack with which each value of P makes the query feasible, i.e., the minimum D at which the query could be run with a given P so that the mutual consistency constraints are satisfied. Figure 3 shows the graph for δ of 5, 10, and 15 respectively. As can be seen when δ is very stringent as with 5, the values of D needed are too large making most of the periods unsuitable to be

Trace	Time Period	Num. Updates	Avg. Update Frequency
CNN	Aug 7 13:04 - Aug 9 14:34	113	every 26 min
NY Times (AP)	Aug 7 14:07 - Aug 9 11:25	233	every 11.6 min
NY Times (Reuters)	Aug 7 14:12 - Aug 9 11:25	133	every 20.3 min
Guardian	Aug 6 13:40 - Aug 9 15:32	902	every 4.9 min

Table 3. Trace Workloads from [20]

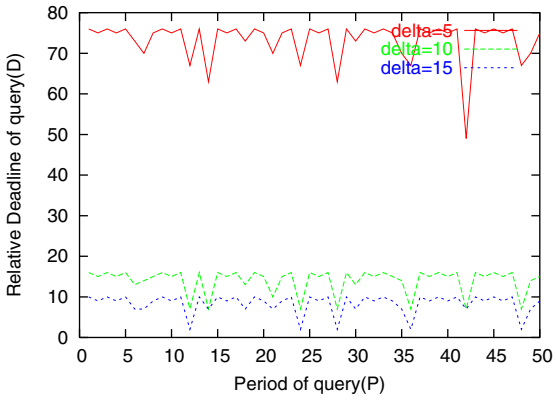

Figure 3. Minimum D given P ; when $\delta = 5$ between objects 2 and 3, the values of D are mostly very large, but with $\delta = 10, 15$ the situation improves.

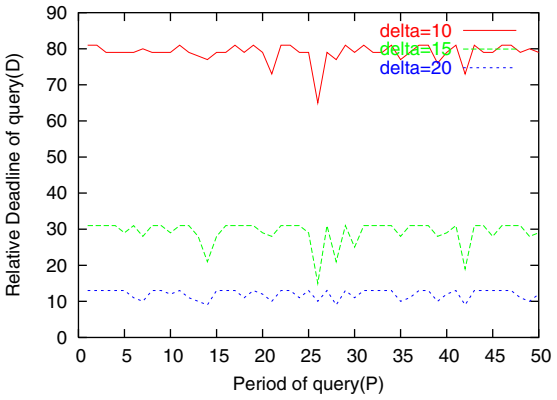

Figure 4. Minimum D given P ; $\delta = 10, 15, 20$ among all objects now, once again for $\delta = 10$, very large values of D are needed for a particular value of P to make the query feasible.

chosen. A period of 42 as shown would need minimum D but that too is as large as 47. When δ is 10 the situation is much better with lower deadlines and when δ is as tolerable as 15 most of the values of P look good. Similarly, Figure 5 shows the plot of minimum possible P with which the query becomes feasible versus the D. The situation for $\delta = 5$ is once again very poor with P being as high as 80 for even values of D till 40. But the situation improves sharply with a δ of 10. As can be seen, one can now attain feasibility at even values of D till 10, with values of P less than 10. With the help of these graphs a user can obtain insight about the best combination of P and D.

To extend the experiment on cases with the mutual consistency constraints on a set,we conducted the second set of experiments imposing a δ of 10,15,20 on the entire set of data objects. Similar graphs are shown in Figure 4 and 6. Compared to Figures 3 and 5, we observe one notable difference: P obtained in Figure 6 is much larger than that in Figure 5 while D obtained in Figure 4 is similar to that in Figure 3. This is because P is more sensitive than D when the number of objects increases, i.e., it is more difficult to get a period that works.

3 Multiple Queries

In case of multiple queries, the set of feasible points for different queries on the timeline could intersect. Hence we have to decide which point be allotted to which query so that all of them read mutually consistent data. Note that the timeline we are talking about is of length $l = LCM(P_{q_1}, P_{q_2}, \ldots)$, where P_{q_i}s are the periods of the queries. For the case of multiple queries we assume the following:

- The queries are non-preemptable.
- $\forall$ queries, $D \leq P$.
- $\forall$ queries, P divides l.

Let $v_{i,j}$ denote the boolean variable indicating if at time i, query j starts, where $j = 0$ means no query starts. Then we have

$$\forall i, \sum_{j=0}^{k} v_{i,j} = 1;$$

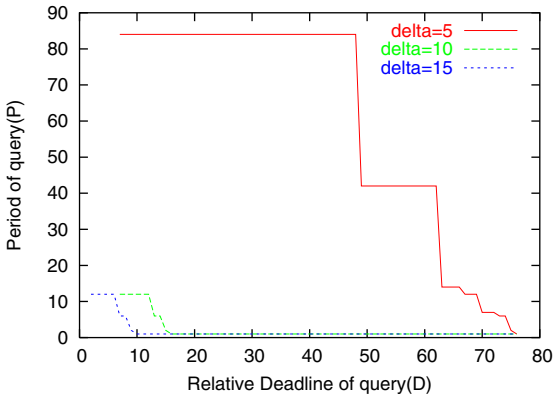

Figure 5. Minimum P given D ; $\delta = 5, 10, 15$ between objects 2 and 3

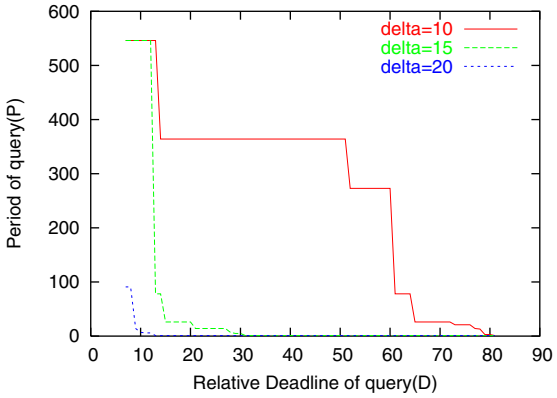

Figure 6. Minimum P given D ; $\delta = 10, 15, 20$ between all objects

Also, the possibility of a query to start on any of the possible points within the deadline can be expressed in the form of constraints as

$$v_{i_1,j} + v_{i_2,j} + \ldots + v_{i_l,j} = 1;$$

where $i_1, \ldots i_l$ are the points where the mutual consistency constraints are satisfied and also there is enough time for the query to finish execution before the deadline. The query has to choose exactly one amongst these points to start. This is because we assume that all data objects are read at the point that a query starts so it is important that the query start at a point where mutual consistency is satisfied . The equality holds because we assume that $\forall$ queries i, $D_{q_i} \leq P_{q_i}$. This assumption is necessary since if $D_{q_i} > P_{q_i}$ then the set of "allowed" points would intersect with those for the next instance and hence the equality is no longer valid. For example, say the period is 5, deadline is 8 and the first instance of the query has to choose a point to start among 6,8,11,12 and the next instant amongst 11,12,14. So it is possible that the first instance starts executing at 11, the second at 12 (ignore the execution time for the time being). So $v_{6,p} + v_{8,p} + v_{11,p} + v_{12,p} \neq 1$. Hence the assumption holds. Now if the execution time of query j is c_j, then if query j chose a mutual-consistency satisfying point t to start then we cannot have any other query starting from $t+1$ to $t + c_j - 1$ on the same processor, hence

$$v_{t+i,0} \geq v_{t,j}\ ,\ \ 1 \leq i < c_j\ \forall t$$

Note that we are dealing with non-preemptable queries here. So the problem is equivalent to solving the above system of equations which is a 0-1 ILP. The problem in general is NP-hard but [1],[2],[3] solve the above problem to a very good approximation. In our experiments, we use *OPBDP* [2] – a *Davis-Putnam Based Enumeration* algorithm for solving (non)linear 0-1 (or pseudo-boolean) optimization problems with integer coefficients.

Now we show how to extend our solution in order to relax the second and third assumption as listed at the beginning of this section. If we see the rationale for introducing the assumption $D \leq P$, which was that the "feasible" set of points for various instances of the same query intersect, the solution to revoke this assumption becomes apparent as well. We just have to treat the various instances of the same query as different ones in case $D > P$, the overhead will be paid in terms of the number of variables that would increase.

The assumption that P divides l was used once again to simplify the calculations, so that a timeline of l works. In case the periods of queries do not divide l, the length of timeline would be $LCM\,(l, l_1)$, where l_1 is the LCM of the periods of queries. So the possible values of t would increase, which would once again mean more variables.

3.1 Empirical Evaluation

For the case of more than one query, we demonstrate the estimated percentage of queries that can be scheduled for a given mutual consistency constraint. Assuming a relative deadline (D) of 10 and execution time (C) of 2 we randomly choose the periods for the queries between D and $LCM\,(P2, P3)$ such that it divides $LCM\,(P2, P3)$. We do this 20 times and then see how many times the query set was schedulable. As stated above we use [2] for our experiments with a timeout of 30 minutes. This timeout may not be sufficient especially when the size of query set increases, so we have run our experiments only for 2,3,4 queries. The results with the mutual consistency constraints of 10,15,20 are presented in Table 4. It should be expected that with larger, i.e., more lenient, mutual consistency constraints the number of queries schedulable would increase and with increase in the size of query set this number should decrease.

	Size of query set $\longrightarrow$		
$\delta \downarrow$	2	3	4
10	16	15	0
15	20	10	14
20	20	19	20

Table 4. The number of feasible query sets (out of 20 randomly generated sets), when the number of queries in each set were 2,3,4 and $\delta = 10, 15, 20$.

This is not strictly followed in Table 4, but that is probably because of inadequate timeout given. However, the number of queries schedulable come out to be fairly good, despite the low timeout given, which is encouraging.

4 Harmonic Periods and Scaling Down ILP

Since the calculations so often involve LCM, we investigate the effect of using harmonic periods. We define the sequence of periods $P_1, P_2, \ldots, P_k$ as harmonic if $P_1 < P_2$ implies P_1 divides P_2. Now note that the length of timeline in both single- and multiple-query cases was LCM of the periods of all transactions. In case we have periods of transactions that are harmonic, the LCM would be the maximum period, say P_k. So the change in scale is from the order of worst case $P_1 \times P_2 \times \ldots \times P_k$ to P_k.

As we saw in the multiple-query case the number of variables could get very large and hence the ILP could get intractable and the ILP solvers might give up without giving any solution. Although these calculations are done offline, the above problem might still arise in some circumstances. If the periods of queries are harmonic we show that instead of one big ILP, we can reduce the solution to many scaled down small ILPs. Let $P_1 < \ldots < P_k$ be the periods of the queries and all the assumptions of multiple-query solution in Section 4 hold true. Since $D_i \leq P_i$ ($\forall 1 \leq i \leq k$), note that any query that starts between time 0 and $P_k - 1$ has to finish in the same interval as well, assuming as already stated all queries commence synchronously at time 0. This should not be difficult to see, for example query k has $D_k \leq P_k$, so it will finish by D_k. Any other query with period, say P_1, would have its instances starting periodically at $\{0, P_1, 2P_1, \ldots, P_k, \ldots\}$. As P_1 divides P_k, and all those instances have to finish by $\{D_1, P_1 + D_1, \ldots\} \leq \{P_1, 2P_1, \ldots, P_k, \ldots\}$. This was shown to establish that actually the calculations for the intervals $(0, P_k - 1), (P_k, 2P_k - 1)$, and so on, can be done just exactly as shown for multiple-query case without concerning the other intervals. This will reduce the number of variables from $l.(n+1)$ to $P_k.(n+1)$ and the number of inequalities from approximately $l + l/P_1 + \ldots + l/P_k + C.n.l$ to $P_k + P_k/P_1 + \ldots + P_k/P_k + C.n.P_k$, where C is the execution time of queries (assume same execution time), n is the number of queries and $l = LCM$ of the periods of transactions. As one can see the ILP is heavily scaled down, of course now instead of one ILP we have l/P_k ILPs, but solving ILPs of this scale should not be a problem and the important thing is now we do not have to worry about ILP solvers giving up without any solution when the "parameters" become very large.

There was nothing holy about harmonic periods that brought down the scale of our ILPs, in fact take the $l_1 = LCM$ of the period of all queries and then we can break the big ILP into l/l_1 ILPs, just as in the above case since feasibility on the timeline could be checked separately for $(0, l_1 - 1), (l_1, 2l_1 - 1)$ and so on. Thus the formidable looking ILP can be scaled down in general as well this way.

5 Related Work

Recently, there has been extensive work in real-time database systems (RTDBSs) for guaranteeing data *freshness* [18, 11, 12, 9, 16, 17, 5, 19, 21, 10, 25, 7, 8, 4, 22, 23, 6, 24]. Data freshness is maintained in [16, 9] by using *Half-Half* approach where an update transaction's period and relative deadline are defined to be half of the validity interval length of the updated object. *More-Less*, studied in [22, 25, 4], reduces update workload compared to *Half-Half* while guaranteeing data freshness. In *More-Less*, the period of an update transaction is more than half of the validity interval length, while its corresponding relative deadline is less than half of the validity interval length of the same object. If the sum of the period and deadline is equal to the validity interval length, and the update transaction set can be scheduled by the *deadline monotonic* scheduling algorithm [13], then data freshness can be guaranteed. Recently, the deferrable scheduling algorithm (*DS-FP*) has been studied in [23, 24]. Compared to *More-Less*, *DS-FP* reduces update workload further by adaptively adjusting the separation of two consecutive jobs while guaranteeing *data freshness*.

In [6], a safety-critical automotive application, adaptive cruise control, is studied. It deals with critical data and involves deadline bound computations on data gathered from the automobiles' environment. These applications have stringent requirements on the freshness of data items and completion time of the tasks. Gustafsson and Hansson study guaranteeing the validity constraint of real-time data for embedded systems in a vehicular application [7, 8]. [8] presents a vehicular application with embedded engine control systems, and an on-demand scheduling algorithm for enforcing base and derived data freshness. [7] proposes an algorithm for updating data objects that can skip unneces-

sary updates allowing for better utilization of the CPU in the vehicular application. The concept of *data-deadline* is proposed in [21] along with data-deadline based scheduling. Forced-wait and similarity-based scheduling techniques are additionally used to maintain the freshness of real-time data and meet transaction deadlines in RTDBSs.

However, there is much less work related to *mutual consistency*. Urgaonkar et. al. study various approaches to maintain mutual consistency in the web domain [20]. Song and Liu [18] study real-time transaction scheduling to maintain temporal consistency including both absolute consistency and relative consistency (a.k.a. mutual consistency) in RTDBSs. The performance of several concurrency control algorithms for maintaining temporal consistency are studied in [18]. However, it is assumed in [18] that earliest deadline first and rate monotonic scheduling algorithms [14] are used for on-line scheduling of queries and transactions. Our work is different as we focus on deriving schedules of queries off-line, which is achieved by choosing a right time to start queries such that they can preserve mutual consistency of accessed objects. Since we only assume that periods and deadlines for update transactions, schedulability wise, for non-preemptive query schedules are known, a valid assumption for real-time applications, it is appropriate for offline scheduling decisions.

6 Conclusions

We have proposed approaches to check for mutual consistency, for both single- and multiple-query cases. The approach developed here should be viewed as a first-cut solution to the problem of satisfying the mutual consistency requirements as query input. In the single-query case, we show how to pick a suitable period and deadline for the query. In the multiple-query case, we show how to determine if a given set of queries with relative deadlines and periods can guarantee mutual consistency. Besides that, we also show means of scaling down the complexity of solution for the multiple-query case.

The mutual consistency problem in the real-time databases has not been paid much attention until now and there are no other approaches for our solutions to be compared with. Our solution is based on calculations done offline and assumes non-preemptable queries. The proposed approach needs to be extended to work for preemptable queries. Also, the queries may not always need "completely" fresh data. For some applications, it is acceptable for queries to access stale data with bounded inconsistency so that mutual consistency can be achieved. In such cases, we need to maintain multiple versions of an object. Analysis and performance of our solution with real world data can provide insights into weakening some of the assumptions.

References

[1] http://www.cplex.com/.

[2] http://www.mpi-sb.mpg.de/units/ag2/software/opbdp/.

[3] http://www.eecs.umich.edu/∼faloul/.

[4] D. Chen and A. K. Mok. Scheduling similarity-constrained real-time tasks. In *ESA/VLSI*, pages 215–221, 2004.

[5] R. Gerber, S. Hong, and M. Saksena. Guaranteeing end-to-end timing constraints by calibrating intermediate processes. In *IEEE Real-Time Systems Symposium*, December 1994.

[6] G. Goud, N. Sharma, K. Ramamritham, and S. Malewar. Efficient Real-Time Support for Automotive Applications: A Case Study. In *12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, 2006.

[7] T. Gustafsson and J. Hansson. Data management in real-time systems: a case of on-demand updates in vehicle control systems. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 182–191, 2004.

[8] T. Gustafsson and J. Hansson. Dynamic on-demand updating of data in real-time database systems. In *ACM SAC*, 2004.

[9] S. Ho, T. Kuo, and A. K. Mok. Similarity-based load adjustment for real-time data-intensive applications. In *IEEE Real-Time Systems Symposium*, 1997.

[10] K. D. Kang, S. Son, J. A. Stankovic, and T. Abdelzaher. A qos-sensitive approach for timeliness and freshness guarantees in real-time databases. In *EuroMicro Real-Time Systems Conference*, June 2002.

[11] T. Kuo and A. K. Mok. Real-time data semantics and similarity-based concurrency control. In *IEEE Real-Time Systems Symposium*, December 1992.

[12] T. Kuo and A. K. Mok. Ssp: a semantics-based protocol for real-time data access. In *IEEE Real-Time Systems Symposium*, December 1993.

[13] J. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic real-time tasks. In *Performance Evaluation*, pages 237–250, 2(1982).

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

[15] D. Locke. Real-time databases: Real-world requirements. In K.-J. L. Azer Bestavros and S. H. Son, editors, *Real-Time Database Systems: Issues and Applications*, pages 83–91. Kluwer Academic Publishers, 1997.

[16] K. Ramamritham. Real-time databases. *Distributed and Parallel Databases*, pages 199–226, 1(1993).

[17] K. Ramamritham. Where do time constraints come from and where do they go ? *International Journal of Database Management*, 7(2):4–10, 1996.

[18] X. Song and J. W. S. Liu. Maintaining temporal consistency: Pessimistic vs. optimistic concurrency control. In *IEEE Transactions on Knowledge and Data Engineering*, volume 7, pages 786–796, October 1995.

[19] J. A. Stankovic, S. Son, and J. Hansson. Misconceptions about real-time databases. In *IEEE Computer*, volume 32, pages 29–36, June 1999.

[20] B. Urgaonkar, A.G.Ninan, M.S.Raunak, P.Shenoy, and K.Ramamritham. Maintaining mutual consistency for cached web objects. In *International Conference on Distributed Computing Systems*, pages 371 – 380, April 16 -19, 2001.

[21] M. Xiong, K. Ramamritham, J. A. Stankovic, D. Towsley, and R. M. Sivasankaran. Scheduling transactions with temporal constraints: Exploiting data semantics. *IEEE Transactions on Knowledge and Data Engineering*, 14(5):1155–1166, 2002.

[22] M. Xiong and K. Ramamritham. Deriving deadlines and periods for real-time update transactions. *IEEE Transactions on Computers*, 53(5):567 – 583, 2004.

[23] M. Xiong, S. Han, and K. Lam. A deferrable scheduling algorithm for real-time transactions maintaining data freshness. In *IEEE Real-Time Systems Symposium*, 2005.

[24] M. Xiong, S. Han, and D. Chen. Deferrable scheduling for temporal consistency: Schedulability analysis and overhead reduction. In *12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, 2006.

[25] M. Xiong, B. Liang, K. Lam, and Y. Guo. Quality of service guarantee for temporal consistency of real-time transactions. In *IEEE Transactions on Knowledge and Data Engineering*, 18(8), pp. 1097-1110, 2006.