Yonghong and Xiangdong *EURASIP Journal on Embedded Systems* (2016) 2016:9
DOI 10.1186/s13639-016-0030-1

EURASIP Journal on Embedded Systems

REVIEW

Open Access

Research on recovery strategy in embedded real-time main memory databases

Tan Yonghong[1*] and Yin Xiangdong[2]

Abstract

In order to recover data from embedded real-time main memory databases effectively and efficiently, this paper proposes a real-time log-based recovery approach. With respect to the real-time requirement in embedded systems, we classify the consistency in real-time main memory databases into data and transaction consistencies, analyze them theoretically, design rules for correct recovery strategy, and propose real-time log-based recover algorithms for different types of transactions. The experiments show that the proposed approach is more effective and efficient than methods in both traditional and eXtremeDB database systems.

Keywords: Embedded system, Real-time main memory database, Recovery strategy, Consistency

1 Review

With the development of embedded systems, the application of databases in embedded systems [1] is a hotspot in both industry and academia. Embedded systems work in an environment without manual intervention, so when a fault occurs in these systems, they need to diagnose the fault and recover it automatically all by themselves [2]. The main memory databases [3, 4] can reduce the I/O operations greatly while running, and satisfy the real-time requirement of embedded systems, so the databases implemented in embedded systems usually work in the main memory.

In real-time main memory databases [5–7], the main copy of database works in the volatile RAM, and the data is very vulnerable, so the recovery is necessary. Meantime, the I/O operations in real-time main memory databases are few, and recovery is the only part that affects the I/O performance, so the performance of recovery is critical for real-time main memory databases [8, 9]. While recovering from a fault, real-time main memory databases need to satisfy multiple constraints [10, 11], and this pose a huge challenge for designing reasonable recovering strategies.

Checkpoint or memory snapshots [12, 13] is a commonly used program recovery strategy, but the overhead of storing states of running program is very high, and it is not suitable for embedded applications. In addition, the logs in embedded systems record the behaviors of embedded systems, and researchers use different logs to design different recovery strategies, such as partition log [14], real-time log [15], remote log [16], and operation log [17]. However, these strategies only take the requirement of real-time into consideration, and ignore other specific requirements in embedded systems, so they cannot be applied to the embedded environment efficiently. In addition, the method proposed in [13], studied the recovery strategy in main-memory, but the method is based on virtual memory snapshots. In order to improve real-time ability, Levy and Silberschatz [18] proposed an incremental recovery strategy in main-memory database.

In this paper, we analyze the consistency constraints in embedded real-time main memory databases from the perspectives of both data and transaction. Then we design some rules that an efficient recovery strategy must obey in embedded real-time main memory databases. Finally, we propose corresponding recovery algorithms for different tasks of embedded real-time main memory databases.

* Correspondence: hunantanyonghong@sina.com
[1]Experimental Training Center, Hunan University of Science and Engineering, YongZhou City, Hunan Province, China
Full list of author information is available at the end of the article

2 Analysis of consistency in embedded real-time main memory databases

In this section, we analyze the consistency of embedded real-time main memory databases from the perspective of both data and transaction.

2.1 Data consistency

The embedded real-time main memory databases include three types of data, i.e., image objects, derived objects, and invariant objects.

The objects of real world are sensed by sensors, and their values are written into the databases. The values written into the databases are image objects. An image object is an image of a real world object at some instant, and each image object has its own sampling timestamp and external validity interval.

A derived object is calculated out by a group of image objects during a transaction processing. The timestamp of a derived object is the instant when the transaction is finished, and the validity interval is the intersection of all validity intervals of image objects in the group.

An invariant object is a constant which is invariant as time goes by. The validity of an invariant object is not affected by time, so it is also called non-time series data object.

As there is a validity interval for each image object and derived object, both of them are time series data objects. The sampling time and computing time of time series data are validate only in an interval starting from the system's current time.

Definition 1. If $VI(X)$ is far less than $AT(X)$, i.e., $VI(X) << AT(X)$, then X is short time-limited data.

The data consistency of embedded real-time systems includes internal consistency, external consistency, and mutual consistency.

Definition 2. X is internal consistent, *if and only if* it satisfies the predefined integrity and consistency of traditional database systems.

Here, the internal consistency is the internal consistency in traditional database systems, and it only refers to the internal world of database systems.

Definition 3. X is external consistent, *if and only if* it satisfies $t \le ST(X) + VI(X)$.

The external consistency requires that the sampling data in a database lag the real world within a certain time.

Definition 4. A group of related data used for decision or deriving new data is a mutual consistent set R, and each R is related to a corresponding mutual validity internal R_{mvi}.

Definition 5. Let $R = \{X_1, X_2, \ldots, X_n\}$, then R is mutual consistent, if and only if $\forall X_i \in R$, $\forall X_j \in R$ and $k \ne i$, such that $|ST(X_i) - ST(X_j)| \le R_{\mathrm{mvi}}$.

If R is used to generate new data, then the mutual consistency is used to assure the values in R are generated within the common validity interval.

2.2 Transaction consistency

The embedded real-time main memory database systems interact with real world according to two behaviors. The first one is recording the states and events of the real world into the databases, and the second one is doing some acts to affect the real world. The embedded real-time transactions can be classified into data receiving transactions, data processing transactions and manipulating transactions.

Data receiving transactions sample the external environment periodically and write it into the databases. This kind of transaction generates an image object in one period, and it is a read-only and non-blocking hard real-time transaction.

Data processing transactions do read-only operations to image objects periodically or non-periodically, and read and write deriving objects or invariant objects. This kind of transaction does not interact with the real world, and is a soft real-time transaction.

Manipulating transactions read all kinds of data in a database, and do a set of actions $AS(T) = \{A_i | 1 \le i \le h\}$ to control the embedded system. If this kind of transaction exceeds the validity interval, disastrous results will be generated, so it is also a hard real-time transaction. Manipulating transactions are read-only operations, and they do not affect the consistency of databases, but they can change the states of real world.

The same as data consistency, transaction consistency in embedded real-time main memory database systems also include internal consistency, external consistency, and mutual consistency.

Definition 6. T is internal consistent, *if and only if* the value it reads and/or writes satisfies the predefined internal integrity and consistency of traditional database systems.

Definition 7. T is external consistent, *if and only if* $t \le D(T)$ and $\forall X_i \in DS(T)$, $t \le ST(X_i) + VI(X_i)$.

The external consistency of embedded real-time transactions requires that each transaction is in its validity internal, and all read/write operations are within its validity interval.

Theorem 1. *Let $MVI(T)$ be the minimum of all validate terminal instants of T while reading/writing data objects, then the final terminal instant of T is $D_R(T) = \min(D(T), MVI(T))$.*

Proof: If $MVI(T) < t < D(T)$, then $\exists X_i \in DS(T)$, such that $t > ST(X_i) + VI(X_i)$, that is, there exists some X_i, which loses the external consistency, so this violates the external consistency constraint while T reads/writes data objects. On the contrary, if $D(T) < MVI(T)$ and $t > D(T)$, then T exceeds the validity interval, and this violates the external constraint of T. So, we can have $D_R(T) = \min(D(T), MVI(T))$.

Definition 8. T is mutual consistent, *if and only if* $\forall X_i, X_j \in DS(T)$, and $i \neq j$, such that $|ST(X_i) - ST(X_j)| \leq R_{\mathrm{mvi}}$.

The mutual consistency of embedded real-time transactions means that the time interval between any two data objects is not bigger than the given value $R_{\mathrm{mvi}}(T)$.

With the same reason, when T is both external consistent and mutual consistent, then it is time consistent. A validate submit of transaction in embedded real-time systems depends not only on the internal consistency, but also on the time consistency. So, we have the following corollary.

Corollary 1. *T is consistent, if and only if the following constraints satisfy at the same time*:

(1)$\forall X_i, X_i \in DS(T)$;
(2)$CT(T) \leq D_R(T)$;
(3)$\forall X_i \in RS(T), RT_T(X_i) \leq ST(X_i) + VI(X_i)$;
(4)$\forall X_i, X_j \in RS(T)$ and $i \neq j$, such that $|ST(X_i) - ST(X_j)| \leq R_{\mathrm{mvi}}(T)$.

3 Rules for correct recovery strategy

Taking the internal consistency and time consistency of transactions and data in embedded real-time main memory databases into consideration, we present some rules for correct recovery strategies.

3.1 Non-time series data recovery rule

Rule 1. If T has not been submitted, then for $\forall X_i \in US(T)$ satisfying $S_t(X_i) = UI_T(X_i)$, execute the undo operation.

Rule 2. If T has been submitted, then for $\forall X_i \in US(T)$ satisfying $S_t(X_i) \neq UI_T(X_i)$, execute the redo operation.

Rules 1 and 2 can recover the data such that they satisfy the internal consistent constraint, and non-time series data only have internal consistent constraint, so they can also be used to recover non-time series data.

3.2 Time series data recovery rule

Rule 3. If $\exists X_i \in US(T)$ satisfying $S_t(X_i) = UI_T(X_i)$ and $t \leq ST(X_i) + VI(X_i)$, then whether or not T has been submitted, there is no need to execute any recovery operation for X_i.

Rule 4. If $\exists X_i \in US(T)$ satisfying $S_t(X_i) \neq UI_T(X_i)$ and $t \leq ST(X_i) + VI(X_i)$, then execute the redo operation for X_i.

Rule 5. If $\exists X_i \in US(T)$ satisfying $t > ST(X_i) + VI(X_i)$, then resample by starting the data receiving transaction of X_i.

Theorem 2. *Rules 3~5 can recover the internal and external state consistency of time series data*.

Proof: The recovery of time series data X_i needs to consider the consistency between its internal state $S_t(X_i)$ with its external state $UI_T(X_i)$, but not whether or not the transaction has been submitted.

When $t \leq ST(X_i) + VI(X_i)$, if $S_t(X_i) \neq UI_T(X_i)$, i.e., the internal and external states of X_i are not consistent, then whether or not T has been submitted, the redo operation should be executed according to $UI_T(X_i)$ (Rule 4); and if $S_t(X_i) = UI_T(X_i)$, i.e., the internal and external states of X_i are consistent, then whether or not T has been submitted, there is no need to execute any recovery operation (Rule 3).

When $t > ST(X_i) + VI(X_i)$, executing undo or redo operation is meaningless, and data receiving transaction should be restarted immediately to resample and recover the consistency of X_i between its internal and external states (Rule 5).

3.3 Real world state recovery rule

In embedded real-time applications, if the transactions have been submitted and have changed the real world states, there is no need to recover; and if the transactions have not been submitted, then we should do some compensation to recover the state changes of real world.

Rule 6. If T has not been submitted, then for each action that has happened, i.e., $\forall A_i \in AS(T)$, execute compensation or recovery task for A_i.

Theorem 3. *Rule 6 can recover the consistency of real world state*.

Proof: Manipulating transactions is read-only, and they do not violate the consistency of data objects. The atomicity of manipulating transactions is that, whether all actions of T, $AS(T) = \{A_i | 1 \leq i \leq h\}$, are executed or none of them is executed.

Let $OAS(T) = \{A_j | 1 \leq j \leq h\}$ be the set of actions that has been executed in T when a fault occurs. According to Rule 6, when $OAS(T) \neq \varnothing$ and $OAS(T) \neq AS(T)$, we need to compensate and recover for $\forall A_j \in OAS(T)$. So, the real world states, that have been changed, can be recovered correctly.

3.4 Transaction restart rule

No manual intervention is a typical feature of embedded real-time databases, and thus, the database systems should restart all kinds of transactions automatically when faults occur. The transactions needed to restart include two kinds. The first one is that restarting period has passed by or running time has exceeded the running period, and the second one includes non-periodic transactions, that do not finish successfully but still satisfy all consistencies.

Rule 7. For a periodic transaction *T*, if *T* does not finish normally, or *T* finishes normally and satisfies $t \geq BT(T) + P(T)$, then restart *T*.

Rule 8. For a non-periodic transaction *T*, that does not finish normally, if the following conditions satisfy at the same time, then restart *T*.

(1)$t + EET(T) \leq D_R(T)$;
(2)$\forall X_i \in RS(T)$, $t \leq ST(X_i) + VI(X_i)$;
(3)$\forall X_i, X_j \in RS(T)$, $i \neq j$ and $1 \leq i, j \leq n$, $|ST(X_i) - ST(X_j)| \leq R_{\mathrm{mvi}}(T)$.

Rule 8 is the same as Corollary 1, i.e., when a fault occurs, only when all consistencies of a transaction have been satisfied, then we can restart the transaction.

4 Log-based recovery strategy

In order to recover from faults, embedded real-time main memory databases need to log the time and triggered actions for each transaction and data. These logs include real-time transaction logs, data logs, and action logs. Taking the limits of CPU, storage and energy in embedded systems, we propose the following data recovering strategies based on the rules in the last section.

Strategy 1. If *X* is a series data with short limited time, then there is no need to log the updates of data.

Strategy 2. If $\frac{|AFI(X_i)-BFI(X_i)|}{BFI(X_i)} \geq \delta(X_i)$, then log the current data update operation; and otherwise, log nothing.

Strategy 3. Update the time series data objects immediately. That is updating the states of database before a transaction is submitted.

Strategy 4. Deferred update the non-time series data objects. That is updating the states of database when a transaction is submitted.

Strategies 1 and 2 can greatly reduce the overhead of logging the updates of time series data, and also accelerate the recovery speed. Rule 3 makes sure that the latest states of time series data can be written to the databases to reduce the redo operations of time series data. Rule 4 clears the logs of non-time series data and their undo recovery, and can further reduce the overhead of storage and recovery.

Based on the above strategies, we propose corresponding recovery algorithms for data receiving transactions, control transactions, and data processing transactions, and they are described as follows:

```
(1) Recovery algorithm for data receiving transaction:
Procedure:
1.  Scan $T'$ s logs forward;
2.  Determine $US(T)$;
3.  If $US(T) \neq \varnothing$
4.      If $t \leq ST(X) + VI(X)$ then
5.          If $S_t(X) \neq UI_T(X)$ then
6.              Let $S_t(X) \leftarrow UI_T(X)$; //rule 4
7.          Else
8.              Restart $T$; //rule 5
9.  If $t \geq BT(T) + P(T)$ then
10. Restart $T$; //rule 7
```

```
(2) Recovery algorithm for control transactions
Procedure:
1.  Scan $T'$ s logs forward;
2.  Determine $OAS(T)$;
3.  If $T$ has not committed then
4.      If $OAS(T) \neq \varnothing$ and $OAS(T) \neq AS(T)$ then
5.          For $\forall A_i \in OAS(T)$ do
6.              Do compensating or alternative action of $A$;
    //rule 6
7.  If $t + EET(T) \leq D_R(T)$ and $\forall X \in RS(T)$,
    $t < ST(X) + VI(X)$ and $\forall X_i, X_j \in RS(T)$,
    $|ST(X_i) - ST(X_j)| \leq R_{mvi}$ then
8.  Restart $T$; //rule 8
```

```
(3) Recovery algorithm for data processing transactions
Procedure:
1.  Scan $T'$ s logs forward;
2.  Determine the updated temporal data set $UTDS(T)$
    and invariant data set $UIDS(T)$;
3.  For $\forall X_i \in UTDS(T)$ do
4.      If $t < ST(X) + VI(X)$ and $S_t(X_i) \neq UI_T(X_i)$
5.          Let $S_t(X_i) \leftarrow UI_T(X_i)$; //rule 4
6.  For $\forall X_i \in UIDS(T)$ do
7.      If $T$ has committed then
8.          If $S_t(X_i) \neq UI_T(X_i)$ then
9.              Let $S_t(X_i) \leftarrow UI_T(X_i)$; //rule 2
10.     Else //$T$ has not committed
11.         If $S_t(X_i) = UI_T(X_i)$ then
12.             Let $S_t(X_i) \leftarrow BFI(X_i)$; //rule 1
13.             Restart $T$; //rule 7
14.     If $t > BT(T) + P(T)$ then
15. Restart $T$; //rule 7
```

5 Experiments

5.1 Experimental setting

In the experiments, we implement the proposed log-based recovery algorithm on the eXtremeDB embedded database [19], and compare it with the traditional recovery method and the method in eXtremeDB. The experiments contain a small database, and the operations include insert, delete, and modification. Query operations are not in our experiments, because they do not change the data in the database, and the recovery strategy does not need to consider this situation. We mainly compare the system overhead, overtime transaction ratio (ratio of transactions that exceed the validity interval), and rejecting service time (downtime). The meanings and values of experimental parameters in eXtremeDB are in Table 1.

5.2 Experimental results

Firstly, we compare the CPU utilization and log buffer utilization of the three approaches, and the results are in Figs. 1 and 2, respectively. With respect to CPU utilization, our proposed approach is higher than the other two, and the reason is that the proposed approach uses main memory to store data and it has the highest throughput. With respect to the log buffer utilization, the value of the proposed approach is the lowest, which means that the proposed method only logs necessary data and the usage of log buffer is the most efficient.

Secondly, we compare the ratio of transactions exceeding the validity interval in Fig. 3, and the average rejecting service time in Fig. 4. The ratio of transactions exceeding the validity interval is also the ratio of missing transactions. From Fig. 3, we can see that our proposed approach has the least missing transactions. Rejecting service time is also called downtime. Figure 4 illustrates that the proposed approach has the lowest average downtime.

Next, in our proposed approach, we observe the changes of overtime transaction ratio under different "per_short" (short time-limited data ratio) and "threshold" (time series data state change threshold), and the results are in Figs. 5 and 6, respectively. In Fig. 5, the order of overtime transaction ratios for different per_short is 0 > 0.5 > 0.1 > 0.3 > 0.2, which means that we must carefully select per_short to optimize the overtime transaction ratio. Here, per_short = 0.2 is the best. In Fig. 6, the order of overtime transaction ratios for different threshold is the same as that of per_short, so we can have the same conclusion.

Finally, we observe the time series data ratio of the proposed approach under different update modes, and the results are in Fig. 7. From the figure, we can see that the hybrid of deferred and immediate update modes has the lowest time series data ratio, which means that the hybrid update mode has canceled the overhead of undo recovery for the invariant data objects, and thus reduces the ratio of transactions exceeding the validity interval.

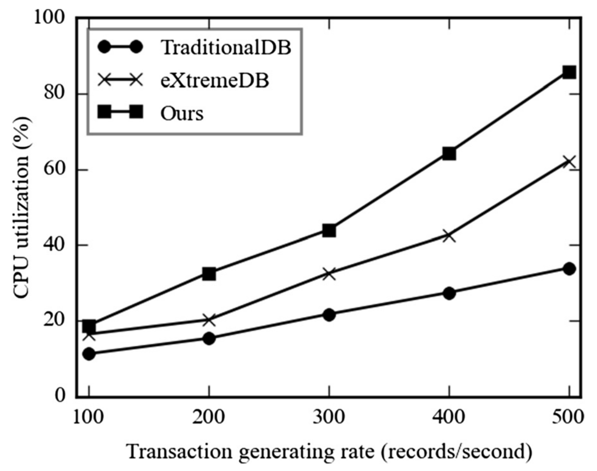

Fig. 1 Comparison of CPU utilization

Table 1 Parameter setting

Parameter	Meaning	Default value	Domain
num_imo	Image data object number	250	50~500
num_deo	Derived data object number	250	50~500
num_ino	Invariant data object number	500	200~1000
per_short	Short time limited data ratio	20%	0~40 %
vi	Time series data validate time	50ms	5ms~10s
δ	Time series data state change threshold	10%	0~20 %
period	Period of periodic transaction	50ms	5ms~10s
generation_ratio	Periodic transaction generating ratio	20/s	5~50/s
trigger_ratio	MT trigger ratio	5/s	2~10/s
update_number	Update number	2	1~3
num_actions	Action number	3	1~5
update_mode	Update mode	Hybrid	Deferred/immediate/hybrid

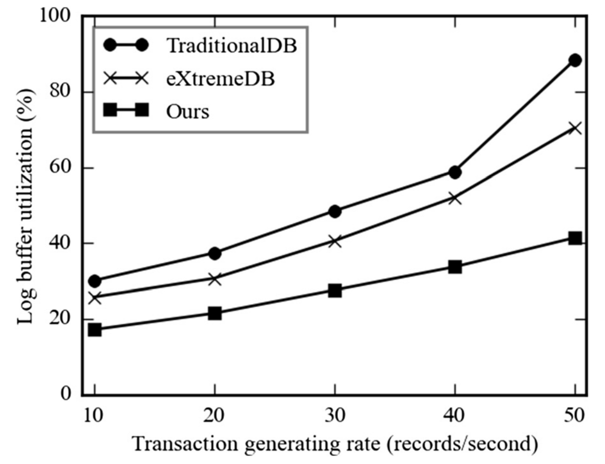

Fig. 2 Comparison of log buffer utilization

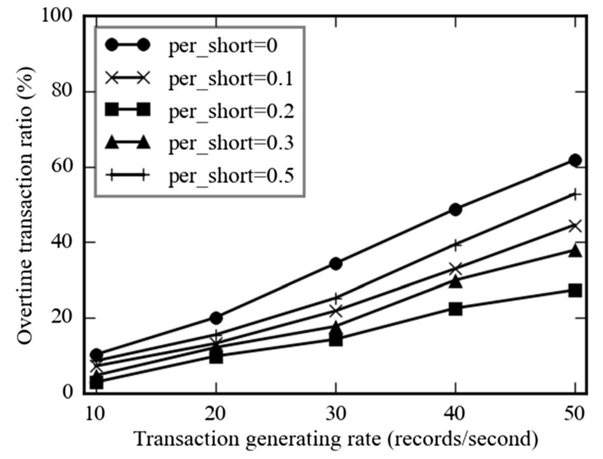

Fig. 5 Overtime transaction ratios for different "per_short"

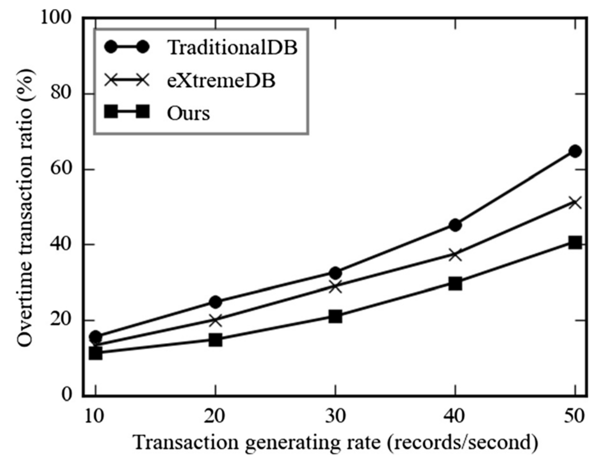

Fig. 3 Comparison of overtime transaction ratio

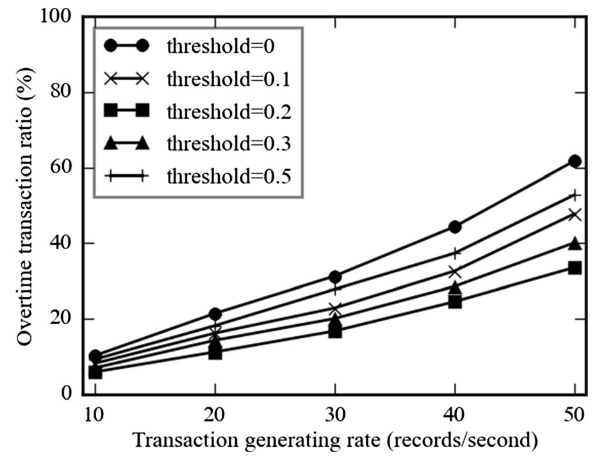

Fig. 6 Overtime transaction ratios for different "threshold"

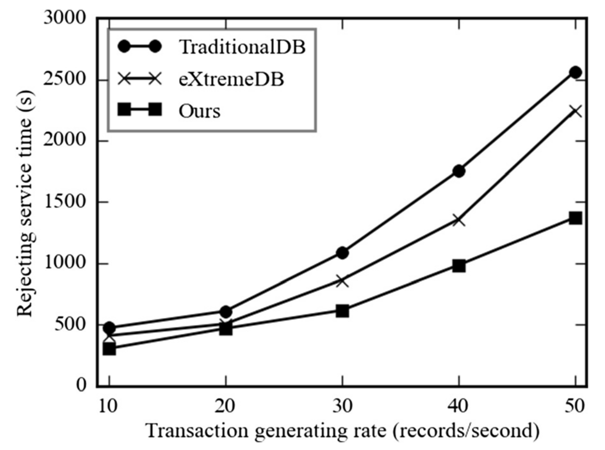

Fig. 4 Comparison of rejecting service time

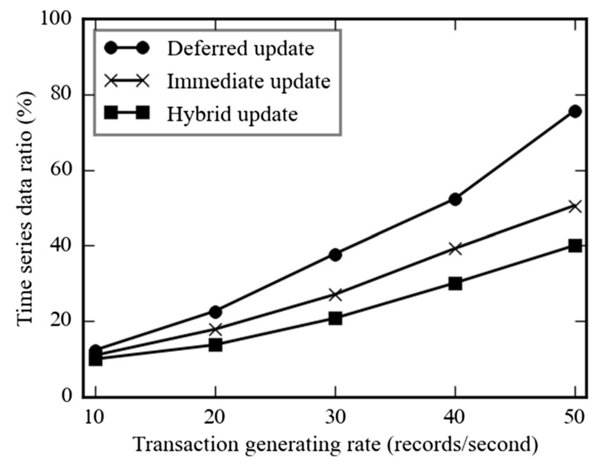

Fig. 7 Time series data ratio for different update modes

6 Conclusions

In this paper, we study the problem of data recovery strategy in embedded real-time main memory databases. Because of real-time requirement in embedded systems, consistency of embedded real-time main memory databases is different from traditional databases. We analyzed both the data and transaction consistencies in embedded real-time main memory databases, designed rules for correct recovery strategy, and proposed real-time log-based recover algorithms for different types of transactions. The experiments show that the proposed approach is more effective and efficient than methods in both traditional and eXtremeDB database systems. The proposed recovery algorithm can be integrated into the eXtremeDB database, and thus provide better recovery performance. Integrating the proposed algorithm into other main memory database will be our future work.

Competing interests

The authors declare that they have no competing interests.

Acknowledgements

The work was supported by the following funds: Hunan Provincial Natural Science Foundation of China (Grant No.2015JJ6043); Hunan University of Science and Engineering; Scientific Research Fund of Hunan Provincial Education Department(Grant No.12A054); The Construct Program of the Key Discipline in Hunan University of Science and Engineering(Circuits and Systems).

Author details

[1]Experimental Training Center, Hunan University of Science and Engineering, YongZhou City, Hunan Province, China. [2]School of Electronics and Information Engineering, Hunan University of Science and Engineering, YongZhou City, Hunan Province, China.

Received: 22 December 2015 Accepted: 20 April 2016
Published online: 04 May 2016

References

1. A Nori, *Mobile and Embedded Databases[C]//Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data. ACM*, 2007, pp. 1175–1177
2. V Narayanan, Y Xie, Reliability concerns in embedded system designs. Computer **39**(1), 118–120 (2006)
3. H Garcia-Molina, K Salem, Main memory database systems: an overview. Knowl. Data Eng. IEEE Trans. **4**(6), 509–516 (1992)
4. J Stankovic, SH Son, J Hansson, Misconceptions about real-time databases. Computer **32**(6), 29–36 (1999)
5. K Ramamritham, Real-time databases. Distrib. Parallel Databases **1**(2), 199–226 (1993)
6. G Özsoyoğlu, RT Snodgrass, Temporal and real-time databases: a survey. Knowl. Data Eng. IEEE Trans. **7**(4), 513–532 (1995)
7. K Ramamritham, SH Son, LC Dipippo, Real-time databases and data services. Real-time Syst. **28**(2-3), 179–215 (2004)
8. KH Kim, HO Welch, Distributed execution of recovery blocks: an approach for uniform treatment of hardware and software faults in real-time applications. Comput. IEEE Trans. **38**(5), 626–636 (1989)
9. RM Sivasankaran, K Ramamritham, JA Stankovic et al., *Data Placement, Logging and Recovery in Real-Time Active Databases[M]//Active and Real-Time Database Systems (ARTDB-95)* (Springer, London, 1996), pp. 226–241
10. Soparkar NR, Silberschatz A, Korth HF. Time-constrained transaction management: real-time constraints in database transaction systems. Kluwer Academic Publishers; 1996.
11. MI Seltzer, MA Olson, *Challenges in Embedded Database System Administration[C]//Proceeding of the Embedded System Workshop*, 1999, pp. 29–31
12. GM Liao, JP Li, *Research on Timely Recovery Technology of Memory Database[C]//Wavelet Active Media Technology and Information Processing (ICWAMTIP), 2012 International Conference on. IEEE*, 2012, pp. 268–271
13. A Kemper, T Neumann, *HyPer: A hybrid OLTP&OLAP Main Memory Database System Based on Virtual Memory Snapshots[C]//Data Engineering (ICDE), 2011 IEEE 27th International Conference on. IEEE*, 2011, pp. 195–206
14. Lam KY, Kuo TW. real-time database systems: architecture and techniques. Kluwer Academic Publishers; 2001.
15. LC Shu, JA Stankovic, SH Son, Achieving bounded and predictable recovery using real-time logging. Comput. J. **47**(3), 373–394 (2004)
16. T Niklander, K Raatikainen, *Using Logs to Increase Availability in Real-Time Main-Memory Database[M]//Parallel and Distributed Processing* (Springer, Berlin Heidelberg, 2000), pp. 720–726
17. N Malviya, A Weisberg, S Madden et al., *Rethinking Main Memory OLTP Recovery[C]//Data Engineering (ICDE), 2014 IEEE 30th International Conference on. IEEE*, 2014, pp. 604–615
18. E Levy, A Silberschatz, Incremental recovery in main memory database systems. Knowl. Data Eng. IEEE Trans. **4**(6), 529–540 (1992)
19. MC Majhi, AK Behera, NM Kulshreshtha et al., *ExtremeDB: a unified web repository of extremophilic archaea and bacteria [J]*, 2013