A Read-Write-Validate Approach to Optimistic Concurrency Control for Energy Efficiency of Resource-Constrained Systems

Kamal Solaiman, Matthew Brook, Gary Ushaw, Graham Morgan

School of Computing Science, Newcastle University
Newcastle-upon-Tyne, United Kingdom.
{kamal.solaiman, m.j.brook1, gary.ushaw, graham.morgan}@ncl.ac.uk

Abstract - Modern smartphones feature multiple applications which access shared data on the solid state storage within the device. As applications become more complex, contention over this memory resource is becoming an issue. This leads to increased battery drain as the applications are forced to touch the solid state device repeatedly after failing to retrieve or store data due to contention from other applications. We describe an optimistic concurrency control algorithm, combining a novel Read-Write-Validate phase sequence with virtual execution. The protocol is suitable for governing transactions operating on databases residing on resource-constrained devices. Increasing energy efficiency and reducing latency are primary goals for our algorithm. We show that this is achieved by reducing persistent store access, and satisfy real-time requirements via transaction scheduling that affords greater determinism.

Keywords - Optimistic concurrency control; transactions; ubiquitous databases

I. INTRODUCTION

The demands made on hardware resources by smartphone applications are increasingly complex. Multiple applications run in parallel on these devices, raising issues with sharing the processor, memory access and network connection [18]. In this paper we focus on the shared solid state disk resource on the device. Accessing the disk is costly in terms of battery usage, so an algorithm which reduces the number of attempted data accesses will improve energy efficiency. If multiple applications attempt to access the memory concurrently, then contention problems arise, leading to more frequent disk accesses. Reducing the frequency of contention leading to a failed read or write (which must then be retried) will increase energy efficiency of the smartphone, as well as improve the throughput of memory.

Concurrency control (CC) is the primary mechanism for coordinating simultaneous access to shared data. Transactional systems rely on high performance CC protocols to achieve significant throughput while maintaining correctness [10]. Optimistic concurrency control (OCC) protocols offer an alternative to pessimistic based locking whereby executing transactions validate with each other to determine if a conflict has occurred. The mechanisms by which these conflicts are detected and handled belong to a well-established research area [1]. The primary performance concern for real-time systems is that of timeliness: transactions commit before they reach their deadline. Systems under high contention invariably struggle to allow transactions to complete before their deadline. Protocol design should allow as many transactions to meet their deadline as possible while maintaining correctness. This can come at the cost of other criteria such as throughput or response time.

Optimistic approaches are more suitable for real-time application than locking approaches. Research has focused on timestamp based techniques which show a high degree of concurrency and scalability. It also provides a smaller overhead of unnecessary rollback which is a major disadvantage in optimistic approaches. The price paid in timestamp is the large overhead of maintaining timestamp management [9], which is acceptable in conventional OCC techniques to satisfy the scalability needed for conventional database systems at the server.

Forward validation schemes [2] are a very good concurrency control approach for mobile devices in broadcast wireless environments for many reasons [12], and are widely deployed [16, 17, 20, 21]. Furthermore, a forward validation scheme has a cheaper validation cost of 1/3 of timestamp [7]. So, we argue that the forward validation scheme is a suitable OCC approach for governing transactions operating on databases on resource-constrained devices. We describe an OCC algorithm based on a forward validation approach that utilizes a read-write-validate (RWV) phase ordering to address the real-time requirements of ubiquitous databases. We argue that, while still maintaining overall system cor-

978-1-4673-2480-9/13/$31.00 ©2013 IEEE

rectness, this ordering provides significant performance improvements. We present results showing that we improve throughput while satisfying more real-time transaction deadlines. We theorized about this approach briefly in [11], and now we present results that confirm the ideas. The non-intuitive ordering we advocate, combined with the requirement of a rerun policy, improves performance by increasing energy efficiency and reducing latency.

II. BACKGROUND AND RELATED WORK

We describe related work in a manner that reflects our approach to deriving a solution. This description includes advancements made regarding the validation process (forward and backward validation schemes) and the introduction of virtual execution (rerunning aborted transaction using in-memory values) to improve overall system performance. We then identify the contribution our protocol makes as a novel departure from existing techniques.

A. Optimistic Concurrency Control

Kung and Robinson [1] proposed the OCC approach using a three-phased transaction consisting of read, validate and write phases. During the read phase, transactions access data without restriction and make a local copy of this data. Any writes are made to the transaction's local copy. The validation phase ensures that changes a transaction has made locally can be satisfied globally. Other executing transactions are considered to determine whether the write requests made by a transaction locally can be satisfied without invalidating the correctness of the overall read-write schedule. If so the transaction enters the write phase and the local changes are committed to the persistent store; otherwise the transaction must abort and restart. A transaction with no writes does not need to enter the write phase but must still validate.

B. Forward and Backward Validation

Härder proposed two schemes: Backward-Oriented and Forward-Oriented Optimistic Concurrency Control (BOCC & FOCC) [2]. In BOCC the read set of a validating transaction is compared to the write sets of all transactions that have finished the read phase before the validating transaction. In FOCC the write set of the validating transaction and the read sets of the transactions that have yet to finish the read phase are compared.

Using BOCC, the validating transaction has to be aborted. FOCC provides a degree of flexibility in that a number of resolution policies are possible. The validating transaction can be delayed and the validation phase restarted at a later time, or all conflicting transactions can be aborted so the validating transaction is allowed to commit, or only validating transactions are aborted.

FOCC has found popularity with researchers due to this flexibility in resolution policy [7, 9]. For example, to satisfy real-time requirements, conflicts can be resolved based on a transaction's deadline. However, a major drawback of FOCC is that concurrent transactions have to be blocked in their read phase while performing the validation and write phase in the critical section.

C. Virtual Execution

Virtual execution [6] allows transactions that are known to conflict to continue execution and complete the read phase with the goal of pre-fetching data for a subsequent rerun. By using the property of access invariance, significant performance gains can be made by allowing the transaction to rerun using the pre-fetched data; there is no disk I/O overhead typically required for the transaction during rerun. However, some of the pre-fetched data may have since been modified, which results in the rerun transaction operating with inconsistent data. Concurrency control techniques must be applied to overcome this problem [6].

D. Contribution

Our goal is to reduce contention of solid state disk access on smartphone devices by combining a FOCC validation scheme with a virtual execution approach. In [11] we made some observations on such a solution: (1) Transactions that enter rerun execute quicker than those in their initial run (as there is no disk access); (2) The validation phase presents a degree of non-determinism with respect to how long it will take (i.e., we can't predict how many transactions need to be validated).

As reruns execute with no disk latency, they can occur multiple times with minimal hindrance to satisfying transaction deadlines. Therefore, it is better to keep transactions in rerun until we can determine that, when they leave rerun, they satisfy their deadline requirements irrelevant of the delay imposed by the validation step. This would prioritize rerun transactions without the concern for non-deterministic latency in the validation phase.

Our assumptions regarding the structuring of a virtual execution enabled OCC implied that the write phase could be moved to before the validation phase. When transactions are in a rerun state we can offset their validation until after the write phase. There are two benefits of this approach. Writes may become visible to transactions in the read phase earlier, affording more likelihood of reading up-to-date data from disk Also overall blocking may be reduced (in the original OCC protocols, transactions in the read phase need to be blocked as a transaction commits changes to the database - such blocking is not required in our protocol, as out-of-date reads are caught by the later validation step).

The contribution made by this paper is the creation of a new virtual execution OCC in which the write phase occurs before the validation phase. We show that this approach reduces the frequency of disk accesses due to contention, leading to improved battery life for the mobile device. We further show that we improve system throughput and the likelihood that transactions complete within their deadline.

III. PROTOCOL

A. Read-Write-Validate

Our protocol fundamentally changes the order of the traditional transactional phases [1]. The write phase now follows the read phase with the validation phase occurring afterwards. Both the write and validation phases are collectively considered a single critical section, allowing only one transaction to execute in either phase [1, 2].

We use a forward validation strategy [2] in combination with a "No Sacrifice" policy [4] that guarantees a transaction entering the critical section will commit. This requires transactions conflicting with the validating transaction to be aborted. We employ a rerun policy so that transactions in their initial run will continue to the end of the read phase before being rerun.

Using a critical section around the write and validation phases, the ordering becomes trivial as we guarantee system correctness (a serial schedule) in either scheme. However, without using forward validation coupled with a No Sacrifice policy, it would be more costly to employ RWV; if a validating transaction is aborted it is expensive to undo the changes made during the write phase. This would also result in an increased number of conflicts due to any transactions that have accessed the same data having to be aborted or rerun. An advantage of this strategy of never aborting validating transactions (NAV) is that the resources utilized by a validating transaction are not wasted [19]. With the addition of a rerun policy we see further performance improvements from a RWV ordering (as presented in the results section).

Real-time centralized transactional databases need to handle transactions with timing constraints in the form of deadlines. Factors such as system contention have a direct impact on satisfying transactional deadlines; such factors occur during validation. Therefore we acknowledge that in the traditional OCC phase ordering, the validation step introduces a degree of non-determinism with regards to how long writes will take to become visible in the database (delaying entering write phase).

The validation phase is required to ensure system correctness with regards to transactions that are still executing rather than providing a direct benefit to the validating transaction itself. If the write phase occurs before the validation phase then we remove the non-deterministic timing constraints of the validation phase allowing the transaction to commit sooner. Consistency is still maintained in a virtual execution environment as the validation phase will detect transactions that are in conflict during rerun stages.

We also remove a degree of blocking present in the original read-validate-write ordering (RVW). Under RVW a transaction executing in the read phase will eventually have to be blocked to allow a transaction in the critical section to complete. Transactions partially through their read phase that do not conflict with the validating transaction that are allowed to continue execution may potentially enter a conflicted state. This arises if a value read by a transaction in the read phase is shared with the write set of a committing transaction. As a result, all concurrently running transactions must be blocked to allow the validating transaction to commit. Any newly arriving transactions will also be blocked from entering the read phase.

By employing a RWV ordering, we no longer have to block any transaction from progressing (we do not consider transactions waiting to enter the critical section as blocked). Having completed the write phase, a validating transaction only needs to validate against transactions that were active while the validating transaction was writing. These active transactions may have read data that has now been updated. Any newly arriving transactions (those arriving while a transaction is validating) cannot conflict with the validating transaction, as the data will have already been updated.

B. Protocol Description

A transaction that reaches the end of the read phase enters a pre-commit set (PCS). One member of the PCS may be chosen to enter the write phase by the scheduler. We employ an earliest deadline policy [3] to give priority to transactions that are closest to expiration.

Transactions that are either in the read phase or are members of PCS may be aborted and rerun if they are found to be in conflict with a validating transaction. We guarantee the validating transaction to commit and so we must rerun any other transactions that are in conflict. A transaction that is in its initial run will complete the read phase, regardless of being in conflict, and enter PCS. Allowing conflicted transactions to complete the read phase improves performance [5] by only accessing the persistent data store once per read operation. A transaction that is rerun will have a local copy of all the required data for it to attempt execution again.

We employ a forward oriented validation scheme which, during the validation phase of T_i, checks if there is an intersection between the write set $WS(T_i)$ with any read set $RS(T_j)$ for all running transactions. This includes transactions executing in the read phase and members of PCS. If an intersection (i.e., a conflict) is found then:

- If the conflicting transaction T_j is in the initial run, it is allowed to proceed with the read phase and is marked for rerun. T_j enters PCS upon completing the read phase but is not eligible to enter the write phase. At this point, T_j is updated with the values from other transactions it has conflicted with and will be rerun.
- If T_j is in rerun then it is aborted. At this point, $RS(T_j)$ is updated with $WS(T_i)$ so that it can be rerun again with the updated read set.

Arriving transactions may start the read phase at any time. We ensure correct execution as follows:

- If a transaction enters the read phase while another transaction is in the write phase there is the possibility of reading inconsistent data. This will be detected when the

transaction in the critical section finishes the write phase and enters the validation phase.

- If a transaction enters the read phase while another transaction is in the validation phase then any reads are made against the updated values from the persistent store (as validation occurs after write). Any transactions entering the read phase at this point do not need to be validated against the currently validating transaction.

In the event of a complete failure then, on restart, all transactions that have not committed begin again and read from the database directly (as if they were in the initial run). Any validation that was occurring before the failure is lost. However, the transaction that is validating has already written the updates to the database which are available on restart.

C. Pseudo-code

We present pseudo-code illustrating the validation phase for a transaction T_i. We use the following conventions:

- Active Transactions (AC) – This is the set of all currently running transactions. It includes transactions in the read phase and those waiting to enter the commit phase.
- Conflicted Set (CS) – CS(T_j) contains the updated read values from validating transactions that T_j has conflicted with. Each item (O_k) in CS(T_j) is cached until RS(T_j) can be updated. We cache these values rather than directly update the read set of T_j to make it clear that the writes would not be automatically updated if we chose to update RS(T_j) directly. RS(T_j) can be updated when T_j has finished the initial run or, if it is in rerun, when it is aborted. Upon updating, CS(T_j) is discarded.

We assume that a transaction executing in the read phase reads the required data and performs any required computation. Similarly, a transaction in the commit phase updates any values that were written to during its read phase. The scheduler handles rerunning identified transactions along with updating the read sets for conflicting transactions.

The pseudo-code for the validation phase is as follows:

```
for each Tj in AC do
  if ((WS(Ti) ∩ RS(Tj)) ≠ {}) then
     for each Ok in (WS(Ti) ∩ RS(Tj)) do
       update Ok in CS(Tj);
     end for
     if Tj in initial run then
       mark Tj for rerun;
     else
       update Tj with CS(Tj), rerun Tj;
     end if
  end if
end for
discard WS(Ti);
```

IV. SIMULATION AND RESULTS

We describe the simulation model which demonstrates our protocol. We provide a brief overview of the structure of the model and the parameters used. We then discuss the results by comparing the performance with the original protocol.

A. Simulation Environment

We produced a simulation model that matches closely the accepted designs seen in the literature [7, 8]. We have introduced a couple of modifications to this design to accommodate the rerun of transactions and the format of our later validation protocol. The model investigates different performance characteristics of our protocol versus a forward validation approach with virtual execution. We present a range of results highlighting the performance benefits of later validation using a virtual execution policy.

Our queuing model consists of a single-site database system operating with a shared-memory multiprocessor. We model two disks and two CPUs with a queue per disk and a shared queue for the CPUs. The parameters for our simulation can be seen in Table 1, and were taken from [7, 13, 15]. The transaction size remains the same for every transaction and we assume that the write set is a subset of the read set.

Pages in database	5000
Transaction size	12-page read set, 4-page write
disk access (read)	36 μs
disk access (write)	200 μs
CPU access	1.5 μs
disk access probability (1st run)	0.5
disk access probability (rerun)	0
Minimum slack factor	2
Maximum slack factor	8
Validation cost (per transaction)	0.5 μs

Table 1. Simulation Parameters

When the transaction performs a read, a 36μs cost is incurred to access the disk and a further 1.5μs for processing the page. A write costs 200μs with 36μs to read the page beforehand. When the transaction enters the commit phase, 200μs per write is incurred. We use disk access probability for a page being present inside the buffer. For rerun transactions this probability is zero, as we guarantee the page is present in memory. The validation cost is based on the number of transactions that have to be validated with a unit cost of 0.5μs. Deadline assignment [9] is controlled by the minimum and maximum slack factor parameters.

Each simulation was performed using the same parameters for 10 random number seeds. Each run consisted of 10000 transactions. To allow the system to stabilize, the results from the first few seconds were discarded. We present the mean values for the performance metrics analysed across experiments.

B. Simulation Results

Figs 1-6 show the average response time, throughput and number of late transactions when 50% and 75% of execution transactions are updating transactions. In each graph we present results for two protocols making use of a virtual execution policy. The first protocol (LV) is the one we in-

troduce in this paper using the RWV phases. The other (FV) is forward validation using the RVW phases.

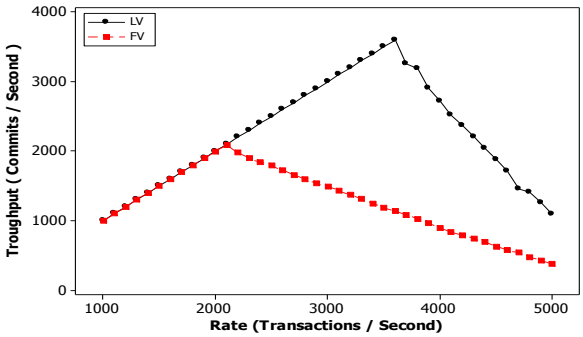

Fig. 1: Throughput with 50 % update transactions

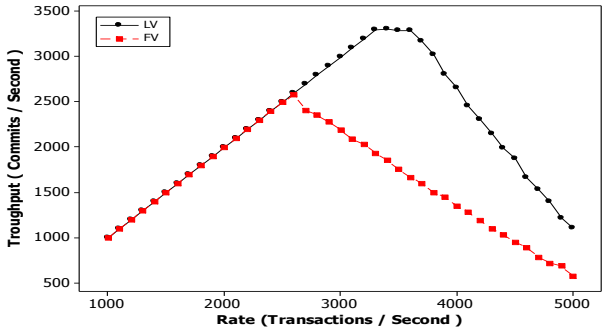

Fig. 2: Throughput with 75 % update transactions

Figs 1-2 show the throughput for an increasing rate of transactions. We measure throughput as the number of committed transactions, with the commit occurring at the end of the write phase for both phase orderings. All protocols share a common progression; a point is reached where contention is too high and the throughput starts to degrade. The number of transactions missing their deadline (figs 5,6) is also impacting the throughput as these transactions are aborted and will never commit. As the rate increases, the number of late transactions increases as the throughput falls. However, fig 2 shows a plateau around 3500 transactions/s which represents a bottleneck in the write and validation phases' critical section. This is not considered a problem as read-only transactions constitute the majority of a typical transactional traffic [14]. The graph still illustrates that later validation protocol sustains a higher level of throughput compared to the other approach.

Figs 3-4 show the average response time for an increasing rate of transactions. The response time is only for transactions that successfully commit. As the rate increases, transaction response time increases due to high contention. We see that, between 1500 and 5000 transactions per second, the LV approach has a lower response time than FV. This indicates that the cost of the validation phase does not affect the transaction's commit time in our approach

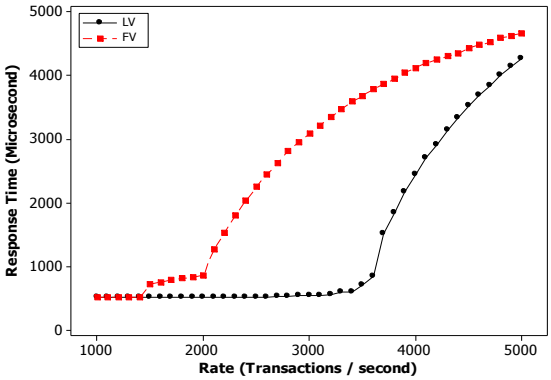

Fig. 3 Average response time with 50 % update transactions

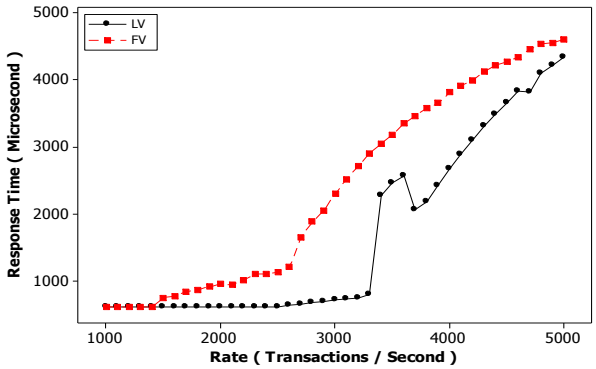

Fig. 4 Average response time with 75 % update transactions

After 5000 transactions, the average response time is similar for both protocols. The response time stabilizes around 4500 microseconds due to the deadline assignment; only transactions that have a sufficiently large deadline will be able to commit. Regardless of the benefits of our protocol, at this level of contention, transactions expire during the initial run in the read phase. The jump in fig 4 at a rate of ~3400 and then decline at ~3700 is explained by the plateau in fig 2. First the response times jump because executing transactions need to wait before they are able to enter the write and validation phases, and then response times decline due to the increase of the miss rate at that arrival rate.

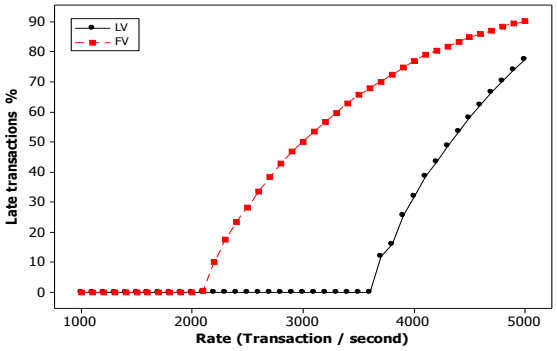

Fig. 5. Late transactions with 50 % update transactions

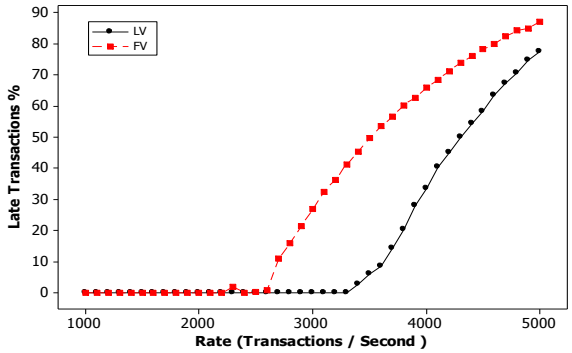

Fig. 6. Late transactions with 75 % update transactions

Figs 5-6 show the percentage of transactions that miss their deadline. For each protocol, as the rate increases, the percentage of missed deadline also increases. Each protocol, at its peak, has a high percentage (around 80%) of missed deadlines. With a high level of system contention, transactions experience longer delays in accessing the disk and the CPU. This results in transactions being more likely to miss the deadline during the initial run and never entering rerun.

V. CONCLUSIONS

In our earlier work [11] we realized that the utilization of virtual execution in OCC allows the write phase to be accomplished before the validation phase. This reversal does not inhibit correctness and our original idea was that it might bring performance benefits, particularly for real-time systems. This paper brings this idea to fruition by presenting a full description of our approach with validate and write phases reversed (read-write-validate).

We have also demonstrated the performance using an appropriate simulation (as used by earlier works in this area). We benchmarked our results against the original optimistic approach that utilizes a virtual execution model. We found that our approach significantly improves throughput and timeliness (transactions achieving deadlines) of the overall system when compared to the conventional approach.

Our approach of changing the phase order to read-write-validate, combined with virtual execution, requires significantly fewer accesses of the solid state disk, as more contentions can be resolved without touching the disk. This leads to better energy efficiency during operation of multiple applications, and therefore increases smartphone battery life.

In future work, we intend to explore other analytical models (e.g., [22][23]) to further investigate our simulation results. Additionally, we will create a real-world implementation and compare it with the simulated results.

REFERENCES

[1] Kung, H.T., Robinson, J.T.: On Optimistic Methods for Concurrency Control. ACM Transactions on Database Systems. 6, 213-226 (1981)

[2] Härder, T.: Observations on Optimistic Concurrency Control Schemes. Information Systems. 9, 111-120 (1984)

[3] Haritsa, J.R., Livny, M., Carey, M.J.: Earliest Deadline Scheduling for Real-Time Database Systems. In: proceedings of the 12th Real-Time System Symposium, pp. 232-242 (1991)

[4] Lee, J.: Concurrency Control Algorithms for Real-Time Database Systems. Ph.D Thesis, University of Virginia: United States (1994)

[5] Yu, P.S., Dias, D.M.: Analysis of Hybrid Concurrency Control Schemes for a High Data Contention Environment. IEEE Transactions on Software Engineering. 18 118-129 (1992)

[6] Franaszek, P.A., Robinson, J.T., Thomasian, A.: Access Invariance and Its Use in High Contention Environments. In: proceedings of the 6th International conference on Data Engineering, pp. 47-55 (1990)

[7] Lee, J.: Precise Serialization for Optimistic Concurrency Control, Data & Knowledge Engineering. 29, 163-178 (1999)

[8] Agrawal, R., Carey, M.J., Livny, M.: Concurrency Control Performance Modeling: Alternatives and Implications. ACM Transactions on Database Systems. 12, 609-654 (1987)

[9] Lee, J., Son, S.H., Using Dynamic Adjustment of Serialization Order for Real-Time Database Systems. In: proceedings of the Real-Time Systems Symposium, pp. 66-75 (1993)

[10] Bernstein, P.A., Hadzilacos, V., Goodman, N.: Concurrency Control and Recovery in Database Systems. Addison-Wesley (1987)

[11] Solaiman, K.., Morgan, G.: Later Validation/Earlier Write: Concurrency Control for Resource-Constrained Systems with Real-Time Properties. In: 30th International Symposium on Reliable Distributed Systems Workshop, pp. 9-12 (2011)

[12] J. Shanmugasundaram, A. Nithrakashyap, R. Sivasankaran, and K. Ramamritham. Efficient Concurrency Control for Broadcast Environments. In ACM SIGMOD, 1999.

[13] Qin Z., Wang Y., Liu D., Shao., Real-Time Flash Translation Layer for NAND Flash Memory Storage Systems, Real-Time and Embedded Technology and Applications Symposium (RTAS), 2012 IEEE 18th, PP. 35-44 (2012)

[14] Ferro D. G., Yabandeh M., A critique of snapshot isolation, EuroSys '12 Proceedings of the 7th ACM european conference on Computer Systems, pp(155-168,2012

[15] Stefan Manegold. Understanding, Modeling, and Improving Main-Memory Database Performance. PhD thesis, CWI Amsterdam, 2002.

[16] V. C. S. Lee, K. Wa Lam, and T.-W. Kuo, "Efficient validation of mobile transactions in wireless environments," *Journal of Systems and Software*, vol. 69, no. 1–2, pp. 183–193, Jan. 2004.

[17] S. Park and S. Jung, "An energy-efficient mobile transaction processing method using random back-off in wireless broadcast environments," *Journal of Systems and Software*, vol. 82, no. 12, pp. 2012–2022, Dec. 2009.

[18] Bosch, G.; Creus; Tuovinen, A.-P, "Feature Interaction Control on Smartphones," *Industrial Embedded Systems, 2007. SIES '07. International Symposium on*, pp.302-309, July 2007.

[19] Huang, J. and J.A. Stankovic, "Concurrency Control in Real-Time Database Systems: Optimistic Scheme vs. Two-Phase Locking," A Technical Report, COINS 90-66, University of Massachusetts, July 1990.

[20] Lei, X., Zhao, Y., Chen, S., Yuan, X.: Concurrency control in mobile distributed real-time database systems. Journal of Parallel and Distributed Computing, 69(10), pp. 866--876 (2009)

[21] S. Jung and K. Choi, "A concurrency control scheme for mobile transactions in broadcast disk environments," Data & Knowledge Engineering, vol. 68, no. 10, pp. 926–945, Oct. 2009.

[22] Shi Z., Beard C., Mitchell K., "Analytical models for understanding space, backoff and flow correlation in CSMA wireless networks", Wireless networks 1-17., 2012

[23] Shi Z., Beard C., Mitchell K., "Competition, cooperation and optimization in multi-hop CSMA networks with correlated traffic", Intl Jnl of next generation computing, 3 (3), 2012