Optimistic Concurrency Control for Energy Efficiency in the Wireless Environment

Kamal Solamain, Matthew Brook, Gary Ushaw, and Graham Morgan

School of Computing Science, Newcastle University, Newcastle-upon-Tyne, UK

Abstract. The ubiquity of smart portable devices has led to concurrency control for the mobile network becoming an area of growing concern. Conventional optimistic concurrency control techniques require retries of failed or disputed transactions, which place additional drain on the energy consumption of both the network and the smart device. We present a Distributed Later Validation Earlier Write Optimistic Concurrency Control (DLVEW) algorithm to efficiently handle transactions running on the server side without disturbing transactions running on clients. Our simulation shows an increase in throughput and reduction in both the response time and the number of missed deadlines of transactions. The corresponding reduction in contentious transactions needing to be restarted leads to a lower power cost for the network as a whole.

1 Introduction

Smartphone applications are placing greater demands on energy resources. Millions of smartphones and tablet devices are being used for more complex tasks, so the power consumption of the servers and network is increasing, and the battery recharge life of each phone or tablet is becoming shorter. An algorithm which reduces the energy cost of a transaction between a client (i.e. the phone) and the server will multiply to a significant energy saving across all devices in use. In particular, if the number of failed transactions due to contention can be reduced, thereby lowering the number of times a transaction must be repeated, then the overall power consumption will also be reduced.

Many applications require an asymmetrical channel whereby the frequency of read transactions requested by the client is significantly higher than the number of write transactions. Taking the example of a stock trading application; there are far more transactions involving a read-only checking of stock prices, compared to the number of transactions involving a sale or other event requiring an update transaction (i.e. users typically check far more share prices than they buy shares). A common implementation of this type of application involves the use of a broadcast disk protocol [1], whereby the database is repeatedly broadcast to the clients in its entirety. This approach means that there is no requirement for the client to send a read request to the server; the client simply waits for the requested piece of data to appear in the cycled transmission, and the server does not have to respond to individual client requests to send data. Clearly this

greatly reduces the amount of traffic on the network, and the amount of requests which the server must process. This type of approach is particularly useful when a relatively small database must be read by many clients.

Earlier studies on transaction processing in wireless environments were focused on read-only transactions [2] [3] [4]. Update transactions must also be considered. Optimistic Concurrency Control (OCC) is a well-understood solution for this type of situation [5]. However these protocols tend to involve heavy use of the network in both directions to request and validate read transactions, which renders the approach less applicable to mobile networks [2] due to limited uplink bandwidth and battery life. In [6] Lee proposed a variant of the OCC algorithm suitable for a broadcast environment known as forward and backward optimistic concurrency control (FBOCC). This algorithm performs partial backward validation [7] against committed transactions at the beginning of every broadcast cycle at mobile clients. It also performs forward validation [7] against concurrently running transactions at the server (including both server transactions and update mobile transactions).

In this paper we develop a DLVEW algorithm for broadcast disk which is more efficient at handling concurrently running transactions at the server without disturbing transactions running on the client. We achieve this by changing the ordering of the validate step at the server so that it takes place after the write step (conventionally it occurs before writing). In [8] we showed that this approach is applicable as optimistic concurrency control on resource-constrained devices such as smart-phones. We now extend this work to the broadcast disk model for mobile network applications that require significantly more read transactions than write transactions. Our results show that, with this technique, the number of client-server transactions which miss their deadline due to concurrency issues is reduced. The non-intuitive ordering of the validation phase, combined with the requirement of a rerun policy, improves efficiency while reducing the energy consumption of the network.

2 Background and Related Work

2.1 Broadcast Disk and Optimistic Concurrency Control

Many studies have proposed transmitting data over wireless networks using data broadcasting techniques [9] [10]. The broadcast disk protocol continuously broadcasts all data objects in the database. Clients view this broadcast as a disk, accessing required data as it is broadcast. The number of mobile devices does not affect their access time (as it is read-only). This approach makes conventional concurrency control techniques inapplicable [2]. E.g. using locking techniques could lead to swamping the server with lock requests. Similarly for timestamp based techniques, communication between clients and the server is needed for every read operation to keep track of both read and write timestamp; this can be unwieldy in broadcast environments. Conventional OCC [5] cannot be directly applied to mobile transaction processing because of the communications which consume the limited uplink bandwidth and battery power [6].

The optimistic concurrency control approach using a three-phased transaction execution consisting of read, validate and write (RVW) phases was described in [5]. During the read phase of a transaction, clients access data without restriction and make their own local copy of this data. If any writes are required, they are made to the client's local copy before the validation phase is entered. The validation phase ensures that any changes a client has made locally can be satisfied globally. Other executing transactions are considered to determine whether the write requests made locally can be satisfied without invalidating the overall read-write schedule. If the write requests are valid then the transaction moves onto the write phase and the local changes are committed to the persistent store at the server. Otherwise, the transaction must abort and restart.

Harder [7] proposed two schemes for the validation phase: Backward Oriented Optimistic Concurrency Control (BOCC) and Forward Oriented Optimistic Concurrency Control (FOCC). BOCC operates by comparing the read set of a validating transaction with the write sets of all currently executing transactions that have finished the read phase before the validating transaction. If a conflict is identified then the validating transaction must be aborted and restarted in its entirety. FOCC, on the other hand, is based on comparing the write set of the validating transaction with the read sets of all currently executing transactions that have yet to finish the read phase. When a conflict is found, FOCC provides a degree of flexibility in that a number of resolution policies are possible. It is this flexibility in resolution policy which has made FOCC the focus of further works [11]. However, aborting validating transactions is expensive because such transactions have used resources and completed execution. The Never Abort Validating transactions (NAV) strategy ensures that these resources will not be wasted by guaranteeing that the validating transaction commits [12]. However, a major drawback of FOCC is that concurrent transactions have to be blocked in their read phase while validating transactions are executing the validation and write phase in a critical section.

Virtual execution [13] involves pre-fetching data which will be required for a subsequent rerun of an aborted transaction. The approach enables transactions that are known to be in conflict to continue execution and complete the read phase, in order to pre-fetch the data that will be required for the subsequent rerun. Significant performance gains can be made when allowing the transaction to rerun using the pre-fetched data, due to access invariance. There is typically no disk I/O overhead required for the transaction during rerun. Significantly, battery power savings can be gained by deploying such a technique on mobile devices [14]. However, the issue of consistency arises for a transaction that operates using pre-fetched data as some of the pre-fetched data may have since been modified.

Lee has proposed a variant of the OCC algorithm called forward and backward optimistic concurrency control (FBOCC) [6]. FBOCC is a concurrency control algorithm suitable for mobile transactions in wireless broadcast environments. It consists of two validation stages. Partial backward validation is performed at clients between the write set of committed transactions at the

server and the read set of running transactions at the client at the beginning of every data cycle. This includes both read-only transactions and update transactions. Any conflicted transaction will be aborted. Successfully validated read-only mobile transactions will proceed to commit locally. Successfully validated mobile updated transactions are sent to the server to be validated globally. Forward validation is performed at the server between the write set of validating transactions and the read set of running transactions. This includes transactions generated and executed at the server, and update transactions which are sent for validation by clients to the server. Server conflicted transactions will be aborted. Conflicted update transactions will be aborted and will restart at the client. Update transactions must perform final partial backward validation at the server before starting forward validation. This final validation is needed in case of existing update transactions committed at the server since the last backward validation performed at the client. FBOCC is designed to minimize the use of the uplink channel in two ways: validation of read-only transactions locally at clients (these constitute the majority of mobile transactions); and early validating and aborting update transactions locally at clients, which makes update transactions more likely to pass the validation and write phases at the server.

2.2 Real-time requirements and phase ordering

The LVEW algorithm [8] [14] changes the order of the traditional RVW phases. The write phase now follows the read phase with validation occurring after the write is complete. In addition to the reordering of the phases the algorithm makes use of a rerun policy. Transactions are rolled back using in-memory data derived from retaining a buffer that records the writes of committing transactions and the reads of uncommitted transactions. Moving the validation phase ensures the nearest to expiring transaction (i.e. the closest to reaching its deadline) is afforded priority to commit. Also, there is no need to block concurrently running transactions during the write and validation phases. This promotes real-time efficiency and allows greater determinism. Writes become visible to transactions in the read phase earlier, affording more likelihood of reading up-to-date data.

In [14] we made two observations when considering real-time requirements. Firstly, transactions that enter rerun execute quicker than those in their initial run (as there is likely to be no disk access). Secondly, 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 require validation). Reruns can occur multiple times with minimal hindrance to transaction deadlines, as they execute with no disk latency. It would therefore be advantageous to keep transactions in rerun until we can deterministically say that, when a transaction leaves rerun, it will complete and meet its deadline, irrelevant of the delay imposed by the validation step. This would provide prioritization of rerun transactions without the concern for non-deterministic latency during the validation phase.

In [8] we applied this thinking to concurrent transactions on the shared resources of a smart device. The use of a virtual execution enabled OCC coupled

with the reordering of the validation and write phases allowed for an overall improvement in performance. When transactions are in a rerun state we can offset their validation until after the write phase of a transaction. The first benefit of this approach is that writes may become visible to transactions in the read phase earlier, affording more likelihood of reading up-to-date data. Secondly, overall blocking may be reduced, as in the original OCC protocols, transactions in the read phase will need to be blocked as a transaction commits changes to the database (to prevent out of date reads from the database). Such blocking would not be required, as out of date reads will be caught by the later validation step.

2.3 Energy Efficiency

An important objective of much of the work on concurrency control for mobile networks is to reduce the energy consumption, especially the battery life of the mobile devices. Much of the literature makes the point that conventional OCC techniques are less suited to mobile applications for this reason [6] [11] [15]. Accessing a conventional hard disk drive is expensive in terms of power usage, as the disk must attain read speed, and the appropriate data sector be found. Even solid state drives are significantly more expensive to access compared to local memory. Consequently reducing the number of times that a disk is accessed will reduce the energy consumed. Clearly a reduction in the frequency of transactions that must be rerun will reduce the amount of disk accesses which must be instigated, leading to a reduction in the energy usage. In general, it is better to perform execution at the fixed server, rather than at a mobile client [16]; this thinking can also be applied to concurrency resolution. Any energy saving achieved at the mobile device must be offset against the additional energy cost caused by any increase in communication over the network [17]. A protocol based on broadcast disk, which reduces the amount of validation messages going back and forth between clients and server, appears to meet this constraint.

2.4 Contribution

The background described in this section leads to the contribution made by this paper. Whereas previous work [18] [19] has described developments for the broadcast disk protocol which improve the client performance, we concentrate on the behaviour at the server. Our improvements are compatible with that existing work on client efficiency. We describe a new optimistic concurrency control algorithm suitable for a wireless broadcast environment in which the write phase occurs before the validation phase at the server. This approach has shown improvement in overall system throughput and the likelihood that transactions complete within their specified deadline.

We also deploy a rerun policy at the client. This reduces the access cost in the read phase when a transaction is aborted; this consequently reduces the battery usage in the mobile device. Additionally we show a reduction of the effect of the conflict increase rate on transaction results due to the increase of throughput rate at the server. Our work allows a server to resolve more contention, and

therefore increases mobile devices' performance, while reducing the energy cost due to retrying failed or conflicted transactions.

3 Protocol

We describe a read-write-validation approach to optimistic concurrency control for energy efficiency of transaction processing in a wireless broadcast disk environment. We also present pseudo-code to describe the algorithms execution. Our protocol builds on the FBOCC protocol proposed in [6] by performing a LVEW algorithm for validation at the server. The algorithm is performed in two stages.

3.1 Partial Backward Validation at Mobile Clients

All running transactions at clients (i.e. both the read-only transactions and update transactions) are validated at the beginning of every broadcast cycle by performing backward validation with the write set of committed transactions at the server. Conflicted transactions are marked for rerun, but continue execution until the end of the read phase in order to pre-fetch all read set data to memory [13]. When a conflicted transaction reaches the end of the read phase, we update the conflicted data objects in memory and rerun without accessing the persistent store. Optimistic concurrency control performs better if transactions are allowed to reach the end of their read phase before being aborted [20]. This is intuitive, as transactions that have been aborted early have not retrieved all the required data for the rerun phase. Rerun policy has a significant impact in saving battery power consumption in resource-constrained clients such as mobile devices [18] [14]. Not conflicted read-only transactions can proceed and commit locally at the client. Not conflicted update transactions will be sent to the server in order to be validated globally.

Pseudo-code for partial backward validation is presented as follows:

Algorithm 3.1: PARTIALBACKWARDVALIDATION(T_m)

if $(C_i \cap RS(T_m)) \neq 0$

then
- **for** each O_k in $(C_i \cap RS(T_m))$ **do** update O_k in $CS(T_m)$
- **if** T_m is in initial run
  - **then** mark T_m for rerun
  - **else**
    - update T_m with $CS(T_m)$
    - rerun T_m

else store the data in C_i

T_m is the transaction generated at the client. ControlInfo(C_i) os the set of data items which was updated. Conflicted Set (CS) given CS(T_m), this contains the updated values from C_i and T_m has been found to conflict with. Each item (O_k) in CS(T_m) is cached until RS(T_m) can be updated with these updated values. We choose to cache these values rather than directly update the read

set of T_m so as to make it clear that the calculations (writes) would not be automatically updated if we chose to update RS(T_m) directly. RS(T_m) can be updated when T_m has finished the initial run or, if it is in rerun, when it is aborted. Upon updating, CS(T_m) is discarded.

We assume that a transaction which is executing in the read phase reads the required data and performs any necessary computation. Similarly, a transaction which is in the commit phase will update any values that were written to during its read phase. The scheduler will handle rerunning transactions that have been marked for rerun, along with the process of updating the read sets for conflicting transactions.

3.2 LVEW and Final Validation at the Server

One of the transactions which are ready to commit will be chosen to enter the write phase by the scheduler. We employ an earliest deadline policy to give priority to transactions that are closest to deadline expiration. Once this transaction has completed the write phase, it performs forward validation against all concurrently running transactions at the server [8] [14]. This includes locally generated transactions and update transactions that have been received from clients for global validation. Any locally generated conflicted transactions will be marked for rerun. They will continue executing until the end of the read phase in the first run as described previously. Conflicted update mobile transactions will be aborted and rerun again at the client. When a validating transaction finishes the write and validation phases, the write set will be broadcast in the next broadcast cycle with the control information table. This information is used for partial backward validation at clients to keep mobile transactions consistent. However, update transactions have to perform final backward validation with any possibly committed transactions after the update transaction has finished partial validation at the client, and before starting LVEW validation at the server [18] [6]. The results of this validation (commit or abort) will also be included in the information table as acknowledgment to the mobile client for further actions.

3.3 Justifying Read-Write-Validate

This approach fundamentally changes the order of the traditional transactional phases as introduced in [5]. The write phase now follows the read phase with the validation phase now occurring after the write phase. Both the write and validation phases are collectively considered a single critical section, so only one transaction is allowed to be executing in either of these phases (adopted widely and described originally in [5]). We use a forward validation strategy in combination with a No Sacrifice policy [15] that guarantees a transaction entering the critical section will commit. This means that transactions which conflict with the validating transaction must be aborted. We choose to employ a rerun policy so that transactions in their initial run will continue to the end of the read phase before being rerun.

By combining the write and validation phases into a single critical section, the ordering of transactions becomes trivial as we can guarantee system correctness based on serializability criteria in either scheme. However, without using forward validation coupled with a No Sacrifice policy, it would be more costly to employ a RWV ordering. Without these mechanisms, if a validating transaction is aborted, it would be expensive to undo the changes made during the write. 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. With the addition of a rerun policy we see further performance improvements when combined with a RWV ordering.

Real-time transactional databases need to handle transactions with timing constraints in the form of deadlines. Upon arrival, a transaction must be processed in a timely fashion to ensure that the changes made during the read phase are successfully committed to the database before a deadline is reached. Factors such as system contention have a direct impact on satisfying transactional deadlines. Such factors occur during validation. Therefore 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 entry to the 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 is brought 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.

By altering the phase ordering we also remove a degree of blocking present in the original FOCC based on 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. If any of these read-phase transactions which do not conflict with the validating transaction are allowed continuing execution, they may potentially enter a conflicted state. This will arise if a future value is read by a transaction in the read phase that is shared with the write set of a committing transaction. There will be ambiguity as to which value would have been read (the one written by the committing transaction or the old value). In essence, this undetected conflicted transaction will read inconsistent data that the validating transaction will have modified during the write phase. 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 during this time to avoid further conflict. By employing a read-write-validate (RWV) ordering, we no longer have to block any transaction from progressing (we do not consider the transactions waiting to enter the critical section as being blocked). Having completed the write phase, a validating transaction will only need 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 transac-

tions (those arriving while a transaction is validating) cannot conflict with the validating transaction, as the data they read will have already been updated.

Pseudo-code for LVEW and final validation using the same notation explained in the section on partial backward validation is presented as follows:

Algorithm 3.2: SERVERVALIDATION(T_v)

comment: 1 Final backward validation:

for each T_i (i = 1...n) $\begin{cases} \textbf{if } (RS(T_v) \cap WS(T_i) \neq 0) \\ \quad \textbf{then return } \text{(fail)} \end{cases}$

comment: 2 Write:

Commit $WS(T_v)$ to database
$C_i = C_i \cup WS(T_v)$

comment: 3 Forward validation:

for each T_j (j = 1...n) $\begin{cases} \textbf{if } (WS(T_v) \cap RS(T_j) \neq 0) \\ \quad \textbf{then } \text{abort } T_j \end{cases}$

Our approach is orthogonal to the back-off method [18] and the OCC for broadcast disks scheme [19]. That is to say, both of these approaches can be combined with our work.

4 Simulation and Results

We describe the simulation model which we have used to demonstrate our protocol, providing a brief overview of the structure of the model and the parameters that were used. We then discuss the results by comparing the performance of our simulated model with the a simulation of the original protocol FBOCC [6].

4.1 Simulation Environment

We have developed a simulation model that is based on the model presented in [6] [18] [19]. We have increased the transaction arrival rate at the server by a factor of 100 to a figure representative of current applications. The model was also extended slightly in order to accommodate the rerun of transactions and the format of our LVEW validation protocol, for meaningful comparison. The model investigates different performance characteristics of our protocol versus FBOCC combined with virtual execution. We present a range of results which highlight the performance benefits of LVEW validation using a virtual execution policy. The simulation model consists of a server, a client, and the broadcast disk structure. Only one client was used in our simulation, to provide direct comparison to the existing work; the work is built upon broadcast disk implementations where the read transaction is carried out entirely on the client (so the number

Parameter	Value
Server	
Transaction length	8
Read operation probability	0.5
Disk access time	1000
Transaction arrival rate	1 per 20000 to 1 per 1667
Concurrency control protocol	OCC with LVEW
Priority scheduling	Earliest deadline first
Client	
Transaction length	4
Read operation probability	0.5
Fraction of read only transactions	0.75
Minimum slack factor	2 (uniformly distributed)
Maximum slack factor	8 (uniformly distributed)
Mean inter-operation delay	65536
Mean inter-transaction delay	131072

Table 1. Parameters used in the simulation experiments

of clients is irrelevant), and mobile update transactions are relatively rare. The server executes the server transactions based on conventional FV and LVEW algorithms. The deadline of transactions is calculated by the following formula:

Deadline = arrival Time + uniform (Minimum Slack factor, Maximum Slack factor) * execution time

Execution time is estimated using the values of transaction length, CPU time and disk access (mean inter-operation delay in mobile transaction). Table 1 shows the parameters which were used during the simulation experiments. The time unit is in bit-time, which is the time to transmit a single bit. For a broadcast bandwidth of 64 kbps, 1 M bit-time is equivalent to approximately 15s.

4.2 Simulation Results

Due to the real-time nature of the application domain, our experiments focus on measuring the miss rate percentage, which is the percentage of transactions missing their deadlines. Another performance metric is the throughput which is strongly connected to miss rate; throughput is the number of transactions committed per time unit. Figures 1-3 show the throughput, average response time and miss rate of server transactions. Figures 4-7 show the throughput and miss rate of clients transactions. In each graph we present the results of two protocols: DLVEW and FBOCC.

Figure 1 shows the throughput for an increasing rate of transactions. We define 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; of particular interest is the point that is reached in both sets of data where contention is too high and the throughput starts to degrade. The number of transactions which miss their deadline (fig 3) is

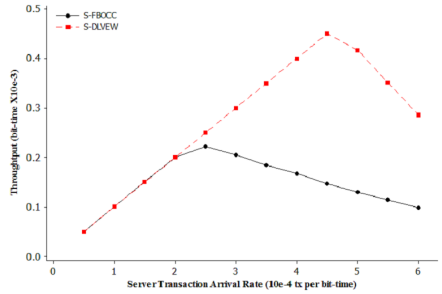

Fig. 1. Throughput at the server.

also impacting the throughput, as these transactions are aborted and will never commit. As the rate increases, the number of late transactions increases and so the throughput falls.

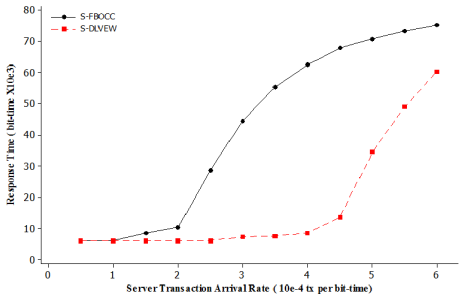

Fig. 2. Response time at the server.

Figure 2 shows the average response time for an increasing rate of transactions. The response time is only included for transactions that successfully commit. As the rate increases, the transaction response time increases due to high contention. We see that, between 1 and $6 * (10^{-4})$ transactions per bit-time, the LV approach has a lower response time than FV. This indicates that the cost of the validation phase does not affect the transactions commit time in our approach. The response time stabilizes after 80000 bit-time 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.

Figure 3 shows the percentage of transactions which miss their deadline. For each protocol, as the rate increases, the percentage of missed deadlines also increases. Between 2 and $6 * (10^{-4})$ transactions per bit-time, the LV approach has a lower miss rate than FV. With a high level of system contention, transac-

tions 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.

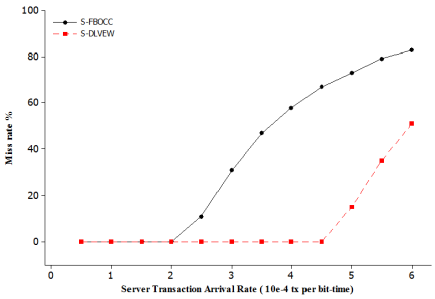

Fig. 3. Miss rate at the server.

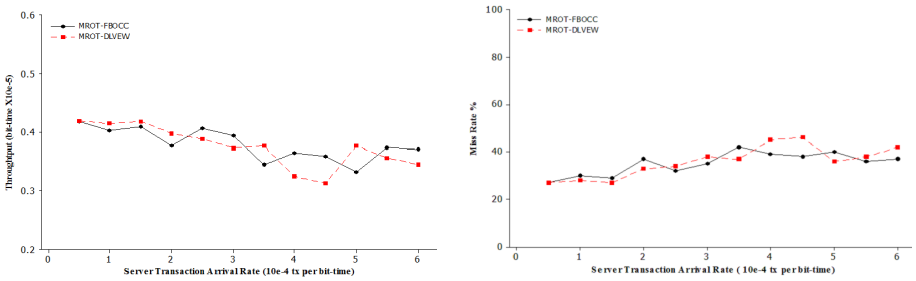

Fig. 4. Throughput and miss rate of read only transactions at clients

Figure 4 shows the miss rate and throughput of mobile read only transactions. The figures demonstrate that both protocols generate similar results. This result was expected because read only transactions execute and commit locally in the client.

Figure 5 shows the miss rate and throughput of update mobile transactions. Figure 5a illustrates that the throughput of both protocols is similar when contention at the server is low. The LV protocol demonstrates higher throughput whenever the server transaction arrival rate has increased. Figure 5b shows that the miss rate of the LV protocol is always lower than the miss rate of the FV protocol in all contentions, which is convenient for real-time mobile applications.

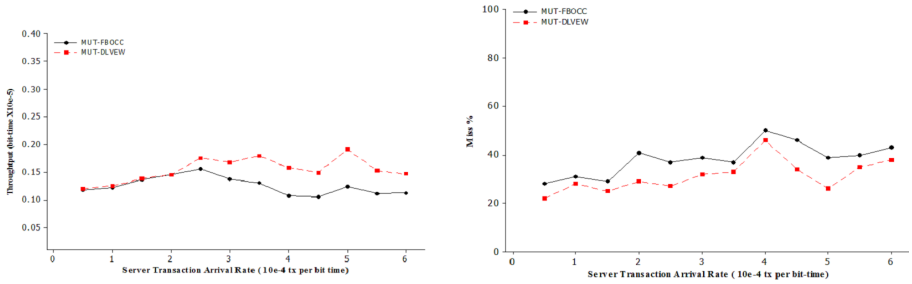

Fig. 5. Throughput and miss rate of update transactions at clients

5 Conclusions

In [8] we identified the possibility that, in combination with virtual execution, a performance improvement could be made by allowing the write phase to be accomplished before the validation phase . In [8] [14] we explored this idea in the context of multiple applications running concurrently on a resource-constrained device. We showed that, not only does this reversal maintain correctness, it also brings performance benefits. This is particularly evident for real-time systems. In this paper we deploy our approach to further develop forward backward optimistic concurrency control for mobile transactions in the wireless environment.

We have developed a simulation of this technique (using an appropriate simulation as used by earlier works in the area) in order to demonstrate the performance. We have then benchmarked the results from these tests against the original FBOCC approach combined with a virtual execution model. We have shown that our approach significantly improves both the throughput and the miss rate of the overall system when compared to the original technique. We have simulated 100x more frequent transaction arrivals than the previous works, to reflect the modern usage of the technology.

Our seemingly counter-intuitive idea of changing the phase order to read-write-validate, combined with virtual execution, requires significantly fewer accesses of the server data, and completely eliminates blocking transactions at the read phase. This leads to resolving more contention by a more able server, and therefore increases mobile devices performance, while reducing the energy cost due to retrying failed or conflicted transactions.

References

1. Acharya, S., Alonso, R., Franklin, M., Zdonik, S.: Broadcast disks: data management for asymmetric communication environments. In: ACM SIGMOD Record. Volume 24., ACM (1995) 199–210
2. Shanmugasundaram, J., Nithrakashyap, A., Sivasankaran, R., Ramamritham, K.: Efficient concurrency control for broadcast environments. ACM SIGMOD Record **28**(2) (1999) 85–96

3. Pitoura, E., Chrysanthis, P.K.: Scalable processing of read-only transactions in broadcast push. In: Distributed Computing Systems, 1999. Proceedings. 19th IEEE International Conference on, IEEE (1999) 432–439
4. Barbará, D.: Certification reports: supporting transactions in wireless systems. In: Distributed Computing Systems, 1997., Proceedings of the 17th International Conference on, IEEE (1997) 466–473
5. Kung, H.T., Robinson, J.T.: On optimistic methods for concurrency control. ACM Transactions on Database Systems (TODS) **6**(2) (1981) 213–226
6. Lee, V.C.S., Lam, K.W., Kuo, T.W.: Efficient validation of mobile transactions in wireless environments. Journal of Systems and Software **69**(1-2) (Jan 2004) 183–193
7. Härder, T.: Observations on optimistic concurrency control schemes. Information Systems **9**(2) (1984) 111–120
8. Solaiman, K., Brook, M., Ushaw, G., Morgan, G.: A read-write-validate approach to optimistic concurrency control for energy efficiency of resource-constrained systems. In: Proceedings of the 9th International Wireless Communications and Mobile Computing Conference, IEEE (2013)
9. Juran, J., Hurson, A., Vijaykrishnan, N., Kim, S.: Data organization and retrieval on parallel air channels: Performance and energy issues. Wireless Networks **10**(2) (2004) 183–195
10. Lee, V.C., Lam, K.W., Son, S.H., Chan, E.Y.: On transaction processing with partial validation and timestamp ordering in mobile broadcast environments. Computers, IEEE Transactions on **51**(10) (2002) 1196–1211
11. Lee, J.: Precise serialization for optimistic concurrency control. Data & knowledge engineering **29**(2) (1999) 163–178
12. Huang, J., Stankovic, J.: Concurrency control in real-time database systems: Optimistic scheme vs. two-phase locking. Univ. of Messachusetts, COINS Technical Report (1990) 90–66
13. Franaszek, P.A., Robinson, J.T., Thomasian, A.: Access invariance and its use in high contention environments. In: Data Engineering, 1990. Proceedings. Sixth International Conference on, IEEE (1990) 47–55
14. Solaiman, K., Morgan, G.: Later validation/earlier write: Concurrency control for resource-constrained systems with real-time properties. In: Reliable Distributed Systems Workshops (SRDSW), 2011 30th IEEE Symposium on, IEEE (2011) 9–12
15. Lee, J.: Concurrency control algorithms for real-time database systems. PhD thesis, Citeseer (1994)
16. Kumar Madria, S., Mohania, M., Bhowmick, S.S., Bhargava, B.: Mobile data and transaction management. Information Sciences **141**(3) (2002) 279–309
17. Miettinen, A.P., Nurminen, J.K.: Energy efficiency of mobile clients in cloud computing. In: Proceedings of the 2nd USENIX conference on Hot topics in cloud computing, USENIX Association (2010) 4–4
18. Park, S., Jung, S.: An energy-efficient mobile transaction processing method using random back-off in wireless broadcast environments. Journal of Systems and Software **82**(12) (2009) 2012–2022
19. Jung, S., Choi, K.: A concurrency control scheme for mobile transactions in broadcast disk environments. Data & Knowledge Engineering **68**(10) (2009) 926–945
20. Yu, P.S., Dias, D.M.: Analysis of hybrid concurrency control schemes for a high data contention environment. Software Engineering, IEEE Transactions on **18**(2) (1992) 118–129