



Master's thesis in
Information- and Communication Technology

Title:
**Data Management and Concurrency Control in
Broadcast based Asymmetric Environments**

Candidates:
Arild Finne
Erik Trædal

Supervisors:
Guohui Li, HUST
Ole Christoffer Granmo, AUC



Introduction

Tens of millions of users have personal handheld devices with several built-in network interfaces, and the numbers are only increasing. This enables many new environments where many users want to access the same data set.

Especially in high density areas, the number of users can be so high that the traditional pull-based method (request and get a response) is not usable. The server would be swamped with requests, and the bandwidth in wireless environments is not big enough.

The solution is to continuously broadcast the data set in cycles, such that the clients can read from the air. This way the server do not get any read requests and the bandwidth is saved. The upload bandwidth is also saved.

The environment can be simplified to transactions committed on a database. Each transaction consists of one or more operations, and each either reads or writes one database item.

The handheld devices are battery powered and have limited processing power. Using the network interface is very power consuming, so the usage should be kept to a minimum.

Data Management

The standard approach for data broadcast is to use a flat pattern that is broadcasted in cycles. Given the data *a, b, c, d, e* it will be broadcasted like this: "*abcdeabcde...*". A client can then tune in a read the data it needs.

The *transaction time* is the time it takes to complete a transaction from beginning to end. If first operation is to read *b* and next is to read *a*, it takes almost a whole cycle to wait for *a* to be broadcasted again.

Techniques such as caching and prefetching aims at reducing this overhead. But not all client devices are capable of these techniques because of limited processor, memory and battery power. Also, the techniques are not efficient on all application types.

Concurrency Control

The concurrency control maintains the consistency of the data, and makes sure the clients read consistent data. Illustration 1 shows an example with transaction T_1 and T_2 where T_1 reads *a*, then T_2 commits write to *a* and *b*. Finally T_1 reads *b*, which is inconsistent with the value T_1 read of *a*. The task of concurrency control is to avoid such inconsistency.

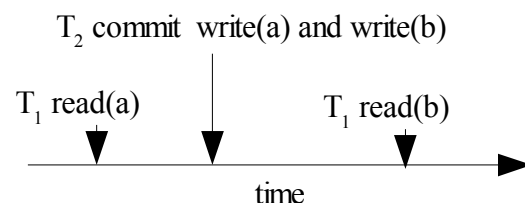


Illustration 1: Reading inconsistent data

Normal approach is to use lock-based protocols. They lock each item they use, so no other process can update this item. So in Illustration 1 T_1 would lock *a* when it was read. Then T_2 would not be allowed to commit its write operations, and T_1 could read a consistent value of *b*.

Optimistic concurrency control (OCC) is another concurrency control approach more adapted to the broadcast environments. It allows all operations until the time of validation. The validation can be done several times, but at a minimum it must be done before the transaction is accepted. In Illustration 1, T_1 would be validated at the end and found inconsistent, and be rejected. Then T_1 would either abort or restart.

Partial Restart

Instead of restarting the whole transaction, we suggest to only restart a part of the transaction.

Because the operations in a transaction is decided out from the value of the preceding operations, it is not enough to restart only the conflicting operation. The value of the conflicting operation has changed, so the following operations are probably no longer executed on the correct data items.

But the operations before the conflicting transaction are still the same, and as long as they are valid it is not necessary to read them again.

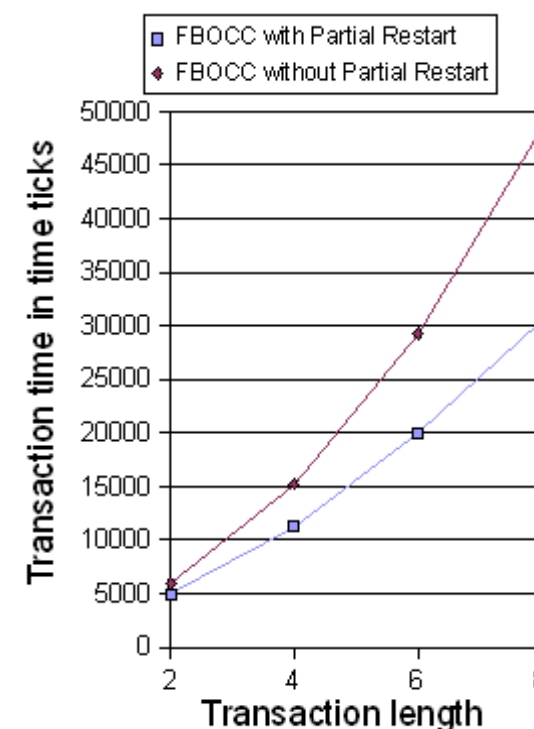


Illustration 2: Transaction time of FBOCC with Partial Restarts compared to normal FBOCC

We define *partial restart* as the following approach:

Instead of restarting the whole transaction when a conflict is discovered, it is only necessary to restart from the operation that caused the conflict. If several conflicts arise, the transaction should restart from the oldest operation (earliest in the transaction order) that causes a conflict.

The challenge of assuring the correctness to the non-restarted operations is dependent of the protocol the approach is implemented in.

None of the reviewed protocols use *partial restart*, but several of them can implement it fairly easy. We have implemented *partial restart* in FBOCC, and the simulations results are promising. Illustration 2 shows the average *transaction time* for FBOCC with and without *partial restart*. For 6 operations per transaction (transaction length = 6) the *transaction time* is 30% shorter with *partial restart*. (The test ran with demanding environment settings).

Conclusion

Data management and concurrency control consists of a lot more then what presented here. In our report we have surveyed the whole topic as well as presented a characterization framework. We have also made a simulation platform in CSIM, implemented several state-of-the-art protocols including our contribution *partial restart*.

The *partial restart* can optimize any concurrency control protocol which has overview of the conflicting operations. The decreased *transaction time* can have a major improvement on real-time transaction which have a deadline to meet. And the reduced number of operations will reduce the battery consumption in battery powered devices because of less network usage.