

Broadcasting Consistent Data in Mobile Environments*

Bryan Hin Cheung Poon, Kwok-Wa Lam, Victor C. S. Lee
Department of Computer Science, City University of Hong Kong
bryanp@cs.cityu.edu.hk, csleric@cityu.edu.hk, csvlee@cityu.edu.hk

1. Introduction

Many studies on transaction processing in mobile environments have an implicit assumption that the server is able to broadcast consistent data to the mobile clients. However, this assumption may not be valid unless there is a special algorithm to handle broadcasting data in a consistent and timely manner. In this paper, we formulate broadcasting data at the server as a broadcast transaction which reads the entire database in a consistent way in the broadcasting process. This issue is not trivial as the broadcast transaction will create high interference to normal transactions at the server.

2. Read-Write Set Test

Some related algorithms on reading the entire database could be candidate to solve the problem. However, those algorithms (such as Color Test [1] and Shape Test [2]) are inadequate in handling transactions in a serializable way. Both algorithms were focused on concurrency control between the global-read transaction and update transactions and ignored the existence of read-only transactions. Unfortunately, those read-only transactions may impair the consistency of the broadcasting data. Therefore we devise two new algorithms called Read-Write Set Test (RWST) and Relaxed Read-Write Set Test (RRWST). Both algorithms can cause least interference to normal transactions when there is a broadcast transaction in execution and prevent any non-serializable schedule.

The RWST is a separate algorithm which guarantees the serializability between the broadcast transaction and other normal transactions including read-only ones. There are four different sets of data during the execution of the broadcast transaction. They are:

- **NRS**: stands for Not Read Set, this set of data objects are not read by the broadcast transaction.
- **ARS**: stands for Already Read Set, this set of data objects are already read by the broadcast transaction.

* The work described in this paper was fully supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China [Project No. CityU 1152/01E].

- **NUS**: this set of data objects is a sub-set of ARS that are written by update transactions at least once.
- **URS**: this set of data objects is a sub-set of NRS that are already read by the update transaction (which also updated the data objects in ARS) or read by read-only transactions.

With these sets of data, the RWST can guarantee that the broadcast transaction can read the entire database in a consistent way by resolving data conflicts with other normal transactions. The data conflicts are resolved by the following rules:

1. $(WS(U) \cap NRS \neq 0) \wedge (WS(U) \cap ARS \neq 0)$
2. $(WS(U) \cap NRS \neq 0) \wedge (RS(U) \cap NUS \neq 0)$
3. $(WS(U) \cap URS \neq 0)$

If an update transaction fails one of the above rules, it will be aborted immediately. Another advantage of the RWST is the easy integration with the existing concurrency control protocols.

3. Conclusion

We have performed a series of simulation experiments to evaluate the performance of the two new algorithms. Preliminary results confirmed our view that conventional concurrency control algorithms are unworkable to process the broadcast transaction as a normal transaction and that using a separate algorithm to process the broadcast transaction is a better approach. The performance of both algorithms is comparable to the Shade Test [2]. However, we found that the RRWST does not outperform the RWST although the RRWST is based a more relaxed notion of consistency.

4. Reference

- [1] Calton Pu, "On-the-fly, Incremental, Consistent Reading of Entire Databases", Proceedings of VLDB85, Stockholm, pp. 369-375.
- [2] Paul Ammann, Sushil Jajodia, and Padamaja Mavuluri, "On-the-fly Reading of Entire Databases", IEEE Transactions on Knowledge and Data Engineering, Vol. 7, No. 5, October 1995.