

# A JAVA SYSTEM THAT COMBINES OPTIMISM AND PESSIMISM IN UPDATING REPLICAS

Claudine Innis, University of Technology, Jamaica, W.I., cinnis@utech.edu.jm  
Joel M. Crichlow, Rowan University, Glassboro, NJ, crichlow@rowan.edu  
Michael Hosein, University of the West Indies, Trinidad, W.I., mhosein@tstt.net.tt  
Steve Hartley, Rowan University, Glassboro, NJ, hartley@elvis.rowan.edu

## ABSTRACT

In a distributed system some or all of the data items are replicated and stored at separate sites. This increases the availability of these items and it is then possible to complete transactions faster than in a single site system. However, the concurrent processing of transactions at separate sites can generate inconsistencies in the stored information. Several replica control techniques have been proposed to deal with the processing of transactions. Some of these techniques include certain consistency checks before processing the transaction, i.e. they are pessimistic in nature. Other techniques process the transaction and do the consistency checking afterwards, i.e. they are optimistic.

The system discussed here is an attempt to show how the increased processing capabilities provided by (i) multi-threaded object-oriented programming, and by (ii) powerful computing nodes in the distributed system can be used to combine optimism and pessimism into one system. The system allows a transaction to be processed immediately (optimistically) at any individual replica as long as the transaction satisfies a cost bound. All transactions are also processed in a concurrent pessimistic manner. Only the changes made to the replicas via pessimistic processing are made permanent.

Keywords: Distributed system, replica management, concurrency control, availability. Optimistic processing, pessimistic processing.

## INTRODUCTION

The system under study contains a collection of computing nodes connected by a communications network. The information that transactions access can have any level of replication across the nodes. Transactions can originate at any node and the transaction processing system attempts to treat all transactions in a uniform manner through cooperation among the nodes.

One of the main reasons for replicating the data in distributed systems is to increase availability. However, this replication increases the need for effective control measures to preserve some level of mutual consistency. Several replica control techniques have been proposed to deal with this issue and these techniques are described to

different levels of detail in many articles and presentations e.g. Bernstein et al [1], Birman [2], Crichlow [4], Jajodia & Mutchler [7], Krishnakumar & Bernstein [8], Wolfson et al [10].

The techniques vary in a number of ways including the number of the replicas that must participate before a change can be made to a replica, the nature of the communication among the replicas, and if a replica can be changed before the others how is the change propagated. A key contributor to the choice of specific procedures is the nature of the application. For example some applications can tolerate mutually inconsistent replicas for longer periods than others. The twin objective is process the transaction correctly as quickly as possible and reflect this at all the replicas so that no harm is done.

Crichlow [3] proposed a scheme combining a simple pessimistic technique with a simple optimistic mechanism. It was argued that increasingly available parallelism (and concurrency) in execution afforded by multiprocessor (and other high-speed) nodes could be effectively employed both to preserve integrity and to increase availability in transaction processing. Availability is considered here as the fraction of all the transactions that are submitted to the system that complete successfully within an acceptable timeframe.

The main objectives in the design are to:

- Provide a high level of availability at a known penalty to the application,
- Preserve data integrity, and
- Build a system that is conceptually simple.

The concept of a known penalty was adopted by Lynch et al [9] in their SHARD system; in which a transaction can be processed immediately at its local node. Subsequently, information about that transaction will be reliably broadcast to all the other nodes. Their system has a fully replicated database. Transactions are totally ordered by a globally unique timestamp. Nodes use this ordering to merge information from different transactions. Yu and Vahdat [11] have more recently proposed a 'continuous consistency model' in which applications can specify a maximum deviation from strong consistency on a per-replica basis.

It is our view that this use of a cost bound is a sensible and effective approach to managing changes to replicas. Our basic argument for the use of a cost bound is that each design decision carries a trade-off. You gain something and you lose something. If you feel that for a particular approach the gain far outweighs the loss or that the loss can be monitored and controlled then it is recommended that that approach be taken. In the distributed system with replicas highest availability can be achieved if each replica can process requests individually. Therefore, we let them do that, but we want to know at what cost?

## THE DESIGN

The system accommodates any level of replication. Transactions can be processed at any node where a replica exists. Transactions can be processed immediately at any available replica if the transaction satisfies a cost bound criterion. This processing is classified as optimistic since processing is done before ensuring that mutual consistency of replicas is preserved. Transactions processed in this way do not change the state of the replica.

All transactions are also run pessimistically using timestamping for one-copy serializability. These results are committed to the replicas. Therefore, a transaction can be processed more than once depending on the current conditions in the system. Since the optimistic processing is controlled, the results from such processing can be published to the users. However, the owners of the application will have to address the cases where a published response is inconsistent with a committed result that followed the pessimistic run.

During this phase of our work we are considering resource allocation (or counting type) systems. Therefore, transactions are of the type that either request resources (debit) from or return (credit) resources to a database. The cost bound  $C$  is in the same units as the resource, e.g., if one is considering the allocation of seats on a flight, then  $C$  is some number of seats.  $C$  may be static or dynamic. A dynamic  $C$  responds to changes in the application.

$C$  is set initially using an application specific value that is considered bearable. For example, in airline reservation it can be related to the number of allowable over-booked seats.

$C$  is distributed across the  $n$  replicas such that for each replica  $i$ , that replica's cost bound is

$$w_i C, \text{ where } \sum_{i=1}^n w_i = 1 \quad (i)$$

and  $w_i$  is some weighting factor.

The mechanism for choosing  $w_i$  may differ with the application. In some cases a simple fractional weight can be used. However, other approaches employing optimization techniques or knowledge-based and

inference methods could be applicable. This is an area for further research.

Each transaction, on initiation, is associated with a front-end or client which is viewed as the owner of that transaction. Each transaction is expressed as a parent and child transactions. The parent remains at the owner while the children are sent to all the replicas. Child transactions join a hold-back queue at the replicas. Transactions, while on the hold-back queue, are processed optimistically (i.e. temporarily) on a FCFS basis. However, concurrently, transactions are globally ordered, removed from the hold-back queue and placed on a stable queue from where they are processed pessimistically (i.e. permanently).

In other words, transactions may receive advanced (temporary) processing while on the hold-back queue. Every replica is allowed to perform temporary processing independently on its hold-back transactions. The permanent (pessimistic) transaction processing is applied to the transactions that are selected from the globally ordered stable queue. Permanent processing has higher priority than temporary processing.

Responses from processed transactions are multicast to the parent and all the siblings. When the parent receives the response from a temporary run it can publish the result. When the parent receives the response from a permanent run it can publish the result only if it differs from a previous temporary result. When a sibling hears that a transaction on its hold-back queue has been completed temporarily elsewhere it halts its own temporary run (or bypasses that child) and moves on to the next. Exchange of messages on permanent processing facilitates consistent permanent state changes to replicas.

For example, let us consider a flight reservation system in which users are allowed to book and cancel seats on a flight. Assume that the number of seats available initially is 300 and that there is an allowable overbooking figure of 60 seats. The initial cost bound  $C$  can then be set to 360. Let there be 3 replicas and distribute  $C$  equally among the replicas, i.e.  $w$  in equation (i) is  $1/3$ . Therefore the initial cost bound at each replica is 120.

We propose a dynamic  $C$  value which changes in two ways: one in response to temporary processing and the other in response to permanent processing. Each replica decrements/increments its cost bound whenever it processes an allocate/de-allocate resource transaction temporarily. In the reservation example a booking request corresponds to an allocate while a cancellation of booking corresponds to de-allocate.

Permanent processing uses the following formula to generate a new cost bound:

$$C_{new} = \left(1 - \frac{RA}{N}\right) * C_{init} \quad (ii)$$

where:

$C_{new}$  is new cost bound;

$C_{init}$  is cost bound initially set (i.e. 360 in our example);  
RA is no. of resources allocated via requests;  
N is no. of resources initially available (i.e. 300 in our example).

After a new cost bound has been obtained in this way (i.e. after permanent processing), equation (i) is used to distribute the new cost bound among the replicas.

## TESTING

The viability of the design was demonstrated via a prototype coded in C. It allowed booking and cancellation of seats on a flight. It was tested on a platform of 5 networked Sun SPARC workstations in a UNIX environment. The nodes communicated by using Sun's RPC (Francis & Crichlow [5]). Fault tolerance was incorporated later in the C program modules in order to investigate the behavior when nodes fail (Hosein and Crichlow [6]).

In order to achieve increased portability, the prototype system is now coded in Java. Java clients accept transactions and interact with Java servers for transaction processing. The servers maintain a fully replicated database and communicate with each other via Java-RMI. Clients interact with a home server also via Java-RMI. In addition, a client and its home server can run on the same computer. The system can run wherever there is a Java Virtual Machine. It has been tested on a Windows NT platform, and it has been ported successfully to a LAN that runs Linux.

The Windows test-bed consisted of four uni-processor computers each running NT Workstation. One computer was used to generate transactions the other three computers managed the replicas. A GUI (see figure 1) was created to facilitate user entry of a number of variables:

*The Generation Rate:* The number of transactions that are generated per second.

*The Minimum and Maximum Change:* Minimum and maximum resource request that a transaction can make.

*Number of Transactions:* The number of transactions to be generated in this test.

*Dec. Ratio and Inc. Ratio:* These are set to specify the proportion of the transactions, which should be Decrements (debits) and Increments (credits) respectively.

*The Server Names* is a list, which contains the names of all the servers in the system to which transactions will be sent.

*The Distribution Ratios:* The proportion of transactions for each server.

Results showed (see figures 2 & 3) that we were able to achieve processing times on temporaries up to 30 times faster than for permanent processing, with the cost-

bound (CB) ranging from 100% to 116% of the resource. However some choices of parameters can generate relatively high undo percentage (see figure 4).

## CONCLUSION

A system has been designed that facilitates the speeding up or slowing down of the processing of transactions in a replicated distributed system. A basic test-bed is in place. The viability of the system has been established. The open-ended nature of the test-bed will facilitate the investigation of several additional features. An immediate interest is an alternative cost-bound scheme.

## REFERENCES

1. Bernstein, P.A., Hadzilacos, V. and Goodman, N., 1987. *Concurrency Control and Recovery in Database Systems*, Addison-Wesley Pub. Co., Reading, Ma.
2. Birman, K.P., 1996. *Building Secure and Reliable Network Applications*, Manning Pub. Co.
3. Crichlow, J.M., 1994. Combining optimism and pessimism to produce high availability in distributed transaction processing, *ACM SIGOPS Operating Systems Review*, 28, 3(July), 43-64.
4. Crichlow, J.M., 2000. *The Essence of Distributed Systems*, Prentice Hall/Pearson Education, U.K.
5. Francis, M.F. and Crichlow, J.M., 1995. A mechanism for combining optimism and pessimism in distributed processing, *Proceedings of the IASTED/ISMM International Conference on Intelligent Information Management Systems*, Washington, D.C. (June), 103-106.
6. Hosein, M. and Crichlow, J.M., 1998. Fault-tolerant Optimistic Concurrency control in a Distributed System, *Proceedings of the IASTED International Conference on Software Engineering*, Las Vegas, October 28-31, 319-322.
7. Jajodia, S. and Mutchler, D., 1990. Dynamic voting algorithms for maintaining the consistency of a replicated database, *ACM Transactions on Database Systems*, 15, 2(Jun), 230-280.
8. Krishnakumar, N & Bernstein, A.J., 1991. Bounded Ignorance in Replicated Systems, *Tenth ACM Symp. On Principles of Database Systems*.
9. Lynch, N.A., Blaustein, B.T. and Siegel, M., 1986. Correctness conditions for highly available replicated databases, *Proceedings of the fifth annual ACM Symposium on Principles of Distributed Computing*, Aug., 11-28.
10. Wolfson, Ouri; Jajodia, Sushil and Huang, Yixiu, 1997. An adaptive data replication algorithm. *ACM Transactions on Database Systems*, 22(2), 255-314.
11. Yu, H. & Vahdat, A., 2001. The Costs and Limits of Availability for Replicated Services, *Proceedings of the 18<sup>th</sup> ACM Symposium on Operating systems Principles*, Alberta, Canada, October 21-24, 29-42.

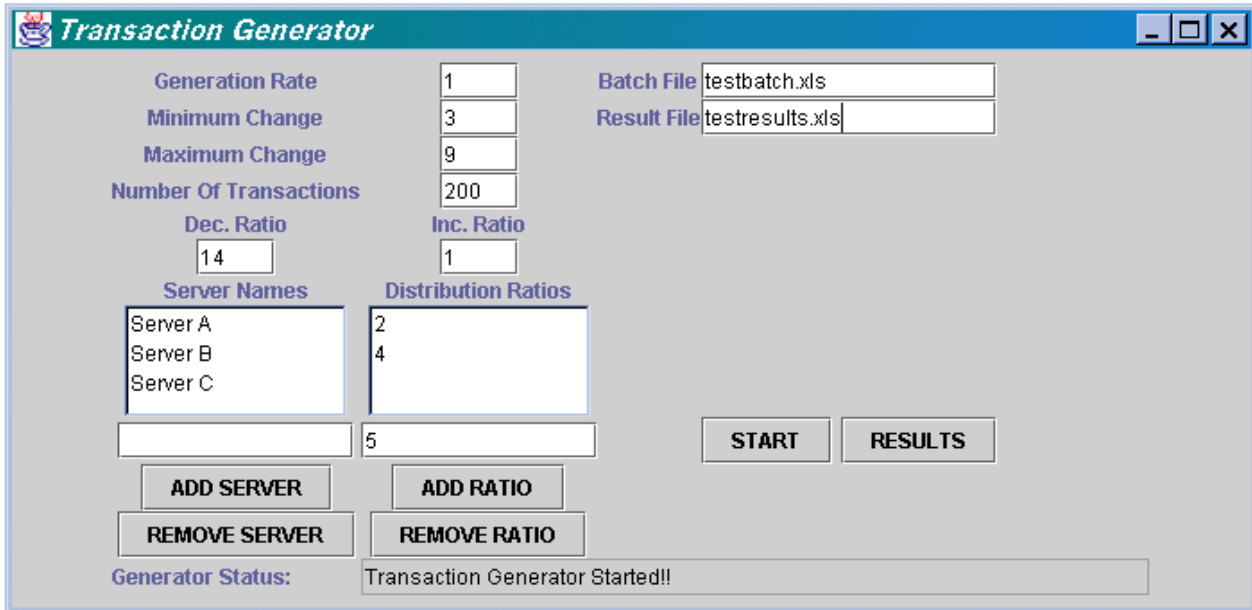
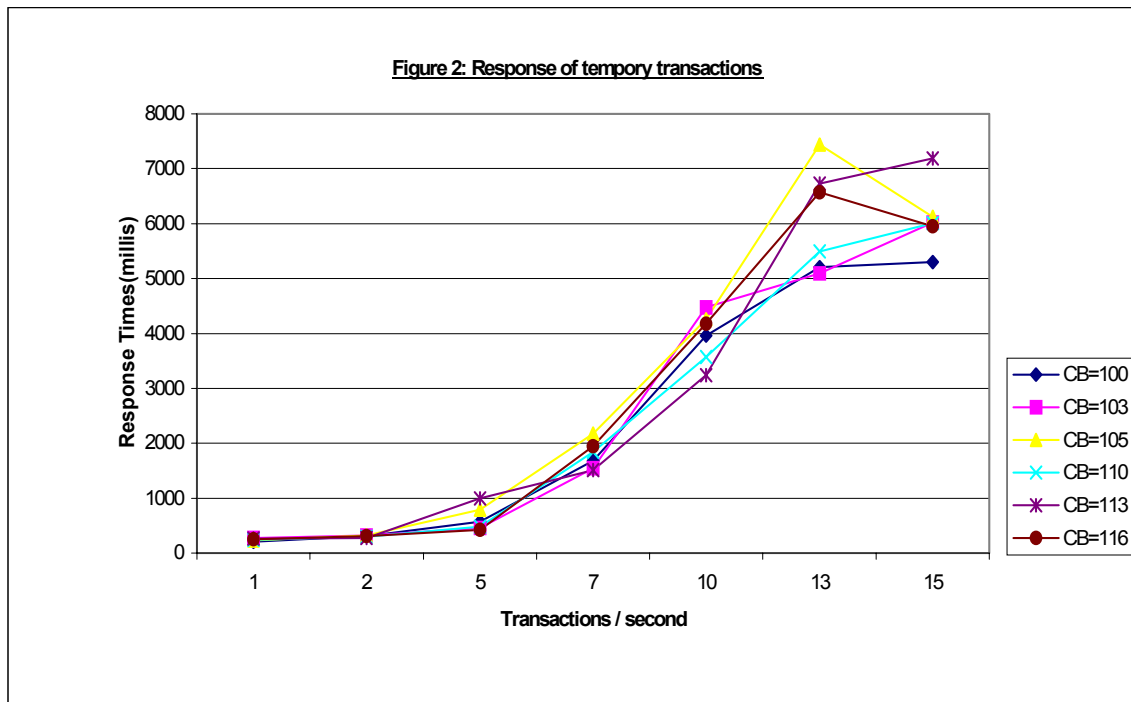
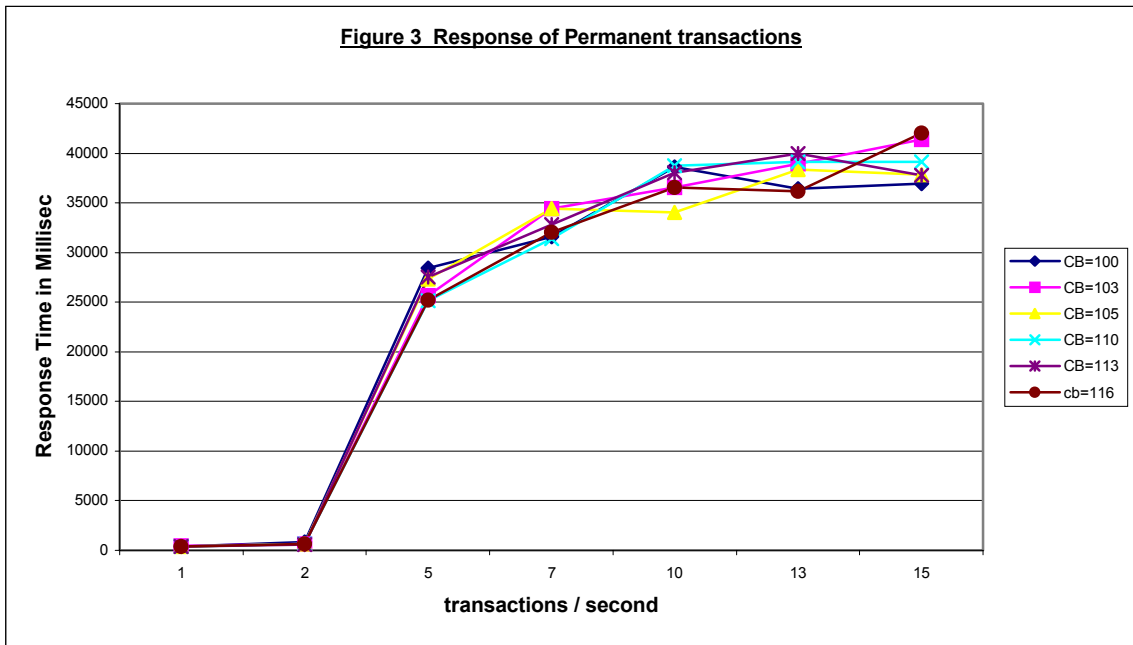


Figure 1.



**Figure 3 Response of Permanent transactions**



**Figure 4 Temporaries undone as fraction of all temps completed**

