

## THE COPAR SERVICE: COMBINING OPTIMISM AND PESSIMISM IN ACCESSING REPLICAS

Joel M. Crichlow, Rowan University, Glassboro, NJ 08028, crichlow@rowan.edu  
Stephen Hartley, Rowan University, Glassboro, NJ, hartley@elvis.rowan.edu  
Michael Hosein, University of the West Indies, Trinidad, mhosein@tstt.net.tt  
Claudine Innis, University of Technology, Jamaica, cinnis@utech.edu.jm

### ABSTRACT

In order to increase availability in a distributed system some or all of the data items are replicated and stored at separate sites. This is an issue of key concern especially since there is such a proliferation of wireless technologies and mobile users. 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.

We have built a distributed service that manages updates to widely deployed counter-like replicas. The service is built on our distributed concurrency control scheme which combines optimism and pessimism in the processing of transactions. The service 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 to ensure mutual consistency.

Information system management, distributed system, replica management, transaction processing.

### 1. Introduction

The COPAR service runs on a collection of computing nodes connected by a communications network. We have had test runs on private LANs as well as over the Internet. The data 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. Replication will become increasingly more useful in the face of wireless technology and roaming users. 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 [8], Krishnakumar & Bernstein [9], Wolfson et al [11].

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. Every transaction submitted to the system enters concurrently a global pessimistic two-phase commit sequence and an optimistic individual replica sequence. The optimistic sequence is moderated by a cost bound, which captures the extent of inconsistency the system will tolerate. The pessimistic sequence serves to validate the processing and commit the changes to the replicas or undo an optimistic run if it generated an inconsistency. Using this scheme we built a service that can provide highly available access to counter-like replicas widely deployed over a network. In testing our system over the Internet the optimistic run achieved 10 times, 100 times and even 1000 times faster turnaround than the pessimistic run. The results are discussed in a later section.

The main objectives in the design were to:

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

The concept of a known penalty was adopted by Lynch et al [10] in their SHARD system in which a transaction was processed immediately at its local node. Subsequently, information about that transaction was reliably broadcast to all the other nodes. Their system had a fully replicated database. Transactions were totally ordered by a globally unique timestamp. Nodes used this ordering to merge information from different transactions. Yu and Vahdat [12] have more recently proposed a ‘continuous consistency model’ in which applications can specify a maximum deviation from strong consistency on a per-replica basis.

Strong consistency ensures that concurrent updates will not conflict and is usually achievable by applying a pessimistic protocol. In the ‘continuous consistency model’ system consistency at each replica can be specified in three metrics. With these metrics Yu and Vahdat [12] have shown that upper bounds on availability can be calculated. The metrics used are *Numerical Error*, *Order Error* and *Staleness*.

Numerical error is the maximum weight of writes not seen by a replica; order error is the maximum weight of tentative writes (not committed) at the replica; and staleness is the maximum amount of time before a replica is guaranteed to observe a write accepted by a remote replica. Our cost bound is a numerical error metric.

We give the application the privilege to set dynamically the maximum numerical error it can accommodate. This cost bound then determines the extent to which a replica can act independently. Availability improves when replicas can act independently in processing transactions.

This is an ongoing project which follows work published earlier (see Francis & Crichlow [5], Hosein & Crichlow [6], and Innis et al [7]). However we have made key upgrades to the system and obtained additional interesting results which we feel constitute an important contribution to this field of study. First of all we spotted and corrected an integer division error in the Java program used in [7]. Due to this error the number of optimistic transactions that were undone were slightly more than necessary, but the pattern of results remains the same. The upgrades are

- The servers were all on a controlled LAN but they are now distributed over 2 New Jersey universities (and at the time of writing clearance is being made for a server in the island of Trinidad, in the Caribbean).
- More feedback information is now used to distribute the dynamically changing cost bound.

The details follow.

## 2. The Cost Bound

During this phase of our work we are considering resources that can be counted, e.g. available seats on a flight. All the resources are of the same type, therefore a request for  $r$  resources can be satisfied if the reachable pool is controlled by a cost bound of  $C$  resources.

$$r \leq C. \quad (i)$$

We can split the cost bound  $C$  among  $n$  replicas such that each replica  $i$  has local access to  $c_i$  resources where

$$\sum_{i=1}^n c_i = C. \quad (ii)$$

However we would like to change the  $c_i$  dynamically. This necessitates some level of interaction among the replicas. Therefore we let this interaction include the pessimistic processing and validation cycle.

Furthermore, application characteristics can permit temporary over-allocation of resources (like over-booking), hence the cost bound  $c_i$  can be set at each replica  $i$ , such that

$$\sum_{i=1}^n c_i = C \geq R, \text{ where } R \text{ is the no. of resources initially available.} \quad (iii)$$

$C$  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

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

and  $w_i$  is some weighting factor. The mechanism for choosing  $w_i$  may differ with the application.

We propose a dynamic  $C$  value which changes in two ways: one in response to optimistic local processing and the other in response to pessimistic processing. Each replica decrements/increments its cost bound whenever it processes an allocate/de-allocate resource transaction optimistically. For testing we decided that pessimistic processing should use the following simple formula to generate a new total cost bound:

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

where:

$C_{new}$  is new cost bound;

$C_{init}$  is cost bound initially set;

$RA$  is no. of resources allocated via requests;

$R$  is no. of resources initially available.

After a new cost bound has been obtained in this way (i.e. after permanent processing), equation (iv) is used to distribute the new cost bound among the replicas. In our earlier testing we used as the weight  $w_i = 1/n$ . This did not allow any use of the individual node's behavior in determining its share of the cost bound.

After a pessimistic cycle we would like the behavior of a replica to have an effect on its share of the new cost bound. This is an area for further study, but at the moment we use the following simple and effective scheme.

Let the resources allocated at replica  $i$  after a pessimistic cycle be  $RA_i$ , therefore

$$RA = \sum_{i=1}^n RA_i . \text{ If we let } w_i = RA_i/RA, \text{ then a replica that}$$

allocated zero resources will get a new cost bound of zero. Therefore we let

$$w_i = (RA_i + 1)/(RA + n). \quad (vi)$$

Note that equation (vi) reduces to  $w_i = 1/n$ , when no resources have been allocated, which is the initial case. This simple feedback formula represents in our view a significant upgrade in our system. We indicate below, under Results, the positive effect the feedback equation (vi) has had on the behavior of the system.

### 3. Transaction Processing and Testing

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. These results are not committed to the database. 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) and their results are committed to the database.

As was indicated above in the Introduction an earlier prototype of the system was tested in controlled LAN environments. It was first coded in C but a Java version now exists. In these tests we did not use equation (vi) but used  $w_i = 1/n$  for each cycle. Equation (vi) has been incorporated into our latest Java prototype.

A transaction generator located at a separate computer initiates transactions and sends them to remote servers for processing. Front-end processing and replica management are done at the servers. The system includes servers on a LAN at Rowan University interconnected via the Internet with a server at Richard Stockton College in southern New Jersey. Another server in the southern Caribbean at the University of the West Indies in Trinidad is not yet commissioned.

The LAN platform consists of four Dell GX110 PCs (128 MB RAM, 700 MHz Pentium III) running Knoppix V3.3-2003-11-19-EN. Sun Microsystem's Java version 1.4.2 is installed on each machine. One of the four machines is used as the transaction generator and the other three as servers, handling transactions from the generator. The fourth server is at Richard Stockton College of New Jersey, about forty miles from Rowan University. It is a Dell PowerEdge 2300 with dual 400 MHz Pentium IIs and 1 gigabyte of RAM, running Slackware Linux version 9.

The transaction generator and servers are all started with a single shell script on the transaction generator machine. The remote shell command `ssh' is used to start the servers from the shell script. The parameters of the simulation (as in Innis et al [7]) are passed from the shell script to the transaction generator program and the server programs via command line arguments.

The results file contains a record for each transaction. This file is analyzed by a statistics program invoked at the end of the shell script. This program counts

- the number of transactions processed optimistically
- the number of transactions that had to be undone
- the percentage of transactions processed optimistically
- two percentages of undone transactions (as a fraction of all transactions and as a fraction of optimistically processed ones).

For each transaction in the results file, the program calculates the ratio of the time to process pessimistically (PST) to the time to process optimistically (OST). For all transactions, the maximum and average of these ratios is computed. A count of ratios in various

ranges (1 to 5, 5 to 10, 10 to 15, exceeding 15) is also determined.

#### 4. Results

We looked at the results from a series of test runs. The resources are available seats on a flight. Transactions book or cancel seats on the flight. We chose values that we felt were easy to reason about. In each test 200 transactions were generated. The transactions were generated at average arrival rates of 1 per second, 5 per second and 10 per second. The number of seats available initially was 200, and the initial global cost bound was 232. All of these values are accepted as input parameters to the system. These tests were run over a shared LAN and the Internet. Therefore service times are affected by network traffic over which we have no control.

At an average of 10 transactions per second we compared the results using  $w_i = 1/n$  with that using  $w_i = (RA_i + 1)/(RA + n)$  i.e. equation (vi) when the system is run on the 3 local (Rowan University) servers. In Table 1 we sent 198 transactions to a single server; each transaction booked 2 seats. The service time ratios were smaller with eq. (vi), yet there were more transactions done optimistically. In Table 2 we distributed the transactions in a ratio 2:4:5 among the servers; the ratio of booking transactions to cancellations was 14:1; the number of seats involved in each transaction ranged randomly from 3 to 9. The average service time ratios are about the same. Again equation (vi) did many more transactions optimistically.

In Tables 3 to 6 we generated transactions at 5, 10 and 1 per second, using  $w_i = (RA_i + 1)/(RA + n)$  i.e. equation (vi) with the system running on the 3 local (Rowan University) servers, on 2 local and 1 remote (Richard-Stockton) server, and on 3 local and 1 remote server. The distribution of transactions is 2:4:5 for 3 servers and 2:4:5:3 for 4 servers. In Tables 3 and 4 each transaction books 2 seats. The indeterminate network traffic effect is evident in the differences in service time ratios. However, high percentages of transactions were done optimistically. Furthermore since there were only 200 seats available initially any excess over 100 transactions (i.e.  $100 * 2$  seats) were correctly undone.

In Tables 5 and 6 the number of seats involved in each transaction ranged randomly from 3 to 9 and the ratio of booking transactions to cancellations was 14:1. Permanent processing has higher priority than temporary processing therefore, as the transaction rate decreases (from 10/sec in Table 5 to 1/sec in Table 6) the transaction queues grow at a slower pace, thus reducing the number of transactions done optimistically.

#### 5. Conclusion

A distributed service has been designed and implemented to provide high availability in accessing replicas in a distributed system. The viability of the system has been established and Java test-beds are being used to explore different feedback mechanisms, and to examine ways in which the performance can be improved. One immediate concern is the number of messages passed. We would like to reduce this number without interfering with the logic of the design. Of interest too is the use of alternative cost bound schemes.

Arrival rate of Transactions – 10/ sec	Weight = 1/n	Weight Eqn. (vi)
<b>3 Local Servers</b>		
Number of Transactions	200	200
Number done Optimistically	145	148
Number Undone	45	48
Done optimistically as fraction of all	0.725	0.74
Undone as fraction of done optimistically	0.31	0.324
Undone as fraction of all	0.225	0.24
Average PST/OST ratio	136.5	84.95
Maximum PST/OST ratio	1030	728
Number with 1.0 <= ratio < 5.0	1	2
Number with 5.0 <= ratio < 10.0	3	2
Number with 10.0 <= ratio < 15.0	1	7
Number with 15.0 <= ratio	140	137

**Table 1. 1 favored server each transaction booking 2 seats.**

Arrival rate of Transactions – 10/ sec	Weight = 1/n	Weight Eqn. (vi)
<b>3 Local Servers</b>		
Number of Transactions	200	200
Number done Optimistically	75	97
Number Undone	24	38
Done optimistically as fraction of all	0.375	0.485
Undone as fraction of done optimistically	0.32	0.39
Undone as fraction of all	0.12	0.19
Average PST/OST ratio	36.01	39.9
Maximum PST/OST ratio	185.53	298.5
Number with 1.0 <= ratio < 5.0	10	14
Number with 5.0 <= ratio < 10.0	7	16
Number with 10.0 <= ratio < 15.0	6	11
Number with 15.0 <= ratio	52	56

**Table 2.**

#### 6. Acknowledgements

We are grateful to Richard Stockton College for the use of one of their machines in our research and to Professor Michael Olan and Computer and Telecommunications staff member Robert Heinrich for their kind assistance.

Arrival rate of Transactions – 5 / sec	3 Local Servers	2 Local 1 Remote	3 Local 1 Remote
Number of Transactions	200	200	200
Number done Optimistically	90	115	124
Number Undone	0	15	24
Done optimistically as fraction of all	0.45	0.575	0.62
Undone as fraction of done optimistically	0.0	0.13	0.194
Undone as fraction of all	0.0	0.075	0.12
Average PST/OST ratio	5.16	571.5	740.1
Maximum PST/OST ratio	19.9	3365	5890
Number with 1.0 <= ratio < 5.0	53	0	1
Number with 5.0 <= ratio < 10.0	28	0	1
Number with 10.0 <= ratio < 15.0	6	0	0
Number with 15.0 <= ratio	3	115	122

**Table 3. Weight = equation (vi).**

Arrival rate of Transactions – 10 / sec	3 Local Servers	2 Local 1 Remote	3 Local 1 Remote
Number of Transactions	200	200	200
Number done Optimistically	117	92	93
Number Undone	17	0	0
Done optimistically as fraction of all	0.585	0.46	0.465
Undone as fraction of done optimistically	0.145	0.0	0.0
Undone as fraction of all	0.085	0.0	0.0
Average PST/OST ratio	65.8	334.6	808.1
Maximum PST/OST ratio	423.7	2352.6	5439.7
Number with 1.0 <= ratio < 5.0	1	0	0
Number with 5.0 <= ratio < 10.0	2	1	1
Number with 10.0 <= ratio < 15.0	3	2	0
Number with 15.0 <= ratio	111	89	92

**Table 4. Weight = equation (vi).**

Arrival rate of Transactions – 10/ sec	3 Local Servers	2 Local 1 Remote	3 Local 1 Remote
Number of Transactions	200	200	200
Number done Optimistically	72	77	85
Number Undone	29	38	40
Done optimistically as fraction of all	0.36	0.385	0.425
Undone as fraction of done optimistically	0.4	0.494	0.47
Undone as fraction of all	0.145	0.19	0.2
Average PST/OST ratio	37	254.1	462.2
Maximum PST/OST ratio	264.9	2876	4204
Number with 1.0 <= ratio < 5.0	15	0	0
Number with 5.0 <= ratio < 10.0	9	2	0
Number with 10.0 <= ratio < 15.0	5	3	0
Number with 15.0 <= ratio	43	72	85

**Table 5.**

Arrival rate of Transactions – 1 / sec	3 Local Servers	2 Local 1 Remote	3 Local 1 Remote
Number of Transactions	200	200	200
Number done Optimistically	45	46	56
Number Undone	0	0	0
Done optimistically as fraction of all	0.225	0.23	0.28
Undone as fraction of done optimistically	0.0	0.0	0.0
Undone as fraction of all	0.0	0.0	0.0
Average PST/OST ratio	3.98	23.1	26.6
Maximum PST/OST ratio	17.42	124.4	182.8
Number with 1.0 <= ratio < 5.0	31	7	5
Number with 5.0 <= ratio < 10.0	12	10	13
Number with 10.0 <= ratio < 15.0	1	11	8
Number with 15.0 <= ratio	1	18	30

**Table 6.**

## References

- Bernstein, P.A., Hadzilacos, V. and Goodman, N., 1987. *Concurrency Control and Recovery in Database Systems*, Addison-Wesley Pub. Co., Reading, Ma.
- Birman, K.P., 1996. *Building Secure and Reliable Network Applications*, Manning Pub. Co.
- 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.
- Crichlow, J.M., 2000. *The Essence of Distributed Systems*, Prentice Hall/Pearson Education, U.K.
- 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.
- 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.
- Innis, C., Crichlow, J., Hosein, M. and Hartley, S., 2002. A Java System that combines Optimism and Pessimism in updating Replicas, *Proceedings of the IASTED International Conference on Software Engineering and Applications*, Cambridge, MA, November 4-6.
- 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.

9. Krishnakumar, N & Bernstein, A.J., 1991. Bounded Ignorance in Replicated Systems, *Tenth ACM Symp. On Principles of Database Systems*.
10. 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.
11. Wolfson, Ouri; Jajodia, Sushil and Huang, Yixiu, 1997. An adaptive data replication algorithm. *ACM Transactions on Database Systems*, 22(2), 255-314.
12. 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.