

USING AN XML FILE TO TEST A DISTRIBUTED REPLICA ACCESSING SYSTEM

Joel M. Crichlow, Rowan University, Glassboro, NJ 08028, crichlow@rowan.edu
Stephen Hartley, Rowan University, Glassboro, NJ 08028, hartley@elvis.rowan.edu
Michael Hosein, University of the West Indies, Trinidad, mhosein@tstt.net.tt
David Ivins, Rowan University, Glassboro, NJ 08028, davidivins@gmail.com

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. 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 system is currently used as a prototype and is being tested using data input via an XML file. This is in keeping with the movement towards Web Services and the use of current standard technologies for data representation.

KEY WORDS

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

1. Introduction

Our system is called **COPAR**: it **C**ombines **O**ptimism and **P**essimism in **A**ccessing **R**eplicas in a distributed transaction processing system. 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 [7], Krishnakumar &

Bernstein [8], Lynch et al [9], Wolfson et al [10], Yu and Vahdat [11].

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.

This is an ongoing project which follows work published earlier (see Innis et al [6] and Crichlow et al [5]). We have increased the number of servers from 4 to 6, including one thousands of miles away in the southern Caribbean, and we now use XML files to drive our test bed. In order to provide the logical framework we include the following brief review.

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

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

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

$$\sum_{i=1}^n c_i = C \geq R, \text{ where } R \text{ is the number of resources initially available.} \quad (\text{iii})$$

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

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

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:

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

where:

- C_{new} is new cost bound;
- C_{init} is cost bound initially set;
- RA is number of resources allocated via requests;
- R is number 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

$RA = \sum_{i=1}^n ra_i$. If we let $w_i = ra_i/RA$, then a replica that allocated zero resources will get a new cost bound of zero. Therefore we let

$$w_i = (ra_i + 1)/(RA + n). \quad (\text{vi})$$

Note that equation (vi) reduces to $w_i = 1/n$, when no resources have been allocated, which is the initial case.

3. Transaction Processing

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 (see Figure 1).

In other words, transactions 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.

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 this response to the owner of the transaction. When the parent receives the response from a permanent run it can publish the response 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.

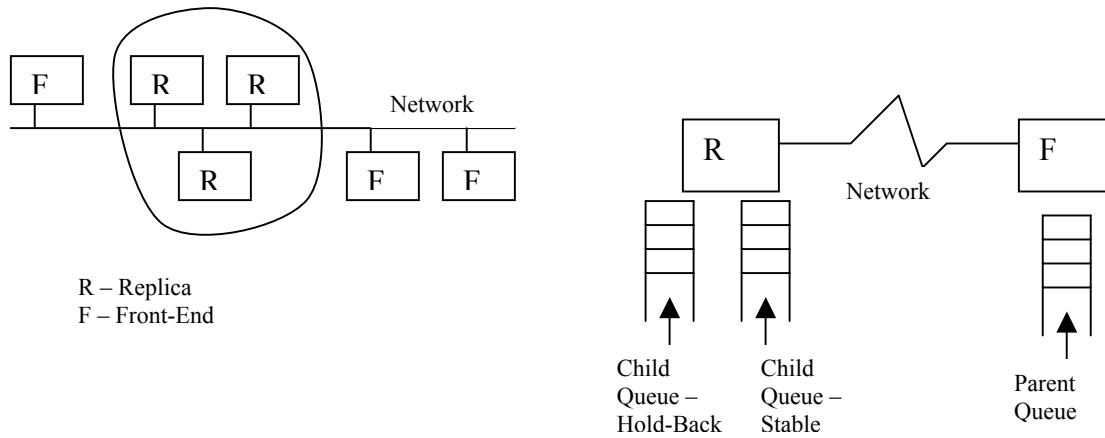


Figure 1. Front-Ends receive and own transactions. Transaction-parents are queued at front-ends, children are sent to the replicas for optimistic and pessimistic processing. Optimistic processing is done on hold-back queue, pessimistic processing is done on stable queue.

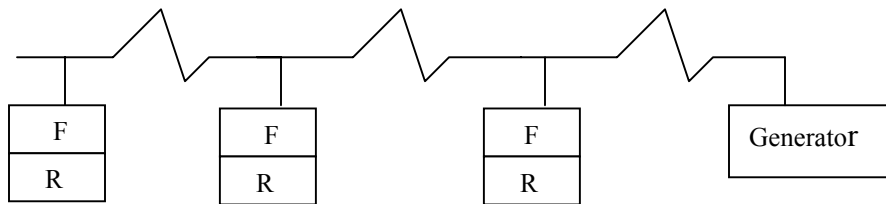


Figure 2. A transaction generator at a separate computer sends transactions to servers which do front-end processing and update replicas

4. Testing

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 (see Figure 2). The system includes servers on a LAN at Rowan University interconnected via the Internet with a server at Richard Stockton College in New Jersey and a server at the University of the West Indies located in Trinidad in the southern Caribbean.

The LAN platform consists of four Dell GX110 PCs (128 MB RAM, 700 MHz Pentium III) running Knoppix V3.3-2003-11-19-EN; and two Dell Optiplex GX1 models, with 500 MHz CPUs and 128 MB of RAM, running the KNOPPIX V5.0.1 2006-06-01 OS. Sun Microsystem's Java version 1.4.2 is installed on each machine. One of the six machines doubles as the transaction generator and a server, and the other five are used as servers. The servers handle transactions from the generator.

One remote 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 other remote server is at the University of the West Indies, St Augustine, Trinidad in the southern Caribbean. That machine is a Dell Optiplex Model GX1 with Pentium II 400 MHz and 192 MB RAM, running Red Hat Linux release 9.

The transaction generator and servers are all started with a program running on the transaction generator machine that reads and parses an XML file containing the data for the run. This program uses the

```
Runtime.getRuntime().exec(String[] arguments)
```

method of Java to run processes external to the JVM. To start the servers from the program, the arguments array consists of

```
arguments[0] = "/bin/bash";
arguments[1] = "-c";
arguments[2] = "ssh ...";
```

where "ssh ..." is the remote shell command followed by the command-line arguments for a server. Parameters include:

- the number of transactions to generate
- the average generation rate
- the names of the server machines
- the minimum and maximum transaction amounts
- the percentage of transactions representing allocations and deallocations
- the percentage of transactions sent to each server
- the initial number of resources
- the initial cost bound (percentage overbooking allowed) for each server
- the names of files in which various program outputs and results are stored

The following is an example of our XML file:

```
<hads>
  <cost-bound>200</cost-bound>
  <cost-pct>116</cost-pct>
  <gen-delay>200</gen-delay>
  <min-change>3</min-change>
  <max-change>9</max-change>
  <num-trans>200</num-trans>
  <dec-ratio>10</dec-ratio>
  <inc-ratio>5</inc-ratio>
  <num-servers>6</num-servers>
  <w-sub>false</w-sub>
  <server>
    <server-name>boerne.rowan.edu</server-name>
    <server-ratio>1</server-ratio>
  </server>
  <server>
    <server-name>64.28.143.30</server-name>
    <server-ratio>2</server-ratio>
  </server>
  <server>
    <server-name>antonio.rowan.edu</server-name>
    <server-ratio>3</server-ratio>
  </server>
  <server>
    <server-name>poteet.rowan.edu</server-name>
    <server-ratio>4</server-ratio>
  </server>
  <server>
    <server-name>zeus.stockton.edu</server-name>
    <server-ratio>5</server-ratio>
  </server>
  <server>
    <server-name>schertz.rowan.edu</server-name>
    <server-ratio>6</server-ratio>
  </server>
  <generator>boerne.rowan.edu</generator>
</hads>
```

The results file contains a record for each transaction. The fields of the record are:

- transaction number
- name of the server to which a transaction was sent
- time (in milliseconds) that the transaction was received by the server
- time that pessimistic processing was completed by the server
- whether or not the transaction was handled optimistically by the server
- if so, time that optimistic processing was completed by the server
- whether or not a transaction processed optimistically had to be undone later by the pessimistic processor

The results file is analyzed by a statistics program at the end of execution of the generator and servers. 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.

5. Results

Our primary interests were in how the system responded to input read from an XML file, and how well did it scale upwards. 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 all the tests we kept certain parameters the same. In each test 200 transactions were generated. The number of seats available initially was 200, and the initial global cost bound was 232. The ratio of booking transactions to cancellations was 2:1; the number of seats involved in each transaction ranged randomly from 3 to 9.

We changed a number of parameters over the tests. The transactions were generated at average arrival rates of 5 per second and 1 per second. The number of servers participating in the tests ranged from 3 to 6. We compared the results using $w_i = 1/n$, with that using

$w_i = (ra_i + I)/(RA + n)$ i.e. equation (vi).

The distribution of ownership of transactions over the servers was also changed in the tests, however we have not yet analyzed the effect of this distribution on the performance.

All of these values are accepted as input parameters via the XML file. 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. In general the system's performance met our expectations.

At an average of 5 transactions per second we compared the results using $w_i = 1/n$ (see Table 1) with that using $w_i = (ra_i + I)/(RA + n)$ i.e. equation (vi) (see Table 2). Results were obtained when the system is run on 3 local (Rowan University) servers, 2 local servers and 1 at Richard-Stockton (RS), 2 local servers and 1 at the University of the West Indies (UWI), and 3 servers one at each of the three sites. The results in Tables 1 and 2 are not significantly different. In each case process time ratios increased with the inclusion of remote nodes.

When the servers are all local the number of transactions done optimistically is less than that done when there are remote nodes participating. This is understandable since the addition of the remote nodes would increase the processing time for pessimistic processing, thus allowing the optimistic processing more time to empty their transaction queues.

Tables 2 and 3 show the performance at 5 transactions per second as we increase the number of servers. The process time ratios show a logical increase which indicates that the system remains stable as the number of servers is scaled upwards.

At 1 transaction per second (see Table 4) a noticeable advantage appeared when the servers were all local (5 and 6 Rowan University servers). As expected the number of transactions done optimistically was reduced since the transaction queues grew at a slower pace. Furthermore, the processing time ratios decreased at the slower arrival rate, since the pessimistic processor was able to reduce the wait time of transactions in its queue.

6. Conclusion

A distributed service has been designed and implemented to provide high availability in accessing replicas in a distributed system. The system runs over the Internet at computers some thousands of miles apart. The viability of the system has been established and Java test-beds are being used to examine the performance under realistic conditions. Several key parameters affecting the configuration and behavior of the system can be modified via input read from XML files. This is another step towards making the system web service compatible.

Arrival rate of Transactions – 5 / sec Weight = 1/n	3 Local Servers	2 Local + RS	2 Local + UWI	1 Local + RS + UWI
Number of Transactions	200	200	200	200
Number done Optimistically	141	200	200	200
Number Undone	3	34	34	34
Done optimistically as fraction of all	0.705	1.0	1.0	1.0
Undone as fraction of done optimistically	0.0213	0.17	0.17	0.17
Undone as fraction of all	0.015	0.17	0.17	0.17
Average PST/OST ratio	4.5625	219.321	7994.3	7224.8
Maximum PST/OST ratio	25.1	1491.9	55141.4	104205.4
Number with 1.0 <= ratio < 5.0	98	12	0	0
Number with 5.0 <= ratio < 10.0	34	5	0	0
Number with 10.0 <= ratio < 15.0	5	5	0	0
Number with 15.0 <= ratio	4	178	200	200

Table 1

Arrival rate of Transactions – 5 / sec Weight = eqn. (vi)	3 Local Servers	2 Local + RS	2 Local + UWI	1 Local + RS + UWI	4 Local Servers	3 Local + RS	3 Local + UWI	2 Local + RS + UWI
Number of Transactions	200	200	200	200	200	200	200	200
Number done Optimistically	134	200	200	199	154	200	200	200
Number Undone	3	34	34	34	5	34	34	34
Done optimistically as fraction of all	0.67	1.0	1.0	0.995	0.77	1.0	1.0	1.0
Undone as fraction of done optimistically	0.0224	0.17	0.17	0.1709	0.03247	0.17	0.17	0.17
Undone as fraction of all	0.015	0.17	0.17	0.17	0.025	0.17	0.17	0.17
Average PST/OST ratio	4.403	210.033	6859.59	7077.66	7.0276	414.827	8765.74	6558.9
Maximum PST/OST ratio	21.25	2305.67	51292.6	71433.89	29.0	3693.5	71229.5	69246.7
Number with 1.0 <= ratio < 5.0	92	9	0	0	74	0	0	0
Number with 5.0 <= ratio < 10.0	32	5	0	0	52	2	0	1
Number with 10.0 <= ratio < 15.0	6	3	0	0	14	4	0	0
Number with 15.0 <= ratio	4	183	200	199	14	194	200	199

Table 2

Arrival rate of Transactions – 5 / sec Weight = eqn. (vi)	5 Local Servers	4 Local + RS	4 Local + UWI	3 Local + RS + UWI
Number of Transactions	200	200	200	200
Number done Optimistically	200	200	200	200
Number Undone	34	34	34	34
Done optimistically as fraction of all	1.0	1.0	1.0	1.0
Undone as fraction of done optimistically	0.17	0.17	0.17	0.17
Undone as fraction of all	0.17	0.17	0.17	0.17
Average PST/OST ratio	155.948	598.206	9030.52	4620.44
Maximum PST/OST ratio	929.692	3966.0	76463.8	43088.2
Number with 1.0 <= ratio < 5.0	11	0	0	0
Number with 5.0 <= ratio < 10.0	7	1	0	0
Number with 10.0 <= ratio < 15.0	7	2	0	0
Number with 15.0 <= ratio	175	197	200	200

Table 3

Arrival rate of Transactions – 1 / sec Weight = eqn. (vi)	5 Local Servers	4 Local + RS	4 Local + UWI	3 Local + RS + UWI	6 Local Servers	5 Local + RS	4 Local + RS + UWI
Number of Transactions	200	200	200	200	200	200	200
Number done Optimistically	149	200	200	200	143	148	200
Number Undone	3	34	34	34	4	4	34
Done optimistically as fraction of all	0.745	1.0	1.0	1.0	0.715	0.74	1.0
Undone as fraction of done optimistically	0.0201	0.17	0.17	0.17	0.028	0.027	0.17
Undone as fraction of all	0.015	0.17	0.17	0.17	0.02	0.02	0.17
Average PST/OST ratio	8.436	2606.85	10033.22	4647.46	9.609	25.846	4689.8
Maximum PST/OST ratio	35.417	22620.0	81875.45	62793.0	58.684	189.8	60987.8
Number with 1.0 <= ratio < 5.0	54	19	0	1	43	8	0
Number with 5.0 <= ratio < 10.0	53	5	0	0	49	24	0
Number with 10.0 <= ratio < 15.0	24	11	0	0	29	30	0
Number with 15.0 <= ratio	18	165	200	199	22	86	200

Table 4 (Fifth server not yet cleared by UWI firewall therefore results from 5 local + UWI not yet available)

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.

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.
- Crichlow, J.M., Hartley, S., Hosein, M., and Innis, C., 2004. The COPAR service: Combining Optimism and Pessimism in Accessing Replicas, *Proceedings of the Third IASTED International Conference on Communications, Internet and Information Technology*, St. Thomas, US Virgin Islands, November, 558-563.
- 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.
- Krishnakumar, N & Bernstein, A.J., 1991. Bounded Ignorance in Replicated Systems, *Tenth ACM Symp. On Principles of Database Systems*.
- 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.
- Wolfson, Ouri; Jajodia, Sushil and Huang, Yixiu, 1997. An adaptive data replication algorithm. *ACM Transactions on Database Systems*, 22(2), 255-314.
- Yu, H. & Vahdat, A., 2001. The Costs and Limits of Availability for Replicated Services, *Proceedings of the 18th ACM Symposium on Operating systems Principles*, Alberta, Canada, October 21-24, 29-42.