A Competitive Dynamic Data Replication Algorithm*

Yixiu Huang, Ouri Wolfson
Electrical Engineering and Computer Science Department
University of Illinois at Chicago

Abstract

In this paper, we present a distributed algorithm for dynamic data replication of an object in a distributed system. The algorithm changes the replication scheme, i.e., number of replicas and their location in the distributed system, to optimize the amount of communication. In other words, the algorithm dynamically adapts the replication scheme of an object to the pattern of read-write requests in the distributed system. We prove that the cost of the algorithm is within a constant factor of the lower bound.

1. Introduction

1.1 Motivation

The replication scheme of a distributed system determines how many replicas of each object are created, and to which processors these replicas are allocated. This scheme critically affects the performance of a distributed system, since reading an object locally is less costly than reading it from a remote site, Therefore in a read-intensive network a widely distributed replication is mandated. On the other hand, an update of an object is usually written to all, or a majority of the replicas, and therefore in a write-intensive network a narrowly distributed replication is mandated. In other words, the optimal replication scheme depends on the read-write pattern for each object.

Traditionally, in computer networks, objects are replicated in a static fashion, i.e. the data replication scheme is designed by the distributed-database manager and it remains fixed until manual reallocation is executed by the manager. If the read-write patterns are fixed and are known a priori, this is a reasonable solution. However, if the read-write patterns change dynamically, in unpredictable ways, a static replication scheme may lead to severe performance problems.

*This research was supported partly by NSF grant IRI-90-03341. – **It appears in the Proceedings of the 9-th Intenational Conference on Data Engineering, pp. 310 - 317, April 1993, Vienna, Austria**

In this paper we propose a practical algorithm, called Competitive-Dynamic-Data-Replication (CDDR), that changes the replication scheme (i.e. the sites which store a replica of the object) of an object dynamically as the read-write pattern of the object changes in the network. We assume that the changes in the read-write pattern are not known a priori.

The algorithm CDDR is based on the primary-copy model, it preserves 1-copy-serializability, and it is distributed in the sense that the decision on whether or not to store a copy of the object is made by each individual site based on locally collected statistics. The algorithm changes the replication scheme of an object to optimize a global cost function for the current read-write pattern. As the read-write pattern changes, so does the replication scheme. Specifically, if during a period of time it is (communication-) cost effective for some site to store a copy of the object (due to the dominant cost of reads initiated locally), then it will do so. When the site determines that, due to the dominant cost of writes initiated at other processors, it is not cost effective to retain the copy, then it will give it up.

The CDDR algorithm is also integrated. In such an algorithm, redistribution of the replicas is integrated into the processing of reads and writes, instead of executing independently of these operations. So, for example, a processor s creates a replica of an object at some other sites, i, in response to i's request to read the object from s. s creates the replica by piggybacking, on the message to i containing a copy of the object, an indication that i should keep a replica (and that s will propagate writes to i).

We demonstrate the practicality of the algorithm by showing how it should be combined with the concurrency control and recovery mechanisms of the database management system.

One of the main and most difficult results of this paper is to show that the CDDR algorithm is competitive. Specifically, we show that for every schedule of read-write requests to an object, our algorithm does not do much worse than the lower bound. The lower bound is the cost of an algorithm in which the requests are totally ordered, the sequence of requests is known a priori, and any decision to change the replication scheme is done in a centralized fashion. Our algorithm is competitive in the sense that the ratio of its cost to the lower bound is bounded by a constant.

1.2 Relevant literatures

Performance and reliability are the two major purposes of data replication. This work addresses the former.

Many performance-oriented works on replicated data consider the static problem of the replication, namely establishing a priori a replication scheme that will optimize performance, but will remain fixed at runtime. This is called the file-allocation problem, and it has been studied extensively in the literature (see [11] and [12] for a survey). In contrast, our algorithm does not keep the replication scheme constant during runtime.

Another approach to improve the performance in a replicated distributed database, which also assumes a static replication scheme, is to relax the serializability requirement. Works on quasi-copies ([13, 14, 15]), lazy replication (in [16]), and bounded ignorance ([17]) fall in this category. In contrast, our approach preserves one-copy-serializability.

In the theoretical computer science community there has been work on online algorithms (e.g. [1]), particularly for paging (e.g. [18]), searching (e.g. [18]) and caching (e.g. [19]). These works are similar in spirit to the CDDR algorithm in the sense that they address competitiveness. However, the models in such works are inappropriate for managing replication in distributed databases. For example, the replacement issue (i.e. which page to replace), that is important in paging, usually is not a factor in replicated data management.

There has also been work addressing dynamic data replication algorithms in [20]. However the algorithms there do not allow concurrent requests, and require centralized decision making by a processor that is aware of all the requests in the network. In contrast, our algorithm is distributed, and allows concurrent read-write requests.

Finally, in [2] we proposed algorithms for dynamic replication. However, these algorithms were dependent on the network having the tree topology, a limitation removed by the CDDR algorithm. In [3], we proposed another dynamic replication algorithm that was also independent of the of the network topology. However, in contrast to CDDR, none of the algorithms (in [2] and [3]) is competitive.

The rest of this paper is organized as follows. Section 2 presents the primary-copy model, and the CDDR algorithm. Section 3 discusses practical issues related to the implementation of the algorithm. Section 4 analyzes the CDDR algorithm from the competitiveness point of view. Section 5 provides final comments and discussed future work.

2. The CDDR algorithm

In this section we present the *CDDR* (Competitive Dynamic Data Replication) algorithm.

We define an *object* to be a unit of data to be replicated, and the *replication scheme* of the object to be the set of sites which hold an object replica. A *data site* is a site that belongs to the replication scheme. A *non data site* is a site that does not belong to the replication scheme. We suppose the read-write model is the following (called here as the *primary-copy* model). At any point in time one of the data sites is designated as the primary site (we denote it by p). Initially the replication scheme contains p alone. Whenever a data site issues a read request for the object, it is serviced locally. When a non-data site issues a read, the request is forwarded to, and serviced by p. A site s writes the object by sending it to p, and in turn p propagates the write to all the available data sites. In other words, the update policy is *'read one write all available'*[5].

Every site in the network has a status. Status 1 indicates that this site is a data site, status 0 means that this site is a non data site. Every site knows its own status and where the current primary site is. The primary site knows every site's status. We define the *r-write* (remote write) of a site s to be the write request issued from any other site (i.e. not from s) in the network.

For every site s there are two associated counters, based on which the status of s is determined. The first is the read-counter, which counts the number of reads s issued. The second counter is the r-write counter, which counts the number of r-writes of site s.

The counters of a data site reside at the data site. These counters can be maintained since the data site knows how many reads it issues, and how many r-writes the primary site propagates to it. All the counters of the non data sites reside at the primary site p. These counters can be maintained at p, since p knows how many reads a non data site issues, and how many writes any site issues. Therefore, the primary site decides for the non data sites whether or not they enter the replication scheme, based on their counters.

Intuitively, these two counters establish the following trade-off. A site s being in the replication scheme increases the communication cost of every r-write, since it has to be propagated to s, and it decreases the communication cost of every read from s. Observe that whether or not s is in the replication scheme does not affect the communication cost of a write from s, nor does it affect the communication cost of a read issued by a site other than s.

At the intuitive level, the replication scheme changes as follows. If the primary site receives more read requests from a non data site j, then it will tell j to enter the replication scheme; whereas if the primary site receives more write requests from sites other than j, then it will keep j out of the replication scheme. The data sites decide by themselves whether or not to exit from the replication scheme based on the counters they have. If the data site i issues more read requests, then it will stay in the replication scheme, whereas if i receives more r-write requests from the primary site, then it will exit from the replication scheme. When the primary site needs to exit from the replication scheme, it has to choose a new site to inherit the primary role.

The following algorithm is the formal description of how these counters are used for the dynamic replication and reallocation. The algorithm has a parameter k, which determines how frequently the replication scheme changes. In section 4.5, we will discuss the tradeoffs in choosing k.

CDDR(k) Algorithm

1. After issuing a read request, a data site i increments its read-counter by one. If i's read-counter reaches k, then i remains in the replication scheme and initiates the following *Reset Action*;

   Reset Action: Reset the read-counter and the r-write counter to 0.

2. Upon receiving a read from a non data site j, the primary site p increments j's read-counter by one. If the read-counter reaches k, then p responds by sending a copy of the object to j and tells j to enter the replication scheme. Specifically, p initiates the following *Join Action* for j;

   Join Action: p changes site j's status from 0 to 1, and it tells j to keep the copy and to initialize its own counters.

3. Upon receiving a propagated write from p, a data site s, which is not the primary site (i.e. $s \neq p$), does the following. s increments its r-write counter by one. If its r-write counter reaches k, then it exits from the replication scheme and informs the primary site of this change, by initiating the following *Exit Action*;

   Exit Action: Reset the status of s to 0, and tell p to initialize the counters for s.

4. Upon receiving a write request from any site, p propagates the write to every available data site, and p increments by one the r-write counters of the non-data sites. For each r-write counter at p that reaches k, p initiates the *Reset Action* (see above). Additionally, if the write request is not initiated from p, and p's own r-write counter reaches k, then p exits from the replication scheme by initiating the following *Switch Action*;

   Switch Action:

   (a1) If currently there is another site in the replication scheme, then p chooses one, say s, to inherit the primary role;

   (a2) Otherwise p selects the sender of the write request, say s, to inherit the primary role, and p initiates the *Join Action* for s;

   (b) Reset the status of p to 0, and tell s to initialize the counters for p;

   (c) p sends to s the counters and statuses it kept;

   (d) p broadcasts a message to every site in the network, indicating that s becomes the new primary site.

We define an *Action* for a site s to be either an *Exit*, or a *Reset*, or a *Join*, or a *Switch*. The following example demonstrates the algorithm *CDDR*(k) for (k=2).

Example 1: Suppose the network consists of three sites {1, 2, 3 }. Initially there is a single replica of the object residing at site 1. The communication structure is as shown in Fig. 1 (the dark round site indicates the primary site, a dark box site means a data site, a light box site means a non data site). The request sequence is $r_1^1 r_3^2 r_2^3 w_1^4 r_3^5 w_3^6 r_2^7 r_3^8 w_2^9 r_1^{10} w_1^{11}$, where the superscript indicates the order of the request in the sequence, and the subscript tells at which site the request is initiated. For example, r_3^8 means that the eighth request is a read issued from site 3, w_1^4 means that the fourth request is a write from site 1.

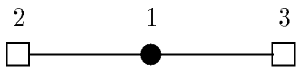

Figure 1.

To start the *CDDR*(2) algorithm, we initialize site 1's status to 1, the others' status to 0, and all sites' counters to 0 at site 1. The table below indicates the sites' status (shadow means a site currently in the replication scheme) after each request, the values of the read-counter (above the slash line) and the r-write counter (below the slash line) after each request.

After request 1, site 1's read-counter becomes 1; After request 2, site 3's read-counter becomes 1; After request 3, site 2's read-counter becomes 1; After request 4, the r-write counters of site 2 and site 3 become 1; Request 5 makes site 3's read-counter reach $k = 2$, site 1 (the primary site) initiates the *Join Action* for site 3, 3 becomes a new data site, and its two counters are both initialized at site 3; The communication structure is shown in Fig. 2. After request 6, site 1's r-write counter becomes 1. Request 6 makes site 2's r-write counter reach $k = 2$, hence the primary site (site 1) initiates the *Reset Action* for site 2, and site 2's counters are thus becoming 0's. After request 7, site 2's read-counter becomes 1; After request 8, site 3's read-counter becomes 1; After request 9, site 3's r-write counter becomes 1, and request 9 makes site 1's r-write counter reach $k = 2$, hence site 1 (the primary site) initiates a *Switch Action* for itself, its primary role is switched to site 3 (the other current data site), and 1's own counters are both cleared to 0's. The communication structure is shown in Fig. 3.

read \ r-write	r^1_1	r^2_3	r^3_2	w^4_1	r^5_3	w^6_3	r^7_2	r^8_3	w^9_2	r^{10}_1	w^{11}_1
site 1	0 \ 1	0 \ 1	0 \ 1	0 \ 1	0 \ 1	1 \ 1	1 \ 1	1 \ 1	0 \ 0	0 \ 1	0 \ 0
site 2	0 \ 0	0 \ 0	0 \ 1	1 \ 1	1 \ 1	0 \ 0	0 \ 1	0 \ 1	0 \ 1	0 \ 1	1 \ 1
site 3	0 \ 0	0 \ 1	0 \ 1	1 \ 1	0 \ 0	0 \ 0	0 \ 0	0 \ 1	1 \ 1	1 \ 1	0 \ 0

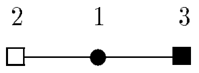

Figure 2.

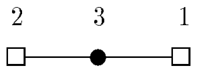

Figure 3.

After request 10, site 1's read-counter becomes 1; After request 11, site 2's r-write counter becomes 1, and this request makes site 3's r-write counter reach $k = 2$. Site 3 initiates a *Switch Action* to switch the primary role from site 3 to site 1 (the writer, a non data site). The communication structure becomes the same as in Fig. 1. □

3. Practical issues

In this section we discuss how the *CDDR* algorithm is implemented in a distributed environment. We address the issues of concurrent read-write requests issued by different transactions, performance considerations, and coping with failures.

3.1 Concurrency control

In this subsection we show how the CDDR algorithm can be combined with distributed two-phase locking to preserve one-copy serializability [6].

The concurrency control can be performed as follows. Each write request exclusively locks (x-lock) all available copies of the object, and each read request s-locks (shared lock) the primary copy, or the local copy, depending on the site that performs the read request. A transaction which accesses the object is executed as follows. Every site executes the CDDR algorithm (i.e. servicing the request, initiating the *Actions* to change the replication scheme, and sending messages as required) between the lock and the unlock of the object.

The 'read one write all' protocol preserves the one-copy serializability (see [8], [9] and [10]). Although the meaning of 'all' in our context changes dynamically, the replication scheme change happens while the transaction is holding the lock(s), thus each transaction has a fixed view of the replication scheme before it starts, and this replication scheme can not be changed by other transactions during its execution. Therefore one-copy serializability is preserved as in the case of static replication.

3.2 Other performance issues

In this subsection we deal with some special performance issues raised by switching the primary site.

Because the primary role may switch from one site to another, a site s may send a service request to the primary site before s receives the primary-site-switch information. If s receives the primary site switch message before the request is serviced, then s re-sends the request to the new primary site. If an off-switched primary site s receives a request directed to a primary site, then s ignores the request.

In the CDDR algorithm, a write request may cause several *Exit* and *Reset Actions* for different sites as part of the transaction. If a *Switch* is to occur as a result of this write, then, for performance reasons, all these *Actions* should complete before the *Switch*. Otherwise, the status of a data site may have to be transmitted twice from the off-switched primary site to the new primary site.

When the primary site p initiates a *Join Action* for a non data site j, it sends a join message to j. This indication is piggybacked on the copy of the object sent to j.

When a propagated write results in an *Exit Action* of a data site i, i will send an exit message to p. This indication is also piggybacked, this time on the write acknowledgment to p.

3.3 Failure and recovery

In this subsection we discuss how the *CDDR* algorithm copes with failures. We suppose here that all failures are *clean*[5], i.e. the failure can be detected and a failed site is totally down. A site failure may occur at the non data site, non-primary data site or at the primary site. Each one of these cases is treated differently.

A non data site's failure does not affect the execution of read and write requests at the other sites. The primary site may not know that the site has failed.

If a non-primary data site i fails, any x-lock requested by the primary site p (due to a write request) will not be acknowledged. If p does not get any response from a data site i to the x-lock request, then p assumes that i has failed. Then p sets i's status to 0, and continues.

If the primary site fails, any read request submitted from non data site can not be serviced (the s-lock requested is not acknowledged), and the write request submitted from any site is not acknowledged. If a site does not get an acknowledgment from p, it assumes that p has failed. All sites, except the primary, will execute an election protocol to select some data site as the new primary site.

Since the number of copies of the object varies in time, the dynamic replication and allocation algorithm is vulnerable to failures that may render the object inaccessible. To address this problem, the user may impose reliability constraints of the following form: "The number of copies cannot decrease below a threshold, say t." If such constraint is present, then the primary site p refuses to accept the exit of a data site, if such exit will downsize the replication scheme below the threshold. In other words, p informs the site s that the request to exit from the replication scheme is denied; subsequently, writes continue to be propagated to s. s continues to reissue the request whenever the exit comparison dictates to do so. The request may be granted later on, if the replication scheme expanded in the meantime.

When a site s recovers from failure, the recovery procedure will send a recovering message to every site in the network. Only the primary site p responds to this message. The other sites ignore this message. Upon receiving this message, p assumes that s is a non data site, resets its status to 0, and initializes its two counters.

4. Analysis of the CDDR algorithm

In this section, we formulate the cost objective function, and we show that $CDDR(k)$ is $2k$-competitive. Also we discuss the strategy of choosing k.

4.1 Request schedules

A finite sequence of read-write requests of the object, $\psi = o_1 o_2 o_3 ... o_n$, will be called a *schedule*. We denote the consecutive reads of ψ between its i^{th} write and its $(i+1)^{st}$ write by $R^{(i)}$, and the consecutive reads before the first write by $R^{(0)}$. Thus for convenience, we sometime denote the schedule by

$$\psi = R^{(0)} w^1 R^{(1)} w^2 \ldots w^n R^{(n)}$$

where w^i is a single *write* request, $R^{(i)}$ is zero or more *read* requests.

Actually, in practice, the read requests in $R^{(i)}$ may be partially ordered. Although in the proof we assume that the read requests in $R^{(i)}$ happen sequentially, the proof still holds even if the reads are partially ordered.

4.2 Cost function

We suppose that the communication cost of moving the object between two sites is 1. Therefore if the replication scheme consists of sites {1,2,3}, and site 4 reads the object, then the communication cost is 1. If site 4 writes the object, then the communication cost is 3.

For any schedule $\psi = o_1 o_2 \ldots o_n$, we define its *Configured Schedule* to be a request-scheme sequence

$$X_0 o_1 X_1 o_2 X_2 \ldots o_n X_n$$

where X_i is the replication scheme after the i^{th} request (we call X_i the *associated replication scheme* of o_i), and X_0 is the replication scheme before the first request of the schedule.

An *offline* dynamic replication algorithm is one which knows the whole request schedule in advance, and maps the schedule to a configured schedule before the execution. It can configure the optimal schedule for different cost functions.

An *online* dynamic replication algorithm does not have knowledge of the whole schedule, it changes the replication scheme based on the prefix received. Upon receiving a request o_i, an online dynamic replication algorithm (say algorithm A) configures the next replication scheme X_i based on the preceding configured schedule and the current request o_i. In other words, the algorithm A configures the associated replication scheme of a request immediately after the request is issued, and before the next request is issued. For a read request, we define its incurred communication cost to be the most efficient way to replicate the most up-to-date copies in the set X_{i-1} to the set X_i, and to service the request. For a write request, we define its incurred communication cost to be the cost to reallocate the most up-to-date copy (held by the writer) to the set X_i. This will become clear in the next example. For the schedule $\psi = o_1 o_2 ... o_n$, we define the incurred cost of algorithm A for schedule ψ, denoted $COST_A(\psi)$ to be the sum of all the communication costs incurred in each request.

4.3 Competitiveness

After formulating the cost function of a dynamic replication algorithm for an arbitrary schedule, we can define the notion of competitiveness. Competitiveness is a widely-accepted way to measure the performance of an on-line algorithm (see [1, 4]). Intuitively, a c-competitive online dynamic replication algorithm is an algorithm which costs at most c times as much as any other (online or offline) dynamic replication algorithm, for any schedule. Formally, a c-competitive dynamic replication algorithm P is one for which there are two constants c and d, such that for any request sequence ψ, $COST_P(\psi) \leq c \cdot COST_A(\psi) + d$, where A can be any on-line or off-line algorithm. It bounds the worst case cost to be within a constant factor of the optimal algorithm. We will show that the algorithm $CDDR(k)$ is $2k$-competitive.

4.4 Request constraint

We analyze the $CDDR$ algorithm for transactions that obey the following constraint.

NBW Constraint 'no *blind write*' is allowed, i.e., a transaction can have at most one write for an object, and each write must have at least one *referencing read* i.e. a read for the same object that occurs before the write.

This constraint is practically reasonable since one does not usually change the value of an object without reading the object beforehand.

The NBW constraint, combined with a concurrency control mechanism that guarantees one-copy serializability ([6] and [7]) (e.g. two-phase locking), ensures the following. In a schedule, for any object O, between any two writes of O there is at least one read of O. The reason for this is that any write (say from transaction T_1) that occurs between a write (say from transaction T_2) and one of its referencing reads (from transaction T_2) will violate serializability.

4.5 The main analysis result

Our main analysis result is the following.

Theorem: If each transaction is NBW-constrained and each schedule is 1-copy-serializable, then $CDDR(k)$ is $2k$-competitive.

The complete proof of the above theorem is omitted because of space limitations. Intuitively, the proof proceeds in three steps. First, we devise the lower bound on the communication cost. Second, we show that $CDDR(1)$ is 2-competitive under constraint NBW, by comparing it to the lower bound. Third, we show that when using $CDDR(k)$ the cost is at most k times as much as that of using $CDDR(1)$. These three steps combined prove the theorem.

A question that arises at this point is why not use $k = 1$ for maximum competitiveness. The answer is that in this paper, we did not consider the cost of control messages, such as the broadcast notifying all sites of the primary site switch. We only considered the communication cost of transmitting the replicated object, since we assume that the object is much larger than a control message.

However, the control messages cost may become significant when k is small, since a lower k increases the probability of a primary site switch. At the extreme, when $k = 1$, every write causes a switch if the write is not initiated at the primary site. By choosing a larger k, the replication scheme will be more stable and the communication cost of control messages will be relatively small, while the CDDR algorithm will still be competitive.

5. Conclusion and future work

We have presented a practical algorithm for dynamic data replication, and we described how it is implemented in a distributed fashion. Additionally, we proved an important theoretical result stating that the CDDR algorithm is competitive, i.e., its cost is within a constant factor of the lower bound.

The CDDR algorithm adapts the replication scheme to changing read-write patterns. For example, if for a period of time, the only access to an object o in the distributed system consists of reads and writes from a particular site s, then the CDDR algorithm will move the replication scheme of o to include s only. In contrast, if for a period of time, all the sites in the distributed system issue only reads (and not writes) of o, then the CDDR algorithm will change the replication scheme of o to include all the sites in the distributed system. It is important to note that these changes occur in a distributed fashion, as a result of statistics collected locally at each site.

The algorithm is based on two techniques. One is the existence of a primary copy (or site), and the second is the 'read one write all available' method.

The CDDR algorithm is performed for each logical object independently of any other logical object. In other words, different objects may have different replication schemes and different primary sites.

We conjecture that the CDDR algorithm has the following property. When the read-write pattern becomes regular (e.g., site 1 issues 2 reads and 1 write per time unit, site 2 issues 3 reads and 2 writes per time unit, etc.), then the replication scheme becomes fixed, and the fixed replication scheme is optimal for the read-write pattern. This is a different (than competitiveness) measure of performance for a dynamic replication algorithm, and it was used in [2]. We intend to prove this conjecture.

This CDDR algorithm has not taken the I/O cost of data replication into account. However we have developed a competitive algorithm that optimizes a cost function which combines both the I/O and the communication cost. This algorithm is beyond the scope of this paper.

References

[1] M. Manasse, L.A. McGeoch, and D.Sleator, *Competitive algorithms for online problems*, Proc. 20th ACM STOC, page 322-333, ACM 1988

[2] Ouri Wolfson and Sushil Jajodia, *Distributed Algorithms for Dynamic Replication of Data*, Proc. of ACM-PODS, 1992

[3] Ouri Wolfson and Sushil Jajodia, *An Algorithm for Dynamic Data Distribution*, Proceedings of WMRD, 1992

[4] A. Fiat, R, Karp, M.Luby, L.A. McGeoch, D.Sleator, N.E. Yong, *Competitive paging algorithms*, Journal of Algorithms, 12, pages 685-699, 1991

[5] N. Goodman, D. Skeen, A. Chan, U.Dayal, S. Fox, D. Ries, *A recovery algorithm for a distributed database system*, Proc. 2nd ACM SIGACT-SIGMOD, Symp. Database System, Atlanta, GA, March 1983, pp 8-15

[6] P. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency control and recovery in database systems*, Addison-Wesley (1987)

[7] C. Papadimitriou, *The serializability of concurrent updates*, Journal of the ACM, 26(4), pp. 631-653

[8] Michael J. Carey, and Miron Livny, *Distributed concurrency control performance: A study of algorithms, distribution and replication*, Proc. of the 14th VLDB Conf. Los Angeles, CA, 1988

[9] Hector Garcia-Molina, and Robert K. Abbott, *Reliable distributed management*, Proc. of the IEEE, Vol. 75, NO. 5, May 1987

[10] P.A. Bernstein, and N. Goodman, *An algorithm for concurrency control and recovery in replicated distributed databases*, ACM-TODS, Vol. 9, No. 4, December 1984, Pages 596-615

[11] O. Wolfson and A. Milo, *The Multicast Policy and Its Relationship to Replicated Data Placement*, ACM TODS, 16 (1), 1991.

[12] L. W. Dowdy and D. V. Foster, *Comparative Models of the File Assignment Problem*, ACM Computing Surveys, 14 (2), 1982.

[13] R. Alonso, D. Barbara, H. Garcia Molina, *Quasi-copies: Efficient data sharing for information retrieval systems*, Proc. of EDBT'88, LNCS 303, Springer Verlag.

[14] R. Alonso, D. Barbara, H. Garcia Molina, *Data caching issues in an information retrieval system*, ACM TODS, 15 (3), 1990.

[15] D. Barbara, H. Garcia Molina, *The case for controlled inconsistency in replicated data*, Proc. of the IEEE workshop on replicated data, 1990.

[16] R. Ladin, B. Liskov, L. Shrira, *A technique for constructing highly available distributed services*, Algorithmica 3, 1988.

[17] N. Krishnakumar and A. Bernstein, *Bounded ignorance in replicated systems*, Proc. of ACM-PODS '91.

[18] D. Sleator and R. Tarjan, *Amortized Efficiency of List Update and Paging Rules*, CACM 28(2), 1985.

[19] A. Karlin, M. Manasse, L. Rudolph, D. Sleator, *Competitive Snoopy Caching*, Algorithmica, 3 (1), 1988.

[20] Y. Bartal, A. Fiat, Y. Rabani, *Competitive Algorithms for Distributed Data Management*, 24th Annual ACM STOC, 5/92, Victoria, B.C. Canada.

APPENDIX: Proof of the Theorem

A.1 Lower Bound

We will firstly show the following *LB* algorithm is the lower bound for an arbitrary schedule. Then we use LB as a yardstick to measure the competitiveness of other algorithms.

Intuitively, *LB* performs as follows. For every read, LB reads from one copy, while for a write it does not write to every available current copies, instead it changes the replication scheme to a singleton and writes to the new replication scheme.

LB Algorithm

1. **For each write request**, the replication scheme shrinks to the singleton consisting of the writing site alone;

2. **For each read request**, it reads locally if it is in the replication scheme; otherwise it reads from the most recent writer, the replication scheme is then expanded to include this reader.

The following example demonstrates the LB algorithm.

Example 2: Suppose the network consists of three sites {1, 2, 3 }. Initially there is a single replica of the object residing at site 1. The request schedule is $r_2^1 r_3^2 w_3^3 r_1^4$, where the superscripts and the subscripts have the same meaning as in Example 1. Then by using LB the replication scheme changes as follows.

After request 1, site 2 joins the replication scheme, which costs 1; After request 2, site 3 joins the replication scheme, which costs 1, and the replication scheme consists of all sites; After request 3, the replication scheme shrinks to consist of site 3 (the writer) alone, site 1 and 2 exit, with 0 cost; After request 4, the replication scheme expands to include site 1 (the reader), which costs 1. i.e., the configured schedule is $\{1\}$ $r_2^1\{1,2\}$ $r_3^2\{1,2,3\}$ $w_3^3\{3\}$ $r_1^4\{1,3\}$, and the total cost is 3. □

We will show the algorithm LB is the best dynamic replication in this model for any request schedule as the following lemma 1.

Lemma 1: LB is cost-optimal.

Proof: Suppose the schedule is

$$\psi = R^{(0)} w^1 R^{(1)} w^2 \dots w^n R^{(n)}$$

We denote the schedule of the form $w^i R^{(i)}$ by σ_i. Since the incurred communication cost of this schedule for algorithm A is the sum of all the costs incurred by each request, thus

$$COST_A(\psi) = COST_A(R^{(0)}) + \sum_{i=1}^{n} COST_A(\sigma_i)$$

For requests $R^{(0)}$, LB costs the number of distinct readers of $R^{(0)}$ which are not in the replication scheme. That is obviously the most economic cost among all dynamic replication algorithms.

Thus it suffices to show that for each σ_i, the incurred communication cost for LB is also optimal.

Assume $\sigma = w_0 r_1 r_2 ... r_k$, where the subscripts denotes the site from which the write request w or read request r is issued. Then we need only to consider the schedule of the form σ. Let the request sequence σ together with the associated replication schemes be as follows

$$\alpha = w_0 X_0 r_1 X_1 ... r_k X_k$$

where w_0 is the *write* request issued by site 0, X_0 is the associated replication scheme after this write request, r_i is the *read* request issued by i and X_i is the associated replication scheme with this read. We assume that there are x distinct sites in the $(k+1)$ readers/writer.

In this varying replication scheme sequence, the object traverses from the *writer* 0 to the set X_0, from X_0 to $\{r_1\} \cup X_1$, from $\{r_1\} \cup X_1$ to $\{r_2\} \cup X_2$ and so on. Finally the edges used will connect all the sites in $Y = \{0, 1, ..., k\}$, and all the sites in X_i for $0 \le i \le k$. Hence these edges will form a connected (possibly multi-edged) subgraph say H. Let H' be the graph obtained from H by simply deleting all the multiple edges. Then H' is obviously a subgraph of the whole network G, and H' has at least $x-1$ edges since it connects x sites. Therefore the lower bound of the cost incurred for σ is $x-1$ using any algorithm A. We can see easily that the algorithm LB will cost exactly $x-1$ for σ, hence LB is cost-optimal. □

A.2 Competitiveness of CDDR(1)

Lemma 2: $CDDR(1)$ is 2-competitive under the constraint NBW.

Proof: Consider the request schedule

$$\psi = R^{(0)} w^1 R^{(1)} w^2 \dots w^n R^{(n)}$$

We know each read will simply expand the replication scheme to include the reader with the cost 1 (if the reader was not in R), and each write will simply write to every replica including the replica the writer is holding (the writer must had a priori read before this write and after any other write due to the NBW constraint) and then the others will exit, with the write cost ($\#\_of\_replicas - 1$). Then we see the cost for $R^{(i)}$ is the number of distinct readers other than the preceding writer in this sequence. The (i+1)st writer is in $R^{(i)}$ due to the NBW constraint, hence this *write* will cost at most (1+number of distinct readers other than the preceding writer in $R^{(i)} - 1$), where the first 1 is for the preceding writer. Therefore w^{i+1} costs at most as much as $R^{(i)}$ does. We see that $R^{(i)}$ will produce the same cost using $CDDR(1)$ as using LB, hence $CDDR(1)$ produces at most twice as much cost as LB does. i.e. $CDDR(1)$ is 2-competitive. □

A.3 Re-formulating the Cost Function

In order to prove this theorem, we introduce the following notation and analyze the cost incurred in the request sequence ψ for an algorithm A where A executes the 'Read One Write All' protocol. Let the request sequence be $\psi = o_{s_1}^1 o_{s_2}^2 ... o_{s_m}^m$ where s_i is the site from which the request $o_{s_i}^i$ is issued. For algorithm A, let the configured schedule is $R_A^0 o_{s_1}^1 R_A^1 o_{s_2}^2 ... R_A^{m-1} o_{s_m}^m R_A^m$. We define

$$c_i = \begin{cases} 0 & o_{s_i}^i \text{ is a read, } s_i \in R_A^{i-1} \\ 1 & o_{s_i}^i \text{ is a read, } s_i \notin R_A^{i-1} \\ |R_A^{i-1}| - 1 & o_{s_i}^i \text{ is a write, } s_i \in R_A^{i-1} \\ |R_A^{i-1}| & o_{s_i}^i \text{ is a write, } s_i \notin R_A^{i-1} \end{cases}$$

to be the cost of request $o_{s_i}^i$. The total incurred cost of ψ for A is $COST_A(\psi) = \sum_{i=1}^{m} c_i$.

For each site s, we define the associated *status sequence* of algorithm A for ψ is $t_0^s o_{s_1}^1 t_1^s o_{s_2}^2 ... o_{s_m}^m t_m^s$. We define the p-cost of a site s for the request $o_{s_i}^i$ of the algorithm A as follows

$$p_i^s(A) = \begin{cases} 0 & o_{s_i}^i \text{ is a read, } s_i \neq s \\ 0 & o_{s_i}^i \text{ is a read, } s_i = s,\ t_{i-1}^s = 1 \\ 1 & o_{s_i}^i \text{ is a read, } s_i = s,\ t_{i-1}^s = 0 \\ 0 & o_{s_i}^i \text{ is a write, } t_{i-1}^s = 0 \\ 0 & o_{s_i}^i \text{ is a write, } t_{i-1}^s = 1,\ s_i = s \\ 1 & o_{s_i}^i \text{ is a write, } t_{i-1}^s = 1,\ s_i \neq s \end{cases}$$

i.e. the p-cost of site s for the request $o_{s_i}^i$ is one only if (**1**) s is not in the replication scheme and $o_{s_i}^i$ is a read from s or (**2**) s is in the replication scheme and $o_{s_i}^i$ is an r-write of s. Otherwise its p-cost is zero. For the site s, the total p-cost is $PC_s(A) = \sum_{i=1}^{m} p_i^s(A)$. Notice that the p-cost of a site s depends on its own status and the request, it has nothing to do with the data replication scheme. Assume N is the number of sites in the network, then $c_i = \sum_{s=1}^{N} p_i^s$. The total incurred cost of ψ for A is

$$COST_A(\psi) = \sum_{i=1}^{m} c_i = \sum_{i=1}^{m}\sum_{s=1}^{N} p_i^s$$
$$= \sum_{s=1}^{N}\sum_{i=1}^{m} p_i^s = \sum_{s=1}^{N} PC_s(A)$$

To prove the theorem, we compare the p-cost of each site s for $CDDR(k)$ with that for $CDDR(1)$. we are going to show that the p-cost of each site for $CDDR(k)$ is at most k times as much as that for $CDDR(1)$, thus to conclude the theorem.

Assume by $CDDR(k)$, the site s's status will be reset n times as $t_1, t_2, ..., t_n$, the request sequence ψ is to be split by these statuses as subsequences as

$$t_0\psi_0 t_1\psi_1 ... t_i\psi_i t_{i+1} ... t_n\psi_n$$

where t_0 is the initial status of the site s before the request sequence ψ_0. We define this sequence to be the *action-status sequence* of site s, and each ψ_i $(0 \leq i \leq n)$ is called a *subsequence of the action-status sequence*. $\psi = \psi_0\psi_1...\psi_n$ and inside each subsequence ψ_i no *Action* of site s is taken. Then we have the following lemmas:

A.4 Auxiliary Results

Lemma 3: Given a sequence ψ, by using $CDDR(k)$ algorithm the site s's action-status sequence is

$$t_0\psi_0 t_1\psi_1 ... t_i\psi_i t_{i+1} ... t_n\psi_n$$

Then using algorithm $CDDR(1)$, site s's status will be t_{i+1} after request sequence ψ_i as using $CDDR(k)$ under constraint NBW.

Proof: Assume the last request in ψ_i is q. This q causes some *Action* of site s be taken, If this q is a read from s, the *Action* can only be a *Join* if s was not in the replication scheme, or a *Reset* if s was. If q is an r-write of s, the *Action* can be either a *Reset* if s was not in the replication scheme, or an *Exit* if s was. If q is a write from s, the *Action* can only be a *Switch* no matter s was in the replication scheme or not. q can not be a read from other site since other site's read request will not cause s's status reset. Thus we can verify this lemma in only in the following cases.

Case 1: q is a read from s, and s is a non data site, i.e. $t_i = 0$. Then t_{i+1} is set by the *Join Action* of $CDDR(k)$, hence equals 1. Using $CDDR(1)$, the status of s after its read is obviously one.

Case 2: q is a read from s, and s is a data site, i.e. $t_i = 1$. Then t_{i+1} is set by the *Reset Action* of $CDDR(k)$, hence equals 1. Using $CDDR(1)$, the status of s after q is 1 too.

Case 3: q is an r-write of s, and s is a non data site, i.e. $t_i = 0$. Then the primary site must have detected k r-writes and take the *Reset Action* of $CDDR(k)$, hence $t_{i+1} = 0$. Using $CDDR(1)$, every site (except the writer) will exit from the replication scheme, thus after q, s is not in the replication scheme either.

Case 4: q is an r-write of s, and s is a data site, i.e. $t_i = 1$. Then s detects that it has k r-writes already and takes the *Exit Action* (or *Switch Action* if s is the primary site). Hence $t_{i+1} = 0$. Using $CDDR(1)$, s will not be in the replication scheme either after request q.

Case 5: q is a write from s, s becomes the new primary site by the *Switch Action* taken by the old primary site, then $t_{i+1} = 1$. By the constraint NBW we know that before this q, site s must have issued a read r for the priori reference, and between the r and this q there is no other write request, which means if we use $CDDR(1)$, site s will join the replication scheme after request r and won't exit through request q. Therefore s will be in the replication scheme after request q. $\square$

From this lemma we see that for any sequence ψ, we can divide it into subsequences of the action-status sequence by using $CDDR(k)$. And at the beginning of each such subsequence ψ_i, $CDDR(k)$ will set the site s in the same status as $CDDR(1)$ does. Suppose in ψ_i there are a_i r-writes, b_i reads. In the following lemmas, we are going to compare the p-cost of the subsequence of the action-status sequence between $CDDR(k)$ and $CDDR(1)$.

Lemma 4: Let ψ_i be a subsequence of the action-status sequence $t_0\psi_0 t_1 ... t_n\psi_n$ of site s by using

$CDDR(k)$ $(i \neq n)$. If $t_i = t_{i+1} = 0$, then for ψ_i, $PC_s(CDDR(k)) \leq k \cdot PC_s(CDDR(1))$ under constraint NBW.

Proof: Upon finishing the request sequence ψ_i using $CDDR(k)$, the Action taken to site s must be a *Reset*. Assume ψ_i has a_i r-writes and b_i reads from s, then $a_i = k$, and $b_i < k$. $PC_s(CDDR(k)) = b_i$. If $b_i = 0$, $PC_s(CDDR(k)) = PC_s(CDDR(1)) = 0$, the lemma follows. If $b_i \neq 0$, then the first read of s in ψ_i cost one using $CDDR(1)$ since the site s is not in the replication scheme before ψ_i by lemma 2. Hence $PC_s(CDDR(k)) < k \leq k \cdot PC_s(CDDR(1))$ under constraint NBW. □

Lemma 5: Let ψ_i be a subsequence of the action-status sequence $t_0\psi_0 t_1 ... t_n \psi_n$ of site s by using $CDDR(k)$ $(i \neq n)$. If $t_i = t_{i+1} = 1$, then for ψ_i, $PC_s(CDDR(k)) \leq k \cdot PC_s(CDDR(1))$ under constraint NBW.

Proof: Upon finishing the request sequence ψ_i using $CDDR(k)$, the Action taken to site s must be a *Reset*. Assume ψ_i has a_i r-writes and b_i reads from s, then $b_i = k$, and $a_i < k$. $PC_s(CDDR(k)) = a_i$. If $a_i = 0$, the lemma is obviously true. If $a_i \neq 0$, by lemma 3 we see that the first r-write among a_i r-writes of ψ_i costs 1 by using $CDDR(1)$. Hence $PC_s(CDDR(k)) < k \leq k \cdot PC_s(CDDR(1))$. □

Lemma 6: Let ψ_i be a subsequence of the action-status sequence $t_0\psi_0 t_1 ... t_n \psi_n$ of site s by using $CDDR(k)$ $(i \neq n)$. If $t_i = 0, t_{i+1} = 1$, then for ψ_i, $PC_s(CDDR(k)) \leq k \cdot PC_s(CDDR(1))$ under constraint NBW.

Proof: Upon finishing the request sequence ψ_i using $CDDR(k)$, the Action taken to site s must be either a *Join* or a *Switch*. Denote the last request of ψ_i by q.

If the Action is a *Join*, q must be a read from s. In this case, there are k such reads in ψ_i, and before ψ_i s is not in the replication scheme using either $CDDR(k)$ or $CDDR(1)$ by lemma 3. We see the p-cost of site s for ψ_i is exactly k using $CDDR(k)$. The p-cost of using $CDDR(1)$ is at least one, since the the first read will cost that much. The lemma follows.

If the Action is a *Switch*, q must be a write from s, and s is the new primary site and it is not in the replication scheme before ψ_i. Then s must have issued a read in ψ_i due to constraint NBW and the request immediately preceding ψ_i must be either an r-write request of s or no request (if $i = 0$). We see that the incurred p-cost of s in ψ_i is less than k using $CDDR(k)$, while the incurred cost of s is at least one for that read using $CDDR(1)$. This completes the proof of this lemma. □

Lemma 7: Let ψ_i be a subsequence of the action-status sequence $t_0\psi_0 t_1 ... t_n \psi_n$ of site s by using $CDDR(k)$ $(i \neq n)$. If $t_i = 1, t_{i+1} = 0$, then for ψ_i, $PC_s(CDDR(k)) \leq k \cdot PC_s(CDDR(1))$ under constraint NBW.

Proof: The last request q of ψ_i must be an r-write of s. In this case, there must be k r-writes of s in ψ_i, and s is in the replication scheme using either $CDDR(k)$ or $CDDR(1)$ before ψ_i by lemma 3. We see the p-cost of site s for ψ_i is exactly k using $CDDR(k)$, while the p-cost of that using $CDDR(1)$ is at least one for the first r-write. The lemma follows. □

Lemma 8: Let ψ_n be the last subsequence of the action-status sequence $t_0\psi_0 t_1 .. t_n \psi_n$ of site s by using $CDDR(k)$. Then for ψ_n, $PC_s(CDDR(k)) \leq k \cdot PC_s(CDDR(1))$ under constraint NBW.

Proof: Assume that ψ_n has a r-writes and b reads. Since no Action is taken upon finishing ψ_n, we have $a < k, b < k$. The status of site s before ψ_n is t_n.

Case 1: $t_n = 0$. For ψ_n, $PC_s(CDDR(k)) = b$. If $b = 0$, the lemma is obviously true. If $b \neq 0$, the first read among b reads costs 1 using $CDDR(1)$ since site s is not in the replication scheme before ψ_n by lemma 3. The lemma follows.

Case 2: $t_n = 1$. For ψ_n, $PC_s(CDDR(k)) = a$. If $a = 0$, the lemma is obviously true. If $a \neq 0$, the first r-write among a r-writes costs one using $CDDR(1)$ since site s is in the replication scheme before ψ_n by lemma 3. This completes the proof of the lemma. □

A.5 Proof of the Theorem

Proof of Theorem: Assume by using $CDDR(k)$ algorithm the site s's action-status sequence is

$$t_0\psi_0 t_1 \psi_1 ... t_i \psi_i t_{i+1} ... t_n \psi_n$$

For each subsequence ψ_i^s $(0 \leq i \leq n)$, we see from lemma 4, 5, 6, 7, and 8 that the p-cost of site s satisfies

$$PC_s(CDDR(k)) \leq k \cdot PC_s(CDDR(1))$$

therefore for the whole sequence ψ, the p-cost satisfies $PC_s(CDDR(k)) \leq k \cdot PC_s(CDDR(1))$. Therefore the total incurred cost satisfies

$$COST_{CDDR(k)}(\psi) = \sum_{s=1}^{N} PC_s(CDDR(k)) \leq$$

$$\sum_{s=1}^{N} k \cdot PC_s(CDDR(1)) = k \cdot COST_{CDDR(1)}(\psi)$$

From lemma 8, we know $CDDR(1)$ is 2-competitive under constraint NBW, thus $CDDR(k)$ is $2k$-competitive under constraint NBW. □