ACHEIVING HIGH AVAILABILITY IN A REPLICATED FILE SYSTEM BY
DYNAMICALLY ORDERING TRANSACTIONS

Alain Sandoz
Ecole Polytechnique Federale de Lausanne
Departement d'Informatique
CH-1015 Lausanne, Switzerland

Abstract
This paper presents a new pessimistic consistency-control algorithm. So far, it achieves the highest availability of objects in a partitioned, atomic-transaction based, distributed system. The principle of the algorithm is to dynamically order transactions in the global serialization graph of the system. This order is built by associating numbers to transactions in such a way that these numbers decrease along any path in the graph. Transaction numbers correspond to an arbitrary ordering of replicated objects. The method generalizes to all existing pessimistic algorithms to achieve better availability. The work was developed in the context of distributed file systems.

1. Introduction
The problem of managing replicated data in a distributed system is surveyed extensively in the literature. In this context, two conflicting goals must be met: maintaining global consistency and achieving performance through high availability of objects in the system. When using atomic transactions, the former goal is usually met by ensuring one-copy serializability (1-SR) of transactions [Bernstein 83, 87]. The most important problem with replicated objects is the event of a partition in the network underlying the system.

The distributed application considered in the system will influence the choice of the method used to achieve robustness. Most methods are designed for distributed database systems. In this paper, we present an algorithm conceived to ensure consistency and high availability in a replicated file system: the BOSTRYCHE system under development at the Swiss Federal Institute of Technology in Lausanne. In a distributed file system, several particular constraints must be considered at design time. First, consistency must be ensured, what we will consider equivalent to achieving 1-SR of transactions. Second, undoing of committed transactions should not be allowed. This is due in part to the highly interactive mode of execution of very long transactions, and to the proprietary links between users and objects in a file system. This prevents the use of optimistic consistency control algorithms. Third, availability of objects (files) in the system must be high, as they represent in general unique instances of non-redundant data. Fourth, availability of certain special objects (compilers, editors,...), which will be replicated, must be guaranteed in any circumstance.

By dynamically ordering transactions of different partitions in the global one-copy serialization graph, our algorithm respects these four constraints, meeting both conflicting goals described above. We do this by introducing an ordered numbering of replicated objects. These numbers are used to order transactions depending on partial updates of objects, in such a way that transaction numbers decrease along any path in the graph, thus preventing cycles.

The method generalizes to all pessimistic protocols, always achieving better availability. It is syntactic pessimistic in the sense that no knowledge of the particular type of transactions in the system is used in the algorithm, and that once a transaction is committed, it is never undone. However, we can use some knowledge of the underlying application to enhance performance and availability, making it very nearly a semantic approach. The main result of the algorithm is to allow read-access of replicated objects in most partitions of a failed network, without delaying commitment of transactions, while restricting write-access to at most one partition per replicated object. The immediate availability of replicated objects achieved in this way is almost comparable to that allowed by optimistic methods. We do not however claim that our method performs better than optimistic algorithms. Performance of an access strategy is measured in terms of the transaction success rate. To compare optimistic versus pessimistic strategies is relevant only if committed transaction roll-back is possible. Better performance of one method over the other then depends on the frequency of failures in the system. We do not consider these subjects here.

The structure of the paper is the following. After briefly referring related work, we describe in section 3 our model of the distributed system and the transaction manager. We also introduce replicated objects and the basic access strategy used to illustrate our method. In section 4 we describe both the access algorithm, giving a proof of its correctness, and the recovery algorithm. Section 5 discusses the generalization of the method to other pessimistic algorithms and several related issues.

2. Related work
Much work has been done on consistency in distributed systems. A thorough survey of existing consistency-control methods in replicated systems was given in [Davidson 85]. In that paper, the classification of optimistic, pessimistic, syntactic and semantic algorithms is defined. More recently the concept of quasi-partitioning [Lilien 88] was introduced to model replicated systems using slow back-up links when communication failures occur. We will refer extensively to [Eager 83] in which the notion of maximal partial operability was introduced along with the missing-writes algorithm we use below. We also refer to [Jajodia 89] which formally defines pessimistic algorithms and proves an important result on their correctness.
In [Coan 86], an upper bound on availability in a partitioned network is given under the assumptions of total replication and uniformity of system workload, as defined in that paper. We make neither of these. In particular, objects in our system can be partially replicated, or even not at all. Our method allows to dynamically group transactions in ordered classes in the serialization graph. This has been done statically in the semantic conflict class analysis method [Wright 83], which is the origin of the class-related terminology used in the rest of the paper.

3. The model
We define a distributed system as a set of sites (or nodes) each containing a processing unit, primary storage, and stable secondary storage. Initially, all nodes are linked through a connected network. Failures in the system are site failures or link failures leading to communication breakdowns between sites. We consider these different types of failures as indistinguishable. A node unable to reach some correspondent does not know whether or not the latter node is down. Some failures can lead to a network partition, breaking up the set of nodes into two or more subsets unable to communicate with each other. Moreover, we do not suppose the existence of a mechanism enabling a node to know the state of the system at any time. A site learns of a failure when it is unable to reach its correspondent(s). Our model of site failure corresponds to "crash-failure" [Neiger 88], which excludes for example malicious sites.

Our model of transactions and transaction managers is very simple. A transaction Ti is initiated at a site called its coordinator or home site (noted h(Ti)). It executes read and write operations on the objects in the system. These objects are represented by one or several physical copies that can be local to the home site or stored at remote nodes. Transaction operations are executed on these copies. A read (write) operation by a transaction Ti on an object x will be noted r(x) (resp. w(x)). The objects read (written) by a transaction form its read-set (write-set). Transactions can have only two issues: abort or commit. Once a transaction is committed, its write operations have a permanent effect in the system. We do not consider precommit operations on objects or sets of objects.

Concurrency control is enforced by a standard two-phase lock algorithm, ensuring that transactions are consistently scheduled [Eswaran 76]. The termination algorithm is the well-known two-phase commit algorithm. This ensures that transactions have the same issue on every site they visit. If an object in the read- or write-sets of a transaction Ti is not available, the transaction is aborted altogether.

Objects in the system can be simple or replicated. In the latter case, several physical copies xi of a logical object X are stored at different sites throughout the network. To illustrate our method to enhance availability, we have chosen, as basic access-control strategy, the missing-writes algorithm presented in [Eager 83]. A generalization to the whole class of pessimistic algorithms is given in section 5. The missing-writes algorithm runs in two possible modes: normal and failure. In normal mode, updates to replicated objects are carried out on every physical copy of the object (i.e. the strategy is write-all, read-one). In the case of a network partition or of a site failure at the location of a copy, the update might not be possible on every copy. The algorithm then runs in failure mode: updates on replicated objects can only be made if a write quorum of copies is present in the partition. In this case, mutual consistency of the copies in the system is no more guaranteed because of missing updates. After commitment of the requesting transaction Ti, missing-update information is posted along to every site running a transaction depending on the output of Ti. After repair of the partition, sites containing a copy which was not updated, and receiving missing-update information, can update the copy before proceeding with new transactions. The quorums chosen for our algorithm are simply majority quorums of copies. With the missing-writes method, minority reads are allowed for read-only1 transactions, which can be serialized "in the past". However, the algorithm basically prohibits a transaction which has read a replicated object for which it has no quorum from making an update on any other object. Our method allows transactions reading an object without a quorum to execute and commit in most partitions.

4. Accessing replicated data
In this section we describe our enhanced missing-writes algorithm. Any access-control algorithm for replicated data has two parts: the adaptation protocol and the recovery protocol. The first part is used to control access to replicated data and to enforce global consistency in the system. The second part, the recovery protocol, is used when partitions merge. The protocol must ensure mutual consistency between the copies of replicated objects in the new partition.

We first recall some basic definitions and results on one-copy serializability. We then illustrate the ideas behind our modified access protocol. The access algorithm is described along with the proof of its correctness. Finally we present a complete example of execution and make some comments concerning the recovery algorithm.

4.1. One-copy serializability
One copy serializability (noted 1-SR) [Bernstein 83,87] formalizes the user's intuitive feeling of correctness in a system supporting replicated objects. For the user, transactions are correct, first if each one behaves right individually, and second if the concurrent execution of transactions is equivalent to some serial execution of the same transactions on objects represented by only one copy.

The first condition depends on the semantics of the underlying application, and we will always consider it as fulfilled in what follows. The second condition means, in particular, that any transaction Ti, immediately reading an object updated by another transaction Tj, must precede transactions updating that object after Tj. This leads to the notion of one-copy serialization graph (1-SG) of a set of transactions and to the important result that acyclic 1-SG's characterize 1-SR transactions.

More precisely, we define a serialization graph (SG) on a set o=[T1,...,Tn] of transactions in a (partially) replicated system, as SG(o)=(o,E) where E is a set of directed edges. An edge (Ti,Tj) exists in E iff there are two conflicting physical operations, opi(x) executed by Ti and opj(x) executed by Tj, on a copy of an object X in the system, such that opi(x)
precedes op(x) (two operations conflict if they access the same physical object and one at least is a write).

An example of a set of transactions and its SG is given in Figure 1. The ordering relation among operations noted “<” depicts the partial order in which these operations are executed. The set of all operations in σ, partially ordered by this relation such that any pair of conflicting operations is related, is a journal on σ, noted J(σ).

\[
\begin{align*}
\text{on}(T_0,T_1,T_2,T_3) \\
T_0 &= \text{mod}(x_0,x_0) \\
T_1 &= \text{mod}(x_0,x_0) \\
T_2 &= \text{mod}(x_0,x_0) \\
T_3 &= \text{mod}(x_0,x_0)
\end{align*}
\]

\[
J(σ) = w_0 \rightarrow n_1 \rightarrow w_1 \rightarrow r_1 \rightarrow w_2
\]

SG(σ) = \[
T_2 \rightarrow T_1 \rightarrow T_0
\]

Fig. 1. Journal and serialization graph of a set of transactions

A one-copy serialization graph on σ is defined in the following way: 1-SG(σ) is an extension of SG(σ) in which enough edges are added to translate the serialization condition over transactions, i.e.

1. for every replicated object X in the system, 1-SG induces a total order < on the transactions writing X (note that for simple objects this follows from the two-phased concurrency control between transactions);
2. if Ti, Tj, and Tk are three transactions such that Ti < Tj < Tk (i.e. Tk updates X after Ti);

\[
\text{a)} \quad \text{there exists a copy } x_i \text{ of an object } X \text{ in the system such that } w_i(x_i) \rightarrow r_j(x_j);
\]

\[
\text{b)} \quad \text{there is no transaction } T_k \text{ such that } w_i(x_i) \rightarrow w_k(x_k) \rightarrow r_j(x_j);
\]

\[
\text{c)} \quad T_i < T_k \text{ (i.e. } T_k \text{ updates } X \text{ after } T_i)\]

then there is a path in 1-SG from Tj to Tk.

Condition a) and b) are commonly referred to as the condition “Tj reads-X-from Tj”. Figure 2 is an example of a 1-SG for the transactions of Figure 1. In this figure, edge (T1,T2) is added to the SG to ensure existence of a total order of writes on X. Edge (T3,T4) is added to ensure condition (2) above, since T3 reads-Y-from T0 and T2 updates Y, i.e. T0 < T2. The important result is that if there exists an acyclic 1-SG(σ), then σ is one-copy serializable, i.e. execution of the transactions in σ is correct.

4.2. Idea of the algorithm

Consider the situation where a set σ = \{Tnm\} of transactions has executed in a partitioned system. We note Pi the partitions of the network and Tm a transaction executing in Pi. We also consider a fictive transaction T0 writing every object in the system before the network partitions. Suppose these transactions have all committed before the repair of the partition. Then any traditional pessimistic adaptation protocol, in particular the missing-writes algorithm, ensures that there is no path in SG(σ) between two transactions Ta and Tb such that TmT0. Since it also ensures that the subgraph of SG(σ) built on transactions proper to any partition Pi (called the Pi-subgraph) is 1-SG and acyclic, the global serialization graph is also 1-SG and acyclic [Bernstein 83,87], and thus σ is 1-SR.

Suppose we relax the access rules to tolerate reads on replicated objects in partitions not containing a quorum for these objects. This introduces edges between partition-subgraphs of 1-SG’s of σ to ensure condition (2) of paragraph 4.1. More precisely, suppose transaction T3 reads a copy of an object X and Tα (partially) updates X (or). This means that update of X was incomplete in Pi due to the partition. We then have that T3 reads-X-from T0 and T0 < Tα, so that we might have to introduce one or several edges in SG(σ) to ensure existence of a path from T3 to Tα in SG(σ). We do this by adding the edge (T3,Tα) to SG(σ). This edge will be called a serialization edge, noted se.

The idea of the algorithm is quite simple. We associate with each transaction T of σ a number t cn(T), called T’s transaction class number, such that the following two conditions hold:

1. along any path in 1-SG(σ), class numbers associated with transactions decrease (possibly not strictly between neighboring vertices);
2. the class numbers associated with the starting vertex Ti and ending vertex Tj of any path having at least two serialization edges are different, i.e.

\[
t_{cn}(T_i) < t_{cn}(T_j)
\]

Since any cycle in 1-SG(σ) should follow at least two se’s between partition-subgraphs of SG(σ), these conditions prevent the existence of such a cycle in 1-SG(σ). We thus ensure correct execution of transactions in σ.

For the purpose of ordering transactions that run in different partitions, we introduce an arbitrary total ordering over the set of replicated objects and a numbering of these objects compatible with this order. We call the number associated with an object X its object class number, noted o cn(X). We first describe the way in which these numbers are used and give a small example. In section 4.3 the algorithm is formally stated and shown to enforce conditions (1) and (2) above.

Transaction class numbers are defined in the following way. First, the class number of a transaction must always be smaller or equal to t cn’s of transactions in its partition on which it depends ("<" relation). Second, we impose for t cn(T) to be smaller than the object class number of any object that T incompletely updates. So, if T runs in failure mode, the final value of t cn(T) will be determined by the class numbers of objects updated incompletely by T. More precisely, if a replicated object X is updated by T, not all copies of X being in h(T)’s partition, then t cn(T) ≤ o cn(X).

Transaction class numbers are thus defined recursively as follows:

\[
t_{cn}(T) = \min(t_{cn}(T_i) + t_{cn}(T_j) - t_{cn}(X), X_i \in \text{write-set}(T) \text{ and update of } X_j \text{ by } T \text{ is incomplete})
\]

Transaction class number of To is set to ∞.
The read-access rule then states that a transaction T cannot read a replicated object X for which T has no quorum if $o-cn(X) > t-cn(T)$. In particular if either $o-cn(X) > t-cn(T)$ and $T ightarrow T$ in $SG(o)$, or if there exists a partially updated object $X_j$ in write-set(T) and $o-cn(X) > o-cn(X_j)$, then T must be aborted. This condition must be verified independently of the order of reads and writes by T and must be evaluated at commit-time if T has incompletely updated a replicated object after having made a no-quorum read of some other object.

To illustrate these points, consider Figure 3, in which a set of transactions $T_m$ is executing in a network partitioned into three components. Transaction $T_a$ executes in partition number i. No partition has all the copies of objects X, Y, and Z, but each partition holds a majority of copies of one of these objects (X for the first, Y for the second, and Z for the third). Replicated objects in the system are ordered such that $o-cn(X) < o-cn(Y) < o-cn(Z)$.

Consider first the no-quorum read of Z by $T_1$:

1. $T_1$ incompletely writes X, so $t-cn(T_1) < o-cn(X)$;
2. $T_1$ depends on $T_2$, so $t-cn(T_1) \leq t-cn(T_2)$;
3. $T_1$ no-quorum read of Z by $T_1$ is allowed only if $o-cn(Z) \leq t-cn(T_1)$; however, this is impossible, since $o-cn(X) < o-cn(Z)$, and by (1) and (2), $t-cn(T_1) \leq o-cn(X)$; thus transaction $T_1$ must be aborted.

![Fig. 3. Example of a cycle in the one copy serialization graph, transaction $T_1$ is aborted.](image)

Consider now the no-quorum read of X by $T_2$:

1. $T_2$ incompletely writes Y, so $t-cn(T_2) = o-cn(Y)$;
2. $T_2$ depends on $T_2$, so $t-cn(T_2) \leq t-cn(T_2)$;
3. no-quorum read of X by $T_2$ is allowed if $o-cn(X) \leq t-cn(T_2)$, which is possible since $o-cn(X) < o-cn(Y)$;

The same is true in the third partition where $T_3$ incompletely updates Z and $T_3$ makes a no-quorum read on Y. In the situation of Figure 3, uncontrolled execution of minority-reads would lead to the construction of the dotted serialization graph, which are part of a cycle. However, because $o-cn(X) < o-cn(Z)$, the rules stated above are violated for transaction $T_3$, so $T_3$ is aborted, leading to correct execution; the remaining $se$'s are no longer part of a cycle. Note that neither the global serialization graph, nor any part of it, is constructed by the algorithm. Only local properties of the graph are enforced with the sole knowledge of the ordering of objects.

It is also important to notice that, in our algorithm, the write-access rule for replicated objects is unchanged compared to the basic missing-writes protocol, and the read-access rule is weaker (read-access remains unrestricted if T has a quorum for X). So this protocol provides better availability of objects, which we will see later on a complete example. In the next paragraph we formally state the access-rules along with the adaptation protocol, and prove that they ensure conditions (1) and (2) given above on transaction class numbers.

### 4.3. Adaptation protocol

Define the domain of a transaction T (noted $d(T)$) as the set of sites on which exists any accessed physical copy of an object in the read- or write-sets of a transaction T. This domain is included in the partition which contains the home-site $h(T)$. We first introduce a few variables that are used in the algorithm.

The maximal class number of a site s, noted $s_mcn(s)$ is a variable proper to the transaction manager of each site, and is initially set to $\infty$ on every site. It is used to determine $t-cn(T)$ of transactions T visiting s. This is done in the following manner: to ensure that $t-cn(Ta) < t-cn(Tb)$ in $SG(o)$, we first impose $t-cn(Ta) = \min(s_mcn(s))$, $se(T)$). Second, when transaction T is committed, $s_mcn(s)$ is set to $t-cn(T)$ on all sites s in $d(T)$. In this way, we keep trace of older transactions on which a transaction T might depend, and so enforce the condition stated above that transaction class numbers decrease along any path in partition-subgraphs of the global 1-SG.

Transaction T's temporary class number, noted $t_mcn(T)$ is a variable local to $h(T)$'s transaction manager, and set to $\infty$ when T is initiated. The value of $t-cn(T)$ is an upper bound for the final value of $t-cn(T)$, and is always smaller than $\min(o-cn(X))$. T's partially updated X). In this way we ensure that $t-cn(T) \leq \min(o-cn(X), X \in write-set(T), X \in H(T))$.

The maximal access number of T, noted $t_mn(T)$ is also a variable local to $h(T)$, and is set to 0 when T is initiated. The value of $t_mn(T)$ is always equal to $\max(o-cn(X), X \in write-set(T), X \in H(T))$.

By making sure that $t_mn(T) \leq t-cn(T)$ is always true, we control read-access permission of T to objects without a quorum. As noted above, the rule states that T can read an object X iff either one of two conditions holds: T has a quorum for X or $o-cn(X) < t-cn(T)$. If neither of these conditions is met, then T is aborted. We now give the precise access rules.

**Read-access rule of a copy $x$ of a replicated object X on site s by transaction Ti:**
- if there is no quorum for X in the partition of s and $o-cn(X) > t-cn(Ti)$, then abort(Ti);
- if update of X is incomplete then $t-cn(Ti) = \min(t-cn(Ti), s_mcn(s));$
- if there is no quorum for X in the partition, then $t_mn(Ti) = \max(t_mn(Ti), o-cn(X));$
- if transaction-invariant $t_mn(Ti) \leq t-cn(Ti)$ is not respected, then abort(Ti);

**Write-access rule of a copy $x$ of a replicated object X on site s by transaction Ti:**
- if there is no quorum for X in the partition of s, then abort(Ti);
- if update of X is incomplete then $t-cn(Ti) = \min(t-cn(Ti), o-cn(X));$
- if transaction-invariant $t_mn(Ti) \leq t-cn(Ti)$ is not respected, then abort(Ti);

Note that, if the transaction-invariant $t_mn(Ti) \leq t-cn(Ti)$ is not respected, which leads to abort(Ti), then there must have been a read executed on an object for which Ti has no quorum.
This would always imply abortion of Ti in the basic missing-writes algorithm.

**Termination of transaction Ti:**
- first 2PC-phase: if \( t \cdot tcn(Ti) < t \) and any site of \( d(Ti) \) cannot be reached, then \( abort(Ti) \);
- upon commit of Ti on each site s of \( d(Ti) \), \( s \cdot mcn(s) = \epsilon \cdot tcn(Ti) \) (this is also the value of \( t \cdot cn(Ti) \)).

When an object X is read, a simple algorithm is generally used to determine which copy of X is accessed. Because of the last rule above, it might be possible to enhance later availability by carefully choosing the protocol in this direction.

Further study will be necessary to fully exploit this characteristic.

These steps of the algorithm ensure the read-access rule and the correct evaluation of transaction class numbers. Figure 3 illustrated the impossibility of a cycle in 1-SG(o) using the rules given above. Note that updating \( s \cdot mcn(s) \) to \( t \cdot tcn(Ti) \) only on sites where T made an update is not sufficient to ensure correctness. This is because we do not distinguish between read-write and write-read conflicts in the serialization graph. Further work will examine possible enhancements of the protocol in this direction.

**Proof of the algorithm:** We consider a set \( \{T_0, T_1, ..., T_n\} \) of transactions respecting the access rules to replicated objects stated previously. To is an initialization transaction writing every object, simple or replicated, before any failure. Different transactions are handled in different partitions of the network due to a partition. Construction of 1-SG(o) is described in section 4.2, so only acyclicity remains to be shown. We do this by proving that conditions (1) and (2) of section 4.2 are respected.

We proceed to prove that condition (1) is respected. For paths in 1-SG(o) it is straightforward because maximal class numbers of sites decrease monotonously and are always upper bounds for transaction class numbers of new visiting transactions. For paths containing transactions in different partitions, we must take into account serialization edges in 1-SG(o). Such an edge \((T_i, T_a)\) means that \(T_a\) incompletely updated an object X that \(T_i\) read. In particular, \(T_i\) did not have a quorum for X, since \(T_a\) had one to update the object; so \( o \cdot cn(X) < t \cdot cn(T_i) \). In the same way, since \(T_a\) incompletely updated \(X\), we have \( t \cdot cn(T_a) < o \cdot cn(X) \). Condition (1) follows immediately.

We now show that condition (2) is respected, i.e. that \( t \cdot cn's \) are different at the beginning and end of any path which has two serialization edges in 1-SG(o). Consider such a path \((u_1, v_2, ..., u_n)\), and note \( \epsilon = (u_1, v_2) \) its serialization edges. The first serialization edge (noted \(\epsilon\)) along the path means that transaction \(\epsilon\) read an object X that \(\epsilon+1\) incompletely updated. As a consequence, we have \( t \cdot cn(\epsilon) < o \cdot cn(X) \) for \(\epsilon+2\). The second \(\epsilon\) on the path starts in the same partition as \(\epsilon\), call it \(\epsilon\). Its meaning is that \(\epsilon\) didn't have a quorum for an object \(Y\) that \(\epsilon\) read and that \(\epsilon+1\) incompletely updated. So \( o \cdot cn(Y) < t \cdot cn(\epsilon) \). We now use the second \(\epsilon\) on the path, in a similar way to the previous case, to show that \(t \cdot cn(Y) < o \cdot cn(\epsilon)\), which fulfills condition (2). This completes the correctness proof of the adaptation protocol.

**4.4. Complete example**

We now give a complete example, comparing our protocol both with the classical missing-writes algorithm and with the optimistic approach. We consider a system composed of four sites \(s_1, s_2, s_3, s_4\) in two partitions \(P_1 = \{s_1, s_2\}\) and \(P_2 = \{s_3, s_4\}\). The system contains five replicated objects \(X, Y, Z, T, U\) associated with class numbers such that \( o \cdot cn(X) < o \cdot cn(Y) < o \cdot cn(Z) < o \cdot cn(U) \). Each object is represented by physical copies indexed by their location's site number and distributed among sites in the following way:

- \(x_1, y_1, z_1, t_1 \in s_1; \ y_2, z_2, t_2 \in s_2; \)
- \(x_3, y_3, z_3, t_3 \in s_3; \ y_4, z_4, t_4 \in s_4.\)

Thus \(P_1\) is a majority partition for \(YZ\) and \(T\), \(P_2\) for \(X\) and \(U\). Objects \(X\) and \(Z\) have copies in both partitions. Figure 4 shows this situation.

We consider transactions \(T_0, T_1, ..., T_6\) defined below:

- \(T_0\) is an initialization transaction writing all objects before the network parts;
- coordinator of transactions \(T_1, T_2, T_3\) is \(s_1;\)
- coordinator of transactions \(T_4, T_5, T_6\) is \(s_3;\)
- transactions execute sequentially on their home site;
- \(T_1 = <Y, Z>_w[Y, Z]>;\)
- \(T_2 = <Y, T>_w[Y, T]>;\)
- \(T_3 = <X, Z>_w[X, Z]>;\)
- \(T_4 = <U, U>_w[U]>;\)
- \(T_5 = <X, U>_w[X]>;\)
- \(T_6 = <X, Z>_w[X]>;\)

![Fig. 4. Partitions in the network.](image)

Logical updates are carried on to all physical copies whereas logical reads are performed on one arbitrary copy in the partition of the transaction's coordinator.

**Missing-writes Protocol:** the traditional pessimistic protocol needs a quorum for each object accessed in the partition, it runs as follows.

**In partition \(P_1\):**
- \(T_1 = <Y, Z>_w[Y, Z, X, T]>\)
- \(T_2 = <Y, T>_w[Y, T]>\)
- Transaction \(T_3\) is aborted because there is no quorum for \(X\) in this partition.

**In partition \(P_2\):**
- Transaction \(T_4\) is aborted because there is no quorum for \(X\) in this partition.
- \(T_5 = <X, U>_w[X]>\)
- Transaction \(T_6\) is aborted because there is no quorum for \(Z\) in this partition.
Class-number Protocol: access to a replicated object X by a transaction T is submitted to the presence of a quorum in the partition, except for read-access where \( o-cn(X) \leq t-cn(T) \).

In partition P1:
- \( T_1: <x[y, z], w[y, z], z, x> \)
  Update of Z is incomplete and update of Y is complete, so \( t-cn(T_1) \) is set to \( o-cn(Z) \). Maximal class numbers \( s-mcn(s) \) and \( s-mcn(s) \), initially set to \( \infty \), are now set to \( t-cn(T_1) \) and \( t-cn(T_1) \) is equal to \( o-cn(Z) \).
- \( T_2: <y, y, z, t, x> \)
  Quorums for Y and T are present in the partition so read- and write-access are free. Class number \( t-cn(T_2) \) is equal to \( t-cn(T_1) \), and \( s-mcn(s) \) and \( s-mcn(s) \) remain unchanged.

In partition P2:
- \( T_3: <x, z, z, y, z> \)
  Temporary class number \( t-cn(T_3) \) is set to \( o-cn(s) \). No-quorum read-access to X sets \( s-mcn(T_3) \) to \( o-cn(X) \) and is allowed since \( o-cn(X) = s-mcn(T_3) < t-cn(T_3) = o-cn(Z) \). So transaction class number \( t-cn(T_3) \) is also equal to \( t-cn(T_2) \), and \( s-mcn(s) \) and \( s-mcn(s) \) remain unchanged.
- \( T_4: <z, u, z, w, z> \)
  Transaction class number \( t-cn(T_4) \) is equal to \( t-cn(T_2) \), and \( s-mcn(s) \) and \( s-mcn(s) \) remain unchanged.

Optimistic Protocol: optimistic protocols do not bother with possible inconsistencies at execution time. However, such protocols log operations and rebuild the serialization graph after repair of the partition. If the graph contains a cycle, a minimum set of transactions is undone to restore correctness of execution. Figure 5 shows the serialization graph of transactions \( T_0 \), which contains a cycle. By undoing either transaction \( T_1 \) or \( T_4 \) we get an acyclic one-copy serialization graph and a correct run of the remaining transactions.

![Fig. 5. Serialization graph of \((T_0, ..., T_4)\).](image)

Our class number protocol provides better availability than the traditional missing-writes: transactions do not require to have a quorum for all objects they read-access. It is not necessary to abort transactions \( T_1 \) and \( T_4 \), while all transactions committed by the latter protocol are also committed by ours. As for transaction \( T_6 \), we see that, in this example, the success rate of the optimistic protocol cannot be made better than that of the class-number protocol. We do not however infer any general result on the performance of our method versus optimistic strategies from this example.

4.5. Recovery algorithm

Incomplete updates can be made in a partitioned system. Some sort of protocol must then be used after the repair of communications to ensure mutual consistency between copies of replicated objects in new partitions. In the missing-writes algorithm, incomplete-update information is forwarded through the system as transactions visit sites, in a manner that has been shown to meet this goal. We are however here concerned with proper recovery when class numbers have been used in the adaptation protocol.

Suppose a transaction T accesses an old copy of an object X in a partition containing mutually inconsistent copies of X. If missing-update information is forwarded to the site on which this copy resides, then the transaction must run in failure mode. We must consider two cases: first, if there is quorum for X in the partition, then all copies can be made mutually consistent, and because of the total ordering of writes on X in the system, any transaction reading X afterwards will be correctly serialized. Since there is a quorum for X in the partition, further access to X is unrestricted by the class-number protocol.

Second, if there is no quorum for X in the partition, it might still be possible to bring the accessed copy up to a more recent version. However, this is equivalent to a partial update of X, so the maximal class number of the site s on which this copy resides must be set at most to \( o-cn(X) \). If \( s-mcn(s) \) is already smaller than \( o-cn(X) \), then reading of X is not possible (writing is already impossible because of the absence of a quorum). The other conditions on the \( s-mcn(s) \) in d(T) remain also to be verified. If these conditions are true, then condition (1) on paths in SG remains true in the new partition. Different transactions in P accessing different copies of X remain serializable, because the missing-update information to the copies of X on which these transactions depend prevents them from reading copies which are not up to date. Thus new serialization edges do not come up in the partition-subgraph of P.

Now consider the problem of updating maximal class numbers on sites after the merge of partitions, in a way to achieve optimal availability of objects. This feature is not necessary, since \( s-mcn(s) \) are used only to allow no-quorum read-access of objects. These variables carry restricting information which never leads to inconsistencies even when the network evolves dynamically. However, if \( s-mcn(s) \) are not updated, they might carry information too restrictive in the new configuration, so transactions might be aborted that the class-number protocol should allow to commit.

For this reason, we introduce on every site a data-structure containing in decreasing order the class numbers of objects that were incompletely updated by transactions visiting that site. Suppose we find out that an object X is completely represented in a new partition, after the merging of several old ones. After establishing mutual consistency between copies, we broadcast this information to each site in the partition. These sites can then remove \( o-cn(X) \) from their class-number list and set back their maximal class number to its smallest value. This is because if \( o-cn(X) \) has been removed from the list of site s and \( o-cn(X) < s-mcn(s) \), any incoming serialization edge in the new partition-subgraph must have an origin T such that \( t-cn(T) > o-cn(X) \). So transactions T_i can run on s with \( o-cn(X) < t-cn(T) \).
5. Discussion

We now discuss some problems related to our method for accessing replicated objects, beginning with a few remarks on the use of object class numbers.

Object class numbers are used only to order transactions in the l-SG. They are not used for the scheduling of transactions, so that the global ordering of objects does not affect concurrency control, and has no effect on its cost.

When there is no partition in the network, site maximal class numbers remain set to $\infty$. Since all copies are always available, transactions updating a replicated object need never enforce the second step of the write-access rule of section 4.3, which is the only way to make them decrease. So when there is no failure, performance is not affected by the method.

If no partition has a quorum for an object X, no transaction will be able to update it. This implies that no site s will have $s_{mcn}(s) = o-cn(X)$, but there can exist sites for which $s_{mcn}(s) < o-cn(X)$. This means that read-access to X might be impossible in some (or all) partitions. So a good choice of object class numbers might be important to guarantee high availability of particular objects which are often read-accessed.

The next subsection is devoted to the generalization to pessimistic algorithms other than the missing-writes. Subsection 5.2 considers problems related to the semantics of underlying applications.

5.1. Generalization

In [Jajodia 89], the class of pessimistic algorithms is formally defined as the set of access-control protocols which (1) allow access to copies of a replicated object in at most one partition, called the majority group of sites for that object, (2) ensure identity of all copies of the replicated object in the majority group, and (3) ensure that successive majority groups for an object overlap when the configuration of the system evolves. Any pessimistic algorithm in this sense has been shown to be correct, i.e. transactions are one-copy serializable. In section 4 we considered such an algorithm, the missing-writes algorithm, as the basis for our method. The goal of this subsection is to generalize our approach to all pessimistic algorithms according to the definition above. In this sense, our method leads to higher availability, since it improves the availability provided by any pessimistic algorithm to which it is applied.

Generalization of the adaptation protocol is straightforward provided there exists a global ordering of replicated objects in the network. This is because, in the specification of conditions and rules of access for the enhanced adaptation protocol, we can replace everywhere "there is (is not) a quorum in the partition" by "the partition is (resp. is not) a majority group".

Adaptation of a pessimistic algorithm by using the class-number method does however have a price at recovery time. The problem when partitions merge is that any new partition can hold different copies of a replicated object. If the partition is a majority group for that object, then the classical pessimistic algorithm establishes mutual consistency of copies. However, if this is not the case, then mutual consistency will not be enforced. If ever a transaction in the new partition reads an old copy of an object X, the situation depicted in Figure 6 might occur. This figure shows the case where transaction $T_1$ incompletely updated object X in a partition overlapping $P_1$ before merging of $P_1$ and $P_2$. After the merge of partitions, transaction T which depends on an output of $T_1$, executes a non-majority read on an old copy of X which resides on one of the sites which were in $P_2$. This introduces the serialization edge $(T, T_1)$ and leads to inconsistency.

![Fig. 6. Problems arising when partitions merge.](image_url)

To cope with this problem, it is not sufficient for a transaction T to read the most recent copy of X in the partition of h(T). This would be nice because, when looking for a majority and finding out that there is none in the partition, h(T) would encounter all copies of the object and thus find the most recent one. However, after multiple partitions and merges in the network, the situation of Figure 6 could occur, for example if the copy read by T was the only one left in the partition of h(T).

A solution to this problem is to enforce, at any time and throughout the network, that in each partition all copies of a replicated object be identical. This might seem farfetched and quite expensive to do. However, by condition (2) of the definition above, any pessimistic algorithm already has a mechanism to enforce this in the majority partition. Thus, this mechanism need only be applied to every partition in the network, be it a majority partition or not for each of the replicated objects of which it has a copy, establishing mutual consistency of all copies with the most recent one in the partition. Like in section 4.5, the maximal class number of any site on which such an update takes place must also be updated, i.e. $s_{mcn}(s) \leq o-cn(X)$ for the partial update of an object X on a site s.

We summarize this important result in the following theorem: consider a distributed system in which the set RO of replicated objects is totally ordered by a function $o-cn: RO \rightarrow \mathbb{N}$, and a pessimistic access-control protocol in the sense defined above. Consider also a numbering of transactions by following the rules for transaction class numbers given in section 4.

**Theorem:** If we replace the access rule (1) of the definition of pessimistic protocols for reads by the rule: a transaction T can read an object X if and only if there is a majority partition for X or $o-cn(X) = o-cn(T)$; and if we replace rule (2) by the condition of mutual consistency in any partition; then transactions in the system are still one-copy serializable.

Proof of the theorem follows from section 4.3 and the remarks above.

5.2. Associating class numbers with objects

As was stated earlier, the ordering of replicated objects can be arbitrary. Moreover, as creation of unique identifiers in a distributed system is a well understood problem, the order need not be set once and for all. Creation and deletion of
replicated objects in the system are possible at any time. This means in particular that, as long as class numbers remain unique, reshuffling of the relative order of objects is possible if sufficient caution is taken.

However, the ordering of replicated objects in the system should be adapted to the particular underlying application. For example, in a distributed file system, read-only objects which are of primary importance (compilers, editors, special application software, ...) can be made to carry the smallest class numbers in the system. In this way, access to these objects is always possible everywhere. Following the same idea, replicated objects which are frequently updated should be given large class numbers, thus enabling non-majority reads on replicated objects with smaller class numbers.

As another example, consider a system in which updates are frequent on certain data, but are usually made on some particular site for each object (like personal bank accounts at the owner's local agency). Using a primary-copy strategy with primary copy at that site and a class number generated there by the usual <object number, site number, method for unique id's in distributed systems, most updates are possible and reads are likely to take place in the whole system. Carefully associating class numbers to objects and choosing the right access strategy thus enables us to access data nearly as well as with an optimistic approach, but with the advantage of having serializable transactions upon commit.

As a final remark on the access-control algorithm, we note that associating maximal class numbers to sites in the system is not the best choice to achieve high availability. A better choice would be to associate site numbers to objects, since different transactions can visit the same sites without interfering, which is less often the case when accessing the same objects. Enhancing availability in this way might however be at the expense of performance at recovery time, the number of objects being in general several orders of magnitude that of sites.

6. Conclusion

In this paper we have shown how to enhance availability of objects in a replicated system by adapting any pessimistic protocol as defined formally in [Jajodia 89]. This is done using an order on replicated objects to order transactions running in different partitions of a failed network. We believe that by carefully choosing the total order on replicated objects and the pessimistic access-control approach for a given application, this method can lead to performances close to those of optimistic protocols. In particular the method, which achieves high availability without undoing of transactions, is well suited to distributed file systems.

Acknowledgements

We wish to thank the referees for their useful comments. We would also like to thank A.Schiper for his thorough reading and many remarks during the process of writing the paper.

References