Fault-Tolerant Partial Replication in Large-Scale Database Systems

Pierre Sutra, Marc Shapiro

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Fault-Tolerant Partial Replication in Large-Scale Database Systems

Pierre Sutra Marc Shapiro

Université Paris VI and INRIA Rocquencourt, France

N° ????

Janvier 2008

Thème COM

Rapport de recherche

ISSN 0249-6399 ISRN INRIA/RR--????--FR+ENG

Fault-Tolerant Partial Replication in Large-Scale Database Systems

Pierre Sutra* Marc Shapiro
Université Paris VI and INRIA Rocquencourt, France

Thème COM — Systèmes communicants

Projet Regal

Rapport de recherche n° ???? — Janvier 2008 — 22 pages

Abstract: We investigate a decentralised approach to committing transactions in a replicated database, under partial replication. Previous protocols either reexecute transactions entirely and/or compute a total order of transactions. In contrast, ours applies update values, and orders only conflicting transactions. It results that transactions execute faster, and distributed databases commit in small committees. Both effects contribute to preserve scalability as the number of databases and transactions increase. Our algorithm ensures serializability, and is live and safe in spite of faults.

Key-words: data replication, large-scale, database systems

* LIP6, 104, ave. du Président Kennedy, 75016 Paris, France; mailto:pierre.sutra@lip6.fr

Unité de recherche INRIA Rocquencourt
Domaine de Voluceau, Rocquencourt, BP 105, 78153 Le Chesnay Cedex (France)
Téléphone : +33 1 39 63 55 11 — Télécopie : +33 1 39 63 53 30

Un algoorithme tolérant aux fautes pour la réplication de bases de données dans les systèmes large-échelle

Résumé :

Mots-clés : réplication, large-échelle, base de données,

1 Introduction

One of the major challenges in large-scale systems is designing a solution for non-trivial consistency problems e.g. file systems, collaborative environments, and databases. Recently some architectures have emerged to scale file systems up to thousands of nodes [12, 15, 4], but no practical solutions exist for database systems.

At the grid level however, the landscape is already populated [5, 11, 16], and among the solutions proposed, the ones based on group communication primitives are the most promising [20]. In this article we extend the group communication approach to large-scale systems.

Highlights of our protocol:

- We consider partial replication in a large-scale database system.
- Similarly to the DataBase State Machine [17], replicas do not reexecute transactions, but apply update values only; it preserves scalability [2].
- Contrary to DBSM, we do not compute a total order; instead for each datum x read or written by a transaction T, we total order multicast T to the replicas of x; different TO-multicast are *not* ordered.
- Every replicas of T maintains a graph containing dependencies between T and concurrent conflicting relations; a replica commits T when T is transitively closed in this graph.

The outline of the paper is the following. Section 2 introduces our model and assumptions. Section 3 presents our algorithm. We close in Section 4 after a survey of related work. An appendix follows the bibliography, and contains a proof of correctness.

2 System model and assumptions

We consider a finite set of asynchronous processes or *sites* Π, forming a distributed system. Sites may fail by crashing, and links between sites are asynchronous but quasi-reliable: they do not create messages, nor they duplicate them, and given two correct processes p and q, if p sends a message m to q, q eventually receives m.

Each site holds a database that we model as some finite set of *data items*. We left unspecify the granularity of a data item. In the relationnal model, it can be a column, a table, or even a whole relationnal database. Given a data item x, the replicas of x, denoted $replicas(x)$, is the subset of Π whose database contains x.

We base our algorithm on the three following primitives:

- *Uniform Reliable Multicast* takes as input a unique message m and a *single* group of sites $g \subseteq \Pi$. Uniform reliable multicast consists of the two primitives R-multicast(m) and R-deliver(m). With Uniform Reliable Multicast, all sites in g have the following guarantees:
  - Uniform Integrity: For every message m, every site in g performs R-deliver(m) at most once, and only if some site performed R-multicast(m) previously.
  - Validity: if a correct site in g performs R-multicast(m) then it eventually performs R-deliver(m).
  - Uniform Agreement: if a site in g performs R-deliver(m), then every correct sites in g eventually performs R-deliver(m).

  Uniform Reliable Multicast is solvable in an asynchronous systems with reliable links, and crash-prone sites.
- *Uniform Total Order Multicast* takes as input a unique message m and a single group of sites g. Uniform Total Order Multicast consists of the two primitives TO-multicast(m) and TO-deliver(m). This communication primitive ensures Uniform Integrity, Validity, Uniform Agreement and Uniform Total Order in g:
  - Uniform Total Order: if a site in g performs TO-deliver(m) and TO-deliver(m') in this order, then every site in g that performs TO-deliver(m') has performed previously TO-deliver(m).
- *Eventual Weak Leader Service* Given a group of sites g, a site $i \in g$ may call function *WLeader*(g). *WLeader*(g) returns a *weak leader* of g :
  - *WLeader*(g) $\in g$.
  - Let r be a run of Π such that a non-empty subset c of g is correct in r. It exists a site $i \in c$ and a time t such that for any subsequent calls of *WLeader*(g) on i after t, *WLeader*(g) returns i.

  This service is strictly weaker than the classical eventual leader service Ω [18], since we do not require that every correct site eventually outputs the same leader. An algorithm that returns to every process i itself, trivially implements the Eventual Weak Leader Service.

In the following we make two assumptions: during any run, **A1** for any datum x, at least one replica of x is correct, and **A2** Uniform Total Order Multicast is solvable in *replicas*(x).

2.1 Operations and locks

Clients of the system (not modeled), access data items by read and write operations. Each operation is uniquely identified, and accesses a single data item. A read operation is a singleton: the data item read, a write operation is a couple: the data item written, and the update value.

		lock held		
		R	*W*	*IW*
lock requested	*R*	1	0	0
	W	0	0	0
	IW	0	0	1

Table 1: Lock conflict table

When an operation accesses a data item on a site, it takes a lock. We consider the three following types of locks: read lock (R), write lock (W), and intention to write lock (IW).

Table 1 illustrates how locks conflict with each other; when an operation requests a lock to access a data item, if the lock is already taken and cannot be shared, the request is enqueued in a FIFO queue. In Table 1, 0 means that the request is enqueued, and 1 that the lock is granted.

Given an operation o, we denote:

- $item(o)$, the data item operation o accesses,
- $isRead(o)$ (resp. $isWrite(o)$) a boolean indicating whether o is a read (resp. a write),
- and $replicas(o) = replicas(item(o))$;

We say that two operations o and o' *conflict* if they access the same data item and one of them is a write:

$$conflict(o,o') = \begin{cases} item(o) = item(o') \\ isWrite(o) \vee isWrite(o') \end{cases}$$

2.2 Transactions

Clients group their operations by *transactions*. A transaction is a uniquely identified set of read and write operations.

Given a transaction T,

- for any operation $o \in T$, function $trans(o)$ returns T,
- $ro(T)$ (respectively $wo(T)$) denotes the subset of read (resp. write) operations,
- $item(T)$ denotes the set of data items transaction T accesses: $item(T) = \bigcup_{o \in T} item(o)$.
- and $replicas(T) = replicas(item(T))$.

Once a site i grants a lock to a transaction T, T holds it until i commits T, i aborts T, or we explicitely say that this lock is released.

3 The algorithm

As replicas execute transactions, it creates precedence relations between conflicting transactions. The serializability theory tell us that a correct database system maintains this relation acyclic ([3]).

One solution to this problem is given a transaction T, (i) to execute T on every replicas of T, (ii) to compute the transitive closure of the precedence relations linking T to concurrent conflicting transactions, and (iii) if a cycle appears, to abort at least one the transactions involved in this cycle.

Unfortunately as the number of replicas grows, sites may crash, and the network may experience congestions. Consequently to compute (ii) the replicas of T need to agree upon the set of concurrent transactions accessing $item(T)$.

Our solution is to use a TO-multicast protocol per data item.

3.1 Overview

To ease our presentation we consider in the following that a transaction executes initially on a single site. Section 3.11 generalizes our approach to the case where a tranasction initially executes on more than one site.

We structure our algorithm in five phases:

- In the *initial execution phase*, a transaction T executes at some site i.
- In the *submission phase*, i transmits T to $replicas(T)$.
- In the *certification phase*, a site j aborts T if T has read an outdated value. If T is not aborted, j computes all the precedence constraints linking T to transactions previously received at site j.
- In the *closure phase*, j complete its knowledge about precedence relations linking T to others transactions.
- Once T is closed at site j, the *commitment phase* takes place. j decides locally whether to commit or abort T. This decision is deterministic, and identical on every site replicating a data item written by T.

3.2 Initial execution phase

A site i executes a transaction T coming from a client according to the two-phases locking rule [3], but without applying write operations[1]. When site i reaches a commit statement, T is not committed,

[1] If T write a datum x then reads it, we suppose some internals to ensure that T sees a consistent value.

instead i releases T's read locks, converts T's write locks into intention to write locks, and proceeds to the submission phase.

3.3 Submission phase

In this phase i R-multicasts T to $replicas(T)$. When a site j receives T, j marks all T's operations as pending using variable *pending*. Then if it exists an operation $o \in pending$, such that $j = WLeader(replicas(o))$, j TO-multicasts o to $replicas(o)$.[2]

3.4 Certification phase

When a site i TO-delivers an operation o *for the first time*[3],i removes o from *pending*, and if o is a read, i certifies o.

To certify o, i considers any preceding write operations that conflicts with o. We say that a conflicting operation o' *precedes* o *at site* i, denoted $o' \rightarrow_i o$, if i TO-delivers o' then i TO-delivers o:

$$o' \rightarrow_i o = \begin{cases} \text{TO-deliver}_i(o') \prec \text{TO-deliver}_i(o) \\ conflict(o', o) \end{cases}$$

Where given two events e and e', we denote $e \prec e'$ the relation e *happens-before* e', and $\text{TO-deliver}_i(o')$ the event: "site i TO-delivers operation o'".

If such a conflicting operation o' exists, i aborts $trans(o)$.

If now o is a write, i gives an IW lock to o: function $forceWriteLock(o)$. If an operation o' holds a conflicting IW lock, o and o' share the lock (see Table 1); otherwise it means that $trans(o')$ is still executing at site i, and function $forceWriteLock(o)$ aborts it. (This operation prevents local deadlocks.)

3.5 Precedence graph

Our algorithm decides to commit or abort transactions, according to a *precedence graph*. A precedence graph G is a directed graph where each node is a transaction T, and each directed edge $T \rightarrow T'$, models a precedence relation between an operation of T, and a *write* operation of T':

$$T \rightarrow T' \stackrel{\triangle}{=} \exists (o, o') \in T \times T', \exists i \in \Pi, o' \rightarrow_i o$$

[2] If instead of this procedure, i TO-multicasts all the operations, then the system blocks if i crashes. We use a weak leader and a reliable multicast to preserve liveness.

[3] Recall that the leader is eventual, consequently i may receive o more than one time.

Algorithm 1 *decide*(T,G), code for site i

```
1: variable G' := (Ø,Ø)                                   ▷ a directed graph
2:
3: for all C ⊆ cycles(G) do
4:     if ∀T ∈ C, ¬isAborted(T,G) then
5:         G' := G' ∪ C
6: if T ∈ breakCycles(G') then
7:     return false
8: else
9:     return true
```

A precedence graph contains also for each vertex T a flag indicating whether T is aborted or not: *isAborted*(T,G), and the subset of T's operations: *op*(T,G), which contribute to the relations linking T to others transactions in G.

Given a precedence graph G, we denote $G.\mathcal{V}$ its vertices set, and $G.\mathcal{E}$ its edges set. Let G and G' be two precedence graphs, the union between G and G': $G \cup G'$ is such that:

- $(G \cup G').\mathcal{V} = G.\mathcal{V} \cup G'.\mathcal{V}$,
- $(G \cup G').\mathcal{E} = G.\mathcal{E} \cup G'.\mathcal{E}$,
- $\forall T \in (G \cup G').\mathcal{V}$, *isAborted*$(T,(G \cup G')) =$ *isAborted*$(T,G) \vee$ *isAborted*(T,G').
- and $\forall T \in (G \cup G').\mathcal{V}$, *op*$(T,(G \cup G')) =$ *op*$(T,G) \cup$ *op*(T,G').

We say that G is a subset of G', denoted $G \subseteq G'$, if:

- $G.\mathcal{V} \subseteq G'.\mathcal{V}$,
- $G.\mathcal{E} \subseteq G'.\mathcal{E}$,
- $\forall T \in G.\mathcal{V}$, *isAborted*$(T,G) \Rightarrow$ *isAborted*(T,G'),
- and $\forall T \in G.\mathcal{V}$, *op*$(T,G) \subseteq$ *op*(T,G').

Let G be a precedence graph, *in*(T,G) (respectively *out*(T,G)) denotes the restriction of $G.\mathcal{V}$ to the subset of vertices formed by T and its incoming (resp. outgoing) neighbors. The *predecessors* of T in G: *pred*(T,G), is the precedence graph representing the transitive closure of the dual of the relation $G.\mathcal{E}$ on $\{T\}$.

3.6 Deciding

Each site i stores its own precedence graph G_i, and decides locally to commit or abort a transaction according to it.

More precisely i decides according to the graph $pred(T, G_i)$. Let $cycles(pred(T, G_i))$ denote the set of cycles in $pred(T, G_i)$, for any cycle $C \in cycles(pred(T, G_i))$, i must abort at least one transaction in C. This decision is deterministic, and i tries to minimize the number of transactions aborted.

Formally speaking i solves the minimum feedback vertex set problem over the union of all cycles in $pred(T, G_i)$ containing only non-aborted transactions The minimum feedback vertex set problem is an NP-complete optimization problem, and the literature about this problem is vast [7]. We consequently postulate the existence of an heuristic: *breakCycles*(). *breakCycles*() takes as input a directed graph G, and returns a vertex set S such that $G \setminus S$ is acyclic.

Now considering a transaction $T \in G_i$ and $G = pred(T, G_i)$, using the heuristic *breakCycles*(), $decide(T, pred(T, G_i)G_i)$ returns *false* if i aborts T, or *true* otherwise.

3.7 Closure phase

In our model sites replicate data partially, and consequently maintain an incomplete view of the precedence constraints linking transactions in the system.

They need consequently to complete their view by exchanging parts of their graphs. This is our closure phase:

- When i TO-delivers an operation $o \in T$, i adds T to its precedence graph, and adds o to $op(T, G_i)$. Then i sends $pred(T, G_i)$ to $replicas(out(T, G_i))$ (line 29).
- When i receives a precedence graph G, if $G \not\subseteq G_i$, for every transaction T in G_i, such that $pred(T, G) \not\subseteq pred(T, G_i)$, i sends $pred(T, G \cup G_i)$ to $replicas(out(T, G_i))$. Then i merges G to G_i (lines 31-35).

Once i knows all the precedence relations linking T to others transactions, we say that T is *closed* at site i. Formely T is closed at site i when the following fixed-point equation is true at site i:

$$closed(T, G_i) = \begin{cases} op(T, G) = T \\ \forall T' \in in(T, G_i).\mathcal{V}, closed(T', G_i) \end{cases}$$

Our closure phase ensures that during every run r, for every correct site i, and every transaction T which is eventually in G_i, T is eventually closed at site i.

3.8 Commitment phase

If T is a read-only transaction: $wo(T) = \emptyset$, i commits T as soon as T is executed (line 9).

If T is an update, i waits that T is closed an d holds all its IW locks: function $holdIWLocks(T)$ (line 35).

Algorithm 2 code for site i

1: **variables** $G_i := (\emptyset, \emptyset)$; $pending := \emptyset$
2:
3: **loop** ▷ Initial execution
4: let T be a new transaction
5: $initialExecution(T)$
6: **if** $wo(T) \neq \emptyset$ **then**
7: R-multicast(T) to $replicas(T)$
8: **else**
9: $commit(T)$
10:
11: **when** R-deliver(T) ▷ Submission
12: **for all** $o \in T : i \in replicas(o)$ **do**
13: $pending := pending \cup \{o\}$
14:
15: **when** $\exists o \in pending \wedge i = WLeader(replicas(o))$
16: TO-multicast(o) to $replicas(o)$
17:
18: **when** TO-deliver(o) **for the first time** ▷ Certification
19: $pending := pending \setminus \{o\}$
20: let $T = trans(o)$
21: $G_i.\mathcal{V} := G_i.\mathcal{V} \cup \{T\}$
22: $op(T, G_i) := op(T, G_i) \cup \{o\}$
23: **if** $isRead(o) \wedge \exists o', o' \rightarrow_i o$ **then**
24: $setAborted(T, G_i)$
25: **else if** $isWrite(o)$ **then**
26: $forceWriteLock(o)$
27: **for all** $o' : o' \rightarrow_i o$ **do**
28: $G_i.\mathcal{E} := G_i.\mathcal{E} \cup \{(trans(o'), T)\}$
29: send$(pred(T, G_i))$ to $replicas(out(T, G_i))$
30:
31: **when** receive(T, G) ▷ Closure
32: **for all** $T \in G_i$ **do**
33: **if** $pred(T, G) \not\subseteq pred(T, G_i)$ **then**
34: send$(pred(T, G_i \cup G))$ to $replicas(out(T, G_i))$
35: $G_i := G_i \cup G$
36:
37: **when** $\exists T \in G_i, \begin{cases} i \in replicas(wo(T)) \\ closed(T, G_i) \\ holdIWLocks(T) \end{cases}$ ▷ Commitment
38: **if** $\neg\, isAborted(T, G_i) \wedge decide(T, pred(T, G_i))$ **then**
39: $commit(T)$
40: **else**
41: $abort(T)$
42:

Once these two conditions hold, i computes $decide(T, pred(T, G_i))$. If this call returns *true*, i commits T: for each write operation $o \in wo(T)$, with $i \in replicas(o)$, i considers any write operation o' such that $T \rightarrow trans(o') \in G_i \wedge conflict(o, o')$. If $trans(o')$ is already committed at site i, i does nothing; otherwise i applies o to its database.

3.9 Consistency criterion

Algorithm 2 describes our algorithm. This protocol provides serializability for partially replicated database systems: any run of this protocol is equivalent to a run on a single site [3]. The proof of correctness is in appendix.

3.10 Latency degree

The latency degree measures the minimal length of a causal pathto commit a transaction. We consider the solution of Aguilera et al. to Uniform Reliable Multicast [1], and Paxos as a solution to Uniform Total Order Multicast [14].

In the best run Algorithm 2 achieves a latency degree of 4: 1 for Uniform Reliable Multicast, and 3 for Uniform Total Order Multicast. If a cycle of size c exists, the latency degree increases by c, but only in the worth case since precedence constraints in a cycle are *not* causaly related.

3.11 Initial execution on more than one site

We generalize Algorithm 2 to the case where the initial execution phase does not take place on a single site by computing the read/write dependencies.

More precisely when a site i receives a read o accessing a datum x that it does not replicate locally, i sends o to some replica $j \in replicas(x)$.

When j receives o, j executes it without lock, and upon completion, j sends back to i the write operation on which o depends.

Once i has executed locally or remotely all the read operations in T, i computes the write set and the update values. Then i sends T with the read/write dependencies using Uniform Reliable Multicast. The dependencies are merged to precedence graph when a site receives an operation by Total Order Multicast. The rest of the algorithm remains the same.

4 Related work and concluding remarks

4.1 Related work

Gray et al. [8] investigate how to scale eager and lazy replications to large database systems. They prove that both approach fails as the deadlock rate raises with the square of the number of transactions, and the reconciliation raise as the cube of the number of transactions. Wiesmann et al confirm practically this result [20].

Preventive replication [16] considers that a bound on processor speed, and network delay is known, (i.e. the system is synchronous). Such an assumption does not hold in a large-scale system.

Fritzke et al. [10] propose a replication scheme where sites TO-multicast transactions and executed them upon reception. However this approach requires to prevent deadlocks increasing the abort rate.

In all of these solutions, every replica execute all the operations accessing the data item it replicates. Alonso proves analytically that it reduces the scale-up of the system [2].

The epidemic algorithm of Holiday et al [9] aborts concurrent conflicting transactions and their protocol is not live in spite of one fault.

The DataBase State Machine approach [17] considers full replication and computes a total order of transactions. Recently N.Schiper et al. [19] extend this replication scheme to partial replication, but they still compute a total order.

4.2 Conclusion

We present an algorithm for partially replicated database systems. Our algorithm preserves scalability by applying updates values only, and remains safe and live in presence of faults.

Our protocol solves the problem of partially replicating a database system using the semantics linking concurrent transactions. The key idea is to compute a transitive closure of the semantics relations linking transactions.

The closure of constraints graphs is a classical idea in distributed systems. We may find it in the very first algorithm for State Machine Replication [13], or in a classical algorithm to solve Total Order Multicast [6].[4]

However the concept of closure considering the semantics of concurrent operations has never been studied by the distributed systems community. We believe that this concept applies to any kind of replication problem, and its generalization remains an open and intresting perspective.

[4] In [13] Lamport closes the $\ll$ relation for every request to the critical section. In [6] the total order multicast protocol attributed to Skeen, closes the order over natural numbers to multicast a message.

References

[1] Marcos Kawazoe Aguilera, Sam Toueg, and Borislav Deianov. On the weakest failure detector for uniform reliable broadcast. Technical Report TR99-1741, 30, 1999.

[2] G. Alonso. Partial database replication and group communication primitives, 1997.

[3] Philip A. Bernstein, Vassos Hadzilacos, and Nathan Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987. .

[4] Jean-Michel Busca, Fabio Picconi, and Pierre Sens. Pastis: A highly-scalable multi-user peer-to-peer file system. In José C. Cunha and Pedro D. Medeiros, editors, *Euro-Par*, volume 3648 of *Lecture Notes in Computer Science*, pages 1173–1182. Springer, 2005.

[5] Lásaro Camargos, Fernando Pedone, and Marcin Wieloch. Sprint: a middleware for high-performance transaction processing. *SIGOPS Oper. Syst. Rev.*, 41(3):385–398, 2007.

[6] X. Defago, A. Schiper, and P. Urban. Totally ordered broadcast and multicast algorithms: a comprehensive survey, 2000.

[7] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[8] Jim Gray, Pat Helland, Patrick O'Neil, and Dennis Shasha. The dangers of replication and a solution. In *SIGMOD '96: Proceedings of the 1996 ACM SIGMOD international conference on Management of data*, pages 173–182, New York, NY, USA, 1996. ACM Press.

[9] J. Holliday, D. Agrawal, and A. Abbadi. Partial database replication using epidemic communication, 2002.

[10] Udo Fritzke Jr. and Philippe Ingels. Transactions on partially replicated data based on reliable and atomic multicasts. In *ICDCS '01: Proceedings of the The 21st International Conference on Distributed Computing Systems*, page 284, Washington, DC, USA, 2001. IEEE Computer Society.

[11] Bettina Kemme and Gustavo Alonso. Don't be lazy, be consistent: Postgres-r, a new way to implement database replication. In *The VLDB Journal*, pages 134–143, 2000.

[12] John Kubiatowicz, David Bindel, Yan Chen, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Westly Weimer, Christopher Wells, and Ben Zhao. Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of ACM ASPLOS*. ACM, November 2000.

[13] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, 1978.

[14] Leslie Lamport. Fast paxos. *Distributed Computing*, 19(2):79–103, October 2006.

[15] Athicha Muthitacharoen, Robert Morris, Thomer M. Gil, and Benjie Chen. Ivy: A read/write peer-to-peer file system. In *Proceedings of 5th Symposium on Operating Systems Design and Implementation*, 2002.

[16] Esther Pacitti, Cédric Coulon, Patrick Valduriez, and M. Tamer Özsu. Preventive replication in a database cluster. *Distrib. Parallel Databases*, 18(3):223–251, 2005.

[17] F Pedone, R Guerraoui, and A Schiper. The database state machine approach. *Distrib. Parallel Databases*, 14(1):71–98, July 2003.

[18] 1 Raynal. Eventual leader service in unreliable asynchronous systems: Why? how? *nca*, 00:11–24, 2007.

[19] Nicolas Schiper, Rodrigo Schmidt, and Fernando Pedone. Optimistic Algorithms for Partial Database Replication. In *10th International Conference on Principles of Distributed Systems (OPODIS'2006)*, pages 81–93, 2006. Also published as a Brief Announcement in the Proceedings of the 20th International Symposium on Distributed Computing (DISC'2006).

[20] Matthias Wiesmann and André Schiper. Comparison of database replication techniques based on total order broadcast. *IEEE Transactions on Knowledge and Data Engineering*, 17(4):551–566, 2005.

.1 Additionnal notations

We denote $\mathbb{T}$ the universal set of transactions, $\mathbb{D}$ the universal set of data item, and $\mathbb{G}$ the universal set of precedence graphs.

Let ρ be a run of Algorithm 2, given a site i we denote $event_i$ when the event *event* happens at site i during ρ; moreover if *value* is the result of this event we denote it: $event_i = value$.

Let ρ be a run of Algorithm 2, we denote:

- $faulty(\rho)$ the set of sites that crashes in ρ,
- $correct(\rho)$ the set $\Pi \setminus faulty(\rho)$.
- $committed(\rho)$ the set $\{T \in \mathbb{T}, \exists i \in \Pi, commit_i(T) \in \rho\}$,
- and $aborted(\rho)$ the set $\{T \in \mathbb{T}, \exists i \in \Pi, abort_i(T) \in \rho\}$.

Given a site i and a time t, we denote $G_{i,t}$ the value of G_i at time t.

.2 Proof of correctness

Let ρ be a run of Algorithm 2, we now proove a series of propositions leading us to the conclusion that ρ is serializable run. Since the serializability theory is over a finite set of transactions, we suppose that during ρ a finite subset of $\mathbb{T}$ is sent to the system.

P1

$$\forall T \in \mathbb{T}, (\exists j \in \Pi, \text{R-deliver}_j(T) \in \rho) \Rightarrow (\forall o \in T, \forall i \in replicas(o) \cap correct(\rho), \text{TO-deliver}_i(o) \in \rho)$$

Proof

Let T be a transaction and j a site that R-delivers T during ρ.

F1.1 $\forall i \in replicas(T) \cap correct(\rho), \text{R-deliver}_i(T)$

By the Uniform Agreement property of Uniform Reliable Multicast.

F1.2 $\forall o \in T, \exists k \in correct(\rho) \cap replicas(o), \text{TO-multicast}_k(o) \in \rho$

F1.2.1 $\exists l \in correct(\rho) \cap replicas(o), WLeader_l(replicas(o)) = l \wedge \text{R-deliver}_l(o)$

By fact **F.1.1**, assumption **A1** and the properties of the Eventual Weak Leader Service.

By fact **F1.2.1** eventually a correct site executes line 16 in Algorithm 2.

Fact **F1.2** and the Validity property of Total Order Multicast conclude our claim.

□

In the following we say that a transaction T is submitted to the system: $T \in \mathit{submitted}(\rho)$, if a site i R-delivers T during ρ.

P2

$$\forall T \in \mathit{submitted}(\rho), \forall i \in \mathit{replicas}(T), \forall o \in T, \exists G \in \mathbb{G}, \text{receive}_i(G) \in \rho, o \in \mathit{op}(T,G)$$

Proof

F2.1 $\forall i \in \Pi, \forall t, t', t > t' \Rightarrow G_{i,t} \subseteq G_{i,t'}$

F2.2 $\forall G \in \mathbb{G}, \forall T \in \mathbb{T}, T \in G \Rightarrow T \in \mathit{pred}(T,G)$

By definition of the predecessors of T in precedence graph G.

By proposition **P1**, facts **F2.1** and **F2.2**, and since links are reliable.

□

P3

$$\forall T, T' \in \mathit{submitted}(\rho), T \rightarrow T' \Rightarrow (\exists x \in \mathbb{D}, \exists i \in \mathit{correct}(\rho) \cap \mathit{replicas}(x), o \rightarrow_i o')$$

Proof

By definition of $T \rightarrow T'$, let $o, o' \in T \times T'$ and let j be a site such that $o \rightarrow_j o'$. Since $\mathit{conflict}(o,o')$ and an operation applies on a single data item, we denote x the unique data item such that $x = \mathit{item}(o) = \mathit{item}(o')$.

F3.1 $j \in \mathit{replicas}(x)$

Site j TO-delivers o during ρ and links are reliable.

F3.2 $\exists i \in \mathit{replicas}(x) \cap \mathit{correct}(\rho), \text{TO-deliver}_i(o) \wedge \text{TO-deliver}_i(o')$

By assumption **A1** $\exists i \in replicas(x) \cap correct(\rho)$, and by the Uniform Agreement property of Total Order Multicast, since i is correct during ρ, i TO-delivers both o and o'.

Fact **F3.2** and the Total Order property of Total Order Multicast concludes our claim.

□

P4

$$\forall T \in submitted(\rho), \forall i \in \Pi$$
$$(\exists t, T \in G_{i,t}) \Rightarrow (\exists T_1, \ldots, T_{m \geq 0} \in submitted(\rho), i \in replicasT_m \wedge T \rightarrow T_1 \rightarrow \ldots \rightarrow T_m)$$

Proof

Since $G_{i,0} = (\emptyset, \emptyset)$, let us consider the first time t_0 at which $T \in G_{i,t}$.

According to Algorithm 2 either:

- i TO-delivers an operation $o \in T$ at t_0, and thus $i \in replicas(T)$. QED
- or i receives a precedence graph G' from a site j such that $T \in G'$. Now since links are reliable, denote t_1 the time at which j send G' to i. According to lines 29 and 34, it exists a transactions T' such that $T \in pred(T', G_{j,t_1})$, and a transaction T'' such that $T'' \in out(T', G_{j,t_1})$ and $i \in replicas(T'')$.

  From $T \in pred(T',)$, by definition of the predecessors, we obtain $T \rightarrow \ldots \rightarrow T'$, and from $T'' \in out(T', G_{j,t_1})$ we obtain $T' \rightarrow T''$. Thus $T \rightarrow \ldots \rightarrow T' \rightarrow T''$, with $i \in replicas(T'')$.

□

P5

$$\forall T \in submitted(\rho), \forall i \in correct(\rho),$$
$$(\exists t, T \in G_{i,t}) \Rightarrow (\exists t, op(T, G_{i,t}) = T)$$

Proof

Let T_0 be a transaction submitted during ρ and let i be a site that eventually hold T_0 in G_i.

By proposition **P4** it exists $T_1, \ldots, T_{m \geq 0} \in submitted(\rho)$ such that $i \in replicasT_m$ and $T \rightarrow T_1 \rightarrow \ldots \rightarrow T_m$.

Let $k \in [\![0, m]\!]$, we denote $\mathcal{P}(k)$ the following property:

$$\mathcal{P}(k) \stackrel{\triangle}{=} \forall j \in correct(\rho) \cap replicas(T_k), \exists t \in op(T_0, G_{j,t_0}) = T_0$$

Observe that by proposition **P2** $\mathcal{P}(0)$ is true. We now proove that $\mathcal{P}(k)$ is true for all the k by induction:

Let $o, o' \in T_k \times T_{k+1}$, and $j \in correct(\rho)$ such that $o \rightarrow_j o'$.

Denote t_0 the first time at which j TO-delivers o during ρ.

Denote t_2 the first time at which $op(T_k, G_{j,t}) = T_k$ (since $G_{j,0} = (\emptyset, \emptyset)$, and $\mathcal{P}(k)$ is true).

Denote t_1 the first time at which j To-delivers o' during ρ.

Observe that since $o \rightarrow_j o'$, $t_O < t_1$. It follow that we have three cases to consider:

- cases $t_2 < t_0 < t_1$ and $t_0 < t_2 < t_1$

  In these cases when j To-delivers o', we have:

  $$T_k \rightarrow T_{k+1} \in G_{j,t_1} \wedge op(T_k, G_{j,t_1} = T_k)$$

  Thus,

  $$T_k \in pred(T_{k+1}, G_{j,t_1}) \wedge op(T_k, pred(T_k, G_{j,t_1})) = T_k$$

  and according to Algorithm 2, j sends $pred(T_{k+1}, G_{j,t_1})$ to $replicas(out(T_{k+1},))G_{j,t_1}$.

  Now since $replicas(T_{k+1}) \subseteq replicas(out(T_{k+1},))G_{j,t_1}$, given a site $j \in replicas(T_{k+1})$, eventually j receives $pred(T_{k+1}, G_{j,t_1})$, and merges it into its own precedence graph.

- case $t_0 < t_1 < t_2$

  We consider two-subcases:

  - At t_2 j delivers an operation of T_k, and this operation is different from o'. Now since $T_k \rightarrow T_{k+1} \in G_{j,t_2}$, $\mathcal{P}(k+1)$ is true.
  - If now j receives a graph G such that $op(T_k, G) = T_k$, by definition of t_2, $G \subseteq G_{j,t_2}$, and more precisely, $pred(T_k, G) \not\subseteq pred(T_k, G_{j,t_2})$.

    It follows that j sends $pred(T_k, G \cup G_{j,t_2})$ to $replicas(out(T_k, G \cup \cup G_{j,t_2}))$. Finally since by definition of t_1, $T_k \rightarrow T_{k+1} \in G_{j,t_2}$, we obtain $T_{k+1} \in out(T_k, G \cup G_{j,t_2})$, from which we conclude that $\mathcal{P}(k+1)$ is true.

To conclude observe that since $i \in replicas(T_m)$ and $\mathcal{P}(m)$ is true, eventually $op(T_0, G_{i,t_0}) = T_0$.

□

P6

$$\forall T \in submitted(\rho), \forall i \in correct(\rho),$$
$$(\exists t, T \in G_{i,t}) \Rightarrow (\exists t, \forall T' \in submitted(\rho), T' \rightarrow T \Rightarrow (T', T) \in G_{i,t})$$

Proof

F6.1 $\forall T, T' \in submitted\rho, \forall o, o' \in T \times T', (\exists i \in \Pi, o' \rightarrow_{\Rightarrow} o \forall j \in replicas(o), o \rightarrow_j o'$

By the Uniform Agreement and Total Order properties of Total Order Broadcast

F6.2 $\forall T \in submitted(\rho), \forall o \in T, \forall i \in correct(\rho), (\exists t, oop(T, G_{i,t})) \Rightarrow (\forall T' submitted(\rho), T' \rightarrow T \Rightarrow \exists t, (T, T') \in G_{i,t})$

Since $o \in op(T, G_{i,t})$ and $G_{i,0} = (\emptyset, \emptyset)$, either:

1. $i \in replicas(T) \wedge \text{TO-deliver}_o(i)$

   First observe that since links are reliable $i \in replicas(o)$.

   Let T' be a a transaction, $o' \in T'$ an operation, and k a site such that $o' \rightarrow_k o$.

   By fact **F6.1** since $i, j \in replicas(o)$, $o' \rightarrow_i o$.

2. $\exists G \in \mathbb{G}, \text{receive}_T(G) \wedge o \in op(T, G)$

   According to Algorithm 2 it exists $k_0, \ldots, k_m$ sites sucht that:

   - k_0 TO-delivers o during ρ, and execute line 29 sending $pred(T, G_{k_0})$ with $o \in op(T, predecessorsTG_{k_0})$ and $k_1 \in replicas(out(T, G_{k_0}))$.
   - k_1 receives $pred(T, G_{k_0})$ during ρ and then execute line 29 or line 34, sending a precedence graph G such that $pred(T, G_{k_0}) \subseteq G$ to a set of replicas containinig k_2.
   - etc ... until i receives it.

   Consequently $pred(T, G_{k_0}) \subseteq G_{i,t}$, and according to our reasonning in item 1, we conclude that fact **F6.2** is true.

Fact **F6.2** and proposition **P5** conclude.

□

We are now able to proove our central theorem: every transaction is eventually closed at a correct site.

T1

$$\forall T \in \mathit{submitted}(\rho), \forall i \in \mathit{correct}(\rho), \\ (\exists t, T \in G_{i,t}) \Rightarrow (\exists t, \mathit{closed}(T, G_{i,t}))$$

Proof

We consider that a finite subset of $\mathbb{T}$ are sent to the system, consequently $\mathit{submitted}(\rho)$ is also finite. Let C_T be the graph resulting from the transitive closure of the relation $\rightarrow$ on $\{T\}$. According to proposition **P6**, C_T is eventually in $G_{i,t}$, and thus according to proposition **P5**, T is eventually closed at site i.

□

P7

$$\forall T \in \mathit{submitted}(\rho), \forall i, j \in \Pi, \forall t, t', \\ (T \in G_{i,t} \wedge T \in G_{j,t'} \wedge \mathit{closed}(T, G_{i,t}) \wedge \mathit{closed}(T, G_{j,t'})) \Rightarrow (\mathit{pred}(T, G_{i,t}) = \mathit{pred}(T, G_j t'))$$

Proof

F7.1 $\mathit{pred}(T, G_i).\mathcal{V} = \mathit{pred}(T, G_j).\mathcal{V}$

Let $T' \in \mathit{pred}(T, G_i)$. By definition it exists $T_1, \ldots, T_m$ such that $T' \rightarrow T_1 \rightarrow \ldots \rightarrow T_m \rightarrow T \subseteq G_i$. By an obious induction on m using proposition **P6** we conclude that T' is also in $\mathit{pred}(T, G_j)$.

F7.2 $\mathit{pred}(T, G_i).\mathcal{E} = \mathit{pred}(T, G_j).\mathcal{E}$

Identical to the reasonning proposed for fact **F7.1**.

F7.2 $\forall T' \in \mathit{pred}(T, G_i), \mathit{op}(T', \mathit{pred}(T, G_i)) = \mathit{op}(T', \mathit{pred}(T, G_j))$

By fact **F7.1** and since T is closed at both sites i and j.

F7.4 $\{T' | \mathit{isAborted}(T, G_i)\} = \{T' | \mathit{isAborted}(T, \mathit{pgraphSite}j)\}$

Let $T' \in \mathit{pred}(T, G_i)$ such that $\mathit{isAborted}(T', \mathit{pred}(T, G_i))$. According to Algorithm 2, it exists a site k and a read operation $r \in T'$ such that k TO-delivers r during ρ, and then k set the aborted flag of T' in its precedence graph.

Now let k' be a replica of r, by the Uniform Agreement and the Total Order Property of Total Order Multicast, when k' TO-delivers r, it also set the aborted flag of T' in its precedence graph.

By the conjunction of facts **F7.1** to **F7.4**.

□

We proove now that ρ is serializable [3].

Let $O(x,\rho)$ be the set of write operation over datum x during ρ, we define the relation $\ll$ as follows:

$$\forall x \in \mathbb{D}, \forall o_1, o_2 \in O(x,r), x_1 \ll x_2 \stackrel{\triangle}{=} \exists i \in \mathit{replicas}(x), o \rightarrow_i o'$$

P8 $\ll$ is a version order for ρ.

Proof

Let $O(x,\rho)$ be the set of write operation over datum x during ρ; and let $i \in \mathit{replicas}(x) \cap \mathit{correct}(\rho)$ (assumption **A1**).

According to Algorithm 2 o is executed only if $\mathit{trans}(o)$ is committed during ρ consequently i commits during ρ any transaction T such that $\exists o \in wo(T), O(x,\rho)$. Consequently $\ll$ is total over $O(x,\rho)$, and by the Total Order and Uniform Agreement properties of Total Order Multicast, $\ll$ is an order over $O(x,\rho)$.

□

P9

$$\forall T, T' \in \mathit{MVSG}(\rho, \ll), \quad ((T,T') \in \mathit{MVSG}(\rho, \ll) \wedge wo(T) \neq \emptyset \wedge wo(T') \neq \emptyset) \Rightarrow T \rightarrow T'$$

Proof

F9.1 If (T,T') is a read-from edge, then $T \rightarrow T'$

Let (T,T') be a read-from relation. By definition it exists a site i, a write $w[x] \in T$, and a read $r[x] \in T'$ such that during ρ at site i w write x then r reads the value written by w.

Denote t and t' respectively the times at which these two events occured; according to Algorithm 2:

F9.1.1 TO-deliver$_o(i) <_\rho t <_\rho t'$

Then since $T' \in MVSG(\rho, \rightarrow)$, $T' \in submitted(r)$, by assumption **A1**, it exists $j \in replicas(x) \cap correct(r)$ such that TO-deliver$_j(o')$.

Now, TO-deliver$_i(o) \Rightarrow tomdeliverSiteoj$ by the Uniform Agreement, and the Total Order properties of Total Order Multicast. Consequently using fact **F9.1.1**,

$$\neg(\text{TO-deliver}_i(o') <_\rho \text{TO-deliver}_i(o)) \Rightarrow \text{TO-deliver}_j(o) <_\rho \text{TO-deliver}_j(o')$$

concluding our claim.

F9.2 If (T,T) is a version-order edge, then $T \rightarrow T$

Let T_1, T_2, T_3 be three transactions committed during ρ, and suppose that it exists a version-order edge $(T_1, T_2) \in MVSG(\rho, \rightarrow)$.

According to the definition of a version order it follows either:

1. it exists $w_1 \in wo(T_1), w_2 \in wo(T_2)$, and $r_3 \in ro(T_3)$ such that $r_3[x_3]$, $w_1[x_1]$ and $x_1 \ll x_2$.

   By definition of $x_1 \ll x_2 \Rightarrow T_1 \rightarrow T_2$.

2. it exists $r_1 \in ro(T_1), w_2 \in wo(T_2)$ and $w_3 \in wo(T_3)$ such that $r_1[x_3]$, $w_2[x_2]$ and $x_3 \ll x_2$.

   Let $i \in replicas(x) \cap correct(\rho)$ (by assumption **A1**). Since $T_1, T_2, T_3 \in committed(r) \subseteq submitted(r)$, i TO-delivers r_1, w_1 and w_3 during ρ. Now according to the Total Order property of Total Order Multicast, since $x_3 \ll x_2$, $w_3 \rightarrow_i w_2$.

   Let j be a site on which $r_1[x3]$ happens. Since $w_3 \rightarrow_i w_2$, according to our definition of *commit*() (Section 3.8), $w_2 \prec r_1$.

   Now since $T_1 \in committed(\rho)$, necessarily $r_1 \rightarrow_i w_2$ (otherwise T_1 is aborted: line 24).

By facts **F9.1** and **F9.2**

□

P10 ρ is serializable.

Proof

Consider the sub-graph G_u of $MVSG(r, \ll)$ containing all the transactions T such that $wo(T) \neq \emptyset$, and the edge linking them.

F10.1 G_u is acyclic.

Let $T_1, \ldots, T_m \in G_u$ such that $T_1, \ldots, T_{m \geq 1}$ forms a cycle in G_u, and recall that by definition $T_1, \ldots, T_m \in committed(r)$

According to proposition **P9**, $T_1 \rightarrow \ldots \rightarrow T_m \rightarrow T_1$.

Let $i \in replicas(T_1)$, and denote t the time at which i commits T during ρ.

Acoording to Algorithm 2 at time t, $closed(T_1, G_{i,t})$.

Now according to Algorithm 1, and since i commits T_1 at time t,

$$\exists k \in [\![2, m]\!], T_k \in breakCycles(pred(T_1, G_{i,t}))$$

Let $j \in replicas(T_k)$ such that j commit T_k during ρ, and denote t" the time at which this event happens.

Since $T_1 \in pred(T_k, G_{j,t'})$ and $T_k \in pred(T_1, G_{i,t})$, $pred(T_1, G_{i,t}) = pred(T_k, G_{j,t'}, .)$

Consequently since $breakCycles()$ is deterministic, j cannot commit T_k during ρ. Absurd.

F10.2 $MVSG(\rho, \rightarrow)$ is acyclic.

By fact **F10.1** and since read only transactions are executed using two-phases locking.

Fact **F10.2** induces that ρ is serializable.

□

Unité de recherche INRIA Rocquencourt
Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399