A Mechanism for Sequential Consistency in a Distributed Objects System

Cristian Ţăpuş, Aleksey Nogin, Jason Hickey, and Jerome White
California Institute of Technology
Computer Science Department
MC 256-80, Pasadena, CA 91125
{crt,nogin,jyh,jerome}@cs.caltech.edu

Abstract

This paper presents a new protocol for ensuring sequential consistency in a distributed objects system. The protocol is efficient and simple. In addition to providing a high-level overview of the protocol, we give a brief discussion of the implementation details. We also provide a mathematical model that we used to prove the correctness of our approach.

1 Introduction

Computing systems have evolved from single, static computation nodes to dynamic distributed environments. This evolution has raised new issues for distributed objects systems, such as maintaining consistency in distributed filesystems. Several group communication models have been designed [2, 4], but few address the stricter and more intuitive consistency models required to facilitate such complex communication schemes.

In this paper we present a decentralized protocol for ensuring the sequential consistency of accesses to objects in a loosely coupled distributed environment, with specific application to distributed filesystems.

This paper is organized as follows. First, we present the problem statement. We then give an informal overview of a protocol that guarantees sequential consistency of access in the distributed objects system. Next we present a mathematical model and use it to prove the basic properties of our protocol. We conclude by presenting future directions of research and possible extensions of our protocol.

2 Problem statement

Consider a distributed objects system consisting of processes and data objects. Objects are to be stored in a distributed fashion by subsets of the set of all processes in the system; the membership in these subsets is dynamic. Processes access object data through read and write operations. Each process may be accessing several different objects at a time. The goal of the protocol is to provide a guarantee that the object accesses are sequentially consistent. A system implements a sequentially consistent model if the result of any execution is the same as if the operations of all the processes were executed on a "master copy" in a sequential order, and, furthermore, the operations issued at each process appear in the same sequence as specified in the program.

The protocol we propose makes the following assumptions:

1. The only communication between processes is through read and write operations to the data of the shared objects.

2. The number of processes that access each individual object is small compared to the total number of processes in the system. In particular, achieving a consensus between all processes accessing a certain object at the time is considered practical, while achieving consensus between all processes in the system is considered impractical.

3. The network transport protocol is reliable. If a message is sent by a process, it is eventually delivered.

4. Nodes are fail-stop. When a node fails, it never comes back again.

3 Protocol Overview

We implement access to individual objects via group communication. Processes are organized in groups; each group is assigned to an object Optionally, opening and closing an object could be mapped to join and leave operations for the corresponding group.

We assume that each group uses a *group total order* communication mechanism. Each access to object data (i.e. a read or a write operation) is implemented by sending an appropriate message to the group associated with an object.

When a write message is delivered, the local copy of the object data, if present, is updated accordingly. The result of a read request is based on the local data at the time the corresponding read message is *delivered back* to the initiating process.

The presence of the *explicit read messages*, the group total order, together with some simple constraints on the way individual processes send out their messages guarantee sequential consistency.

4 Sequential consistency

As stated in Section 2, we assume that the only communication mechanism between the applications is through the shared object data. The protocol implements each read or write access as a message in the system. The problem of sequential consistency reduces to the problem of imposing an order on these messages.

Definition 4.1. We will call an ordering O on messages *consistent* if it satisfies the following conditions:

1. On messages originating at the same process, the order O is consistent with the order in which the messages were sent by the process.

2. On messages sent to the same group, the order O is consistent with the group total order imposed by the group communication mechanism.

Next, we show that if there exists a consistent total order of messages, as defined above, the sequence of messages exchanged by the processes taking part in the protocol is sequentially consistent.

Lemma 4.2. If all messages sent in a particular run of the protocol can be arranged in a *consistent total order*, then this run of the protocol is sequentially consistent.

Proof. Suppose $m_0, m_1, \ldots, m_n$ is a consistent *total* ordering of messages. We can make the execution sequential as follows. First, run the application that originated the message m_0 until the point where it originated m_0. Then perform the read/write operation specified by m_0. Next, run the application that originated m_1 until the point where it originated m_1. Furthermore, perform the read/write operation specified by m_1, and so forth. Due to condition 1 in Definition 4.1, the ordering of messages is consistent with the execution order of each specific process, so the sequential schedule is feasible. Because of the condition 2, the operations on each object are performed in the order specified by the group ordering, which means that they are performed in the same order as in each local copy in the parallel execution. This guarantees that the data returned by the read operations in the sequential schedule is the same as it was in the parallelized execution. □

Note that we do not require that all the messages in the system be delivered in the same order. It is acceptable for certain messages (sent to different groups) to be delivered "out of order" by a process in certain cases.

Example 4.3. Figure 1 presents a situation where messages sent by processes P_3 and P_4 could be seen in different orders by P_1 and P_2. Assume P_3 and P_4 each send one write message, call them w_{3x} and w_{4y}. In the case that neither of the processes P_1 and P_2 was actively accessing objects X and Y at the time, they can deliver messages w_{3x} and w_{4y} in any order. This does not break sequential consistency because the applications running at P_1 and P_2 are not allowed to see the effects of these write operations until they send *an explicit read message*.

Assume now that P_1 is active and it sees that w_{3x} happens before w_{4y}. In other words, it sends two read messages — first r_{1x} and then r_{1y}. It delivers r_{1x} after w_{3x}, while delivering r_{1y} before w_{4y}. Now the condition 1 in Definition 4.1 guarantees that any consistent order will contain r_{1x} before r_{1y} and the condition 2 guarantees that any consistent order will contain w_{3x} before r_{1x} and r_{1y} before w_{4y}. This means that every consistent order must place w_{3x} before w_{4y}. However, if P_2 is not actively accessing X and Y, it can still receive messages "out of order" without affecting sequential consistency.

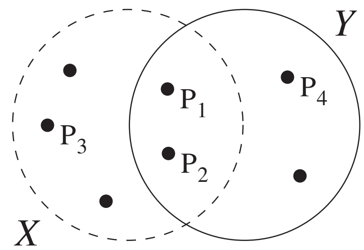

Figure 1: An "out of order" interleaving of messages from P_3 and P_4 is allowed during the passive periods of P_1 and P_2. Once P_1 becomes active it enforces its own thumbprint on the order of messages through the read and write operations it performs.

5 Achieving sequential consistency

We approach the problem of guaranteeing sequential consistency in our implementation by picking a specific (*post-factum*) total ordering of messages O. We provide an implementation which makes sure that O will always be consistent (in the sense of Definition 4.1).

There are different ways of picking an appropriate O. We believe that during the protocol execution, the earlier the messages get assigned to order O, the less intrusive the corresponding protocol adjustments will be. Since O must be consistent with the group total order (see condition 2 in Definition 4.1), unless we adjust the group communication protocol to be aware of O, the earliest a message can be assigned to O is when the group communication protocol assigns the message to a group total order.

Therefore, we pick O as follows — the messages in O are ordered according to the order (from the point of view of an external global clock) in which they are assigned a "group sequence number" by the group total order communication protocol. This is expressed by the following lemma.

Lemma 5.1. The protocol described in Section 3 is sequentially consistent if the following conditions are met:

- The group communication protocol will order messages sent by the same process to the same group consistently with the order in which they were sent.
- When a process attempts to send a message to a different group than it sent its previous message to, the send operation is *blocked* and is *not performed* until all the messages already sent by this process get assigned to the group total order by the group communication protocol.

By "*assigned to the group total order*" we mean that the group communication protocol commits to putting the message right after a specific message (which is already assigned to the group total order) in the group total order.

Proof. Let us define the relation O on messages as "*message* 1 *was assigned to the group total order before (according to the external global clock) message* 2". We claim that under the conditions above, relation O is a consistent global total order.

The relation O is obviously a total order (without the loss of generality, we can assume that no two events are ever perfectly simultaneous). By construction, O agrees with the group total order (this follows from the way we have defined the notion of "*assigned to the group total order*").

We are left to show that on messages sent by the same process, O agrees with the process send order. If both messages were sent to the same group, then condition 5.1 of the lemma ensures that O orders them correctly. If the messages are sent to different groups, then condition 5.1 of the lemma ensures that the sending of the second message will be delayed until it can be guaranteed that O will order them correctly.

Therefore, O is indeed a consistent global total order on messages and by Lemma 4.2 we are guaranteed sequential consistency. □

6 Protocol implementation overview

Lemma 5.1 provides most of the information needed to describe our protocol implementation. In order to achieve sequential consistency, we need a group total order communication protocol, which makes sure that the messages sent by the same process to the same group are ordered consistently with the order in which they were sent.

The implementation of the total order of messages in a small group could be done through a variety of methods. Decentralized group total order communication protocols, like those presented in Section 8, could be used. Centralized methods, like the sequencer approach, can also be used. In this approach, each group has a sequencer node that gives out labels for each of the messages sent in that particular group.

If a process does not maintain a local copy of the object data then the group communication protocol needs to make sure that whenever a read message is processed, the originator of the message is given the appropriate data. In the case of a centralized (sequencer) approach, it is sufficient if only the sequencer maintains the data and sends the correct version in response to all the read messages.

One of the key elements of our approach is the way we guarantee sequential consistency of messages in two groups that have common processes. When a process sends multiple messages to the same group, it is allowed to send the request for a sequence number and then continue with its execution until the sequence number is granted. In the meantime, the process may send multiple sequence number requests to the same group. However, when a process switches the group it sends the message to, it has to wait for all its previously requested sequence numbers to be granted before sending a request for a sequence number in the second group.

According to Lemma 5.1, this guarantees sequential consistency.

7 Mathematical model

This section presents a mathematical model for the problem presented in Section 2 based on the protocol presented in Section 3 and provides a set of requirements for a possible implementation.

7.1 Definitions

First, we define the terms used throughout this section. Consider the set of processes $\mathbb{P} = \{p_i \mid i \in P\}$, where P is a set of indices identifying the processes. We also define a set of groups: $\mathcal{G} = \{g_j \mid j \in G\}$, where G is a set of indices identifying the object associated with each group (group g_i is associated with object i).

Let K be a totally-ordered set of unique labels (or sequence numbers) used by processes to mark the order of the messages they send. Let L be a totally ordered set of unique labels (or sequence numbers) the group total order mechanism assigns to messages sent to each group.

Now we can define the set $\mathbb{M}$ of all messages[1] that can potentially be sent in the system as $\left\{m_{ij}^{kl} \mid i \in P, j \in G, k \in K, l \in L, m \in \{read, write\}\right\}$, where m_{ij}^{kl} represents a message from process p_i with process label k, sent to group g_j with group label l.

[1]Strictly speaking these are not messages, but rather message "headers" that are assigned *post-factum*. However, in this model this distinction can be safely ignored.

We also define two relations $<_p$ and $<_g$ on messages in $\mathbb{M}$ as follows:

- $<_g$: $m_{ab}^{cd} <_g m_{a'b'}^{c'd'}$ *iff* $d < d' \wedge b = b'$
- $<_p$: $m_{ab}^{cd} <_p m_{a'b'}^{c'd'}$ *iff* $c < c' \wedge a = a'$

The $<_g$ defines a relation on messages sent in each group, and $<_p$ defines a relation on messages originating at each process. To make the analogy with the protocol description in Section 4, the $<_g$ relation is given by the order imposed on messages by the group communication mechanism used inside each group. The $<_p$ relation is the order imposed on messages by the run trace of each process. If a send operation on message mx occurs in the run before the send operation of message my, and both messages originate at the same process, then $mx <_p my$.

Now we will consider the set of messages sent in the system for a specific run.

Definition 7.1. We will call a subset M of the set $\mathbb{M}$ *feasible* if for any two messages $m_{ab}^{cd}, m_{a'b'}^{c'd'} \in M$ we have:

1. Uniqueness:

   $(a = a' \wedge c = c') \Leftrightarrow (b = b' \wedge d = d')$.

   This condition states that there can be at most one message in M that was sent by a specific process a, and labeled with a specific process label c. It also states that there can be at most one message sent to a specific group labeled by a specific group label.

2. Ordering consistency: $(a = a' \wedge b = b') \Rightarrow (c < c' \Leftrightarrow d < d')$.

   For messages sent by the same process to the same group, the ordering of messages by the originating process agrees with the ordering of the same messages by the destination group.

Definition 7.2. Finally, we define a relation $\prec \subseteq M \times M$ as $\prec := <_g \cup <_p$. This relation is the minimal requirement for consistency (see Definition 4.1) – an ordering on messages is consistent *iff* it agrees with $\prec$.

7.2 Sequential consistency

Consider relation $\prec$ of Definition 7.2 on a feasible set of messages M. We present an example that illustrates how we can obtain a cyclic dependency of messages that is not sequentially consistent. The setup in Figure 1 can also be used to show a simple

P$_1$		P$_2$	
read(X)	m_{1x}^{ab}		
		m_{2y}^{cd}	read(Y)
write(Y,1)	m_{1y}^{ef}		
		m_{2x}^{gh}	write(X,1)

Table 1: An interleaving of read and write operations leading to non-sequentially consistent outcome.

example of an object access that could lead to a cyclic dependency. Consider the two processes, P$_1$ and P$_2$, that access objects X and Y. Each process issues two operations, one on each object. Without loss of generality, assume process P$_1$ performs a read operation on data object X and a write operation on data object Y. Similarly, process P$_2$ performs a read operation on data object Y and a write operation on data object X. We can order messages according to the order relation $\prec$ as follows. The read operation, represented in message notation as m_{1x}^{ab}, is performed by process P$_1$ before the write operation mapped to message m_{1y}^{cd}. Similarly, the read operation performed by process P$_2$, m_{2y}^{ef}, is performed before the write operation m_{2x}^{gh}. An illustration of this can be found in Table 1.

If we assume that the initial values of both X and Y are 0, and that the read initiated by process P$_2$ returned value 1, then the read operation must have happened after the write performed by P$_1$. We extend the relation to include messages m_{1y}^{gh} and m_{2y}^{ef}, giving us the following:

$$m_{1\ x}^{a\ b} <_p m_{1\ y}^{e\ f} <_g m_{2\ y}^{c\ d} <_p m_{2\ x}^{g\ h}$$

According to the definition of relation $\prec$ and subject only to the feasibility Definition 7.1, we can also include the pair of messages $m_{2\ x}^{g\ h}$ and $m_{1\ x}^{a\ b}$ in our relation $\prec$. This means that the read(X) operation performed by P$_1$ was performed after P$_1$. By a simple examination on labels a, b, c, d, e, f, and g and the relations deduced from the definition of $\prec$, no constraint or assumption has been violated. If that is the case, we have the following order:

$$m_{1\ x}^{a\ b} <_p m_{1\ y}^{e\ f} <_g m_{2\ y}^{c\ d} <_p m_{2\ x}^{g\ h} <_g m_{1\ x}^{a\ b}$$

This enforces the return value of read(X) in process P$_1$ to be 1. It is easy to show now that with these return values from the two read operations, there is no possible sequentially consistent order of the four operations, subject to the conditions presented in Section 5.

We must therefore break the kind of cycles we have seen above.

First we define a new relation on message in M.

Definition 7.3. Consider the relation $<_d$ defined as: $m_{ab}^{cd} <_d m_{a'b'}^{c'd'}$ iff $d < d'$.

Next, we show how to ensure that this relation is acyclic.

Lemma 7.4. Suppose a feasible set M satisfies an additional *strict ordering* property: $a = a' \Rightarrow (c < c' \Leftrightarrow d < d')$. In other words, messages originating at a node a are related under $<_p$ if and only if they are also labeled with group labels that are similarly related under the numbers less than relation. Then the relation $<_d$ is acyclic on M.

Proof. From the initial assumption it follows immediately that L is totally ordered by $<_d$. □

Next, we show that both $<_g$ and $<_p$ relations are subsets of $<_d$. From there we will be able to show that relation $\prec$, which is the union of the two relations, is also a subset of relation $<_d$. It directly follows that $\prec$ is an acyclic relation.

Lemma 7.5. $<_g \subseteq <_d$.

Proof. By the definition of $<_g$, all messages related by $<_g$ are also related by $<_d$. □

Lemma 7.6. Under the conditions of Lemma 7.4, $<_p \subseteq <_d$.

Proof. By the definition of $<_p : m_{ab}^{cd} <_p m_{a'b'}^{c'd'}$ iff $a = a' \wedge c < c'$. According to the strict ordering condition $a = a' \Rightarrow (c < c' \Leftrightarrow d < d')$. This means that $d < d'$, so $m_{ab}^{cd} <_d m_{a'b'}^{c'd'}$. □

Theorem 7.7. Under the conditions of Lemma 7.4, relation $<_d$ is a consistent total order on M (as defined in Definition 4.1).

Proof. From Lemma 7.5 and Lemma 7.6 it immediately follows that $<_d$ is consistent. By definition and Lemma 7.4, $<_d$ is total order on M. □

From this theorem, we conclude that any implementation that guarantees the strict ordering condition of Lemma 7.4, will also guarantee sequential consistency. If we take d to always be the timestamp (according to the external global clock) of the moment each message gets assigned a group serial number by the group communication protocol, we will arrive at exactly the same implementation as outlined in Section 6.

8 Related Work

Group communication mechanisms could be used for the communication needed to maintain consistency of objects stored in distributed environments. However, most of the existing group communication systems in the literature are not well suited for this problem as they usually operate on disjoint groups of nodes and are mostly concerned with maintaining consistency of messages within individual groups only [6]. The distributed objects systems we are concerned with are not suited for global group communication due to the large size of the set of nodes composing the system. Furthermore, the interaction between the nodes in the system is such that actions can not be isolated in disjoint sub-groups. In our work we address both the issue of non-disjoint groups and the sequential consistency of messages in the system.

Other group communication work concentrates on developing a reliable multicast layer to achieve total order [1, 3]. For the purposes of a filesystem, for example, this approach is not well fitted due to the lower level of the approach and the possibility of a very large number of multicast groups present in such systems. The control of such ordering should ideally be at a higher level.

There are systems that address both total order at the application layer and multi-group process membership [7, 5]. Some of these systems assume a hierarchy of the nodes they use to obtain global total ordering in the subgroups.

The main difference between the other presented systems and our work is that while they try to provide a general group communication protocol, we address the problem of providing sequential consistency of accesses to objects in a distributed objects system. Our problem lies at a higher level and is relying on some of the lower level protocols provided by the other work mentioned in this section.

9 Conclusions and Future work

The protocol presented in this paper addresses the issue of sequential consistency of accesses to objects in a distributed objects system. Our main contributions consist of allowing processes to deliver messages in an arbitrary order during their passive periods, as well as providing consistency of inter-group communication. Global sequential consistency is achieved by enforcing explicit read messages, group total order on small sub-groups of nodes, and very simple requirements on sending messages. The correctness of our protocol is proved by a mathematical model.

Future directions of research include the extension of the protocol to support random faults at both the node level and the network level. We are also interested in investigating different models, implemented in a formal model checker, that would allow us to prove complex liveness and safety properties of the protocol.

References

[1] G. V. Chockler, N. Huleihel, and D. Dolev. An adaptive totally ordered multicast protocol that tolerates partitions. In *Proceedings of the seventeenth annual ACM symposium on Principles of distributed computing*, pages 237–246. ACM Press, 1998.

[2] Gregory V. Chockler, Idit Keidar, and Roman Vitenberg. Group communication specifications: a comprehensive study. *ACM Comput. Surv.*, 33(4):427–469, 2001.

[3] George Coulouris, Jean Dollimore, and Tim Kindberg. *Distributed Systems: Concepts and Design.* Addison-Wesley, second edition, 1994.

[4] Xavier Defago, André Schiper, and Peter Urban. Total order broadcast and multicast algorithms: Taxonomy and survey. Technical Report 200356, École Polytechnique Fédérale de Lausanne, September 2003.

[5] Paul D. Ezhilchelvan, Raimundo A. Macêdo, and Santosh K. Shrivastava. Newtop: a fault-tolerant group communication protocol. In *Proceedings of the 15th International Conference on Distributed Computing Systems (ICDCS'95)*, page 296. IEEE Computer Society, 1995.

[6] Jason Hickey, Nancy A. Lynch, and Robbert van Renesse. Specifications and proofs for ensemble layers. In *Proceedings of the 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems*, pages 119–133. Springer-Verlag, 1999.

[7] L. E. Moser, P. M. Melliar-Smith, D. A. Agarwal, R. K. Budhia, and C. A. Lingley-Papadopoulos. Totem: a fault-tolerant multicast group communication system. *Commun. ACM*, 39(4):54–63, 1996.