Research Approach Advance in Concurrency Management Algorithms for Cooperative Work

Eric Garcia, Hervé Guyennet, Julien Henriet, Jean-Christophe Lapayre

Laboratoire d'Informatique de l'Université de Franche-Comté, 16 Route de Gray, 25030 Besançon Cedex, France

Summary
This article presents a study and the development of protocols for concurrency problems in cooperative work applications.We first present our platform, which allows us to create cooperative applications without having to manage communications, consistency and synchronization. In this CORBA bus based platform, we can distinguish services for continuous stream transport and services for cooperation. We place special emphasis on these last services and especially on the two protocols involved : the Pilgrim and the Chameleon. We show the approach that induces us to evolve from the Pilgrim protocol to the Chameleon protocol. The Pilgrim is a token ring based protocol, which orders access on shared objects to allow the management of concurrency problems. But the main default of a token ring based protocol is to visit all sites including no-producers, therefore we have developed the Chameleon protocol, which allows the virtual topology to be reconfigured. We describe these two protocols, we show their performance and we demonstrate their qualities.

Keywords:
Concurrency Protocols, Cooperative Work, Distributed Algorithms,Performance.

Introduction

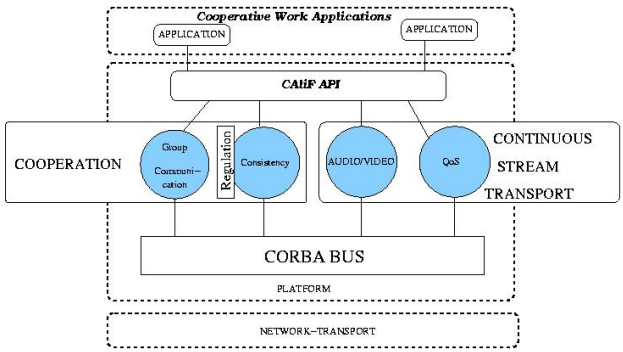


Fig. 1 CAliF Multimedia Platform.

The design of a cooperative application integrating multimedia implies various domains: networks, distributed systems, multimedia, data consistency, Human-Machine Interface... Therefore, it is interesting to create a platform combining the functionalities common to all types of cooperative applications [1]. It is the case of the CAliF Multimedia platform [2,3] that allows us to create cooperative applications without taking into account problems of communication management, consistency, synchronization and multimedia management (Figure 1). It uses a CORBA bus [4] and provides services such as continuous stream management, shared data consistency management, quality of service management, and finally group communication managing the virtual topology of cooperating sites, as well as the linkage for continuous stream transport.

This platform is based on a CORBA bus that manages problems of heterogeneity, interoperability, portability, and access to resources. It has four main services:

- The *Group Communication* service allows the members of a cooperative application to broadcast information (only discrete media: text, drawings...) they produce. All the objects generated and modified by a user are transported towards the other group members. This service also manages membership or departure of the group members, controls message broadcasting such as video connection requests and messages of continuous stream control. It must ensure the consistency of group topology as well as the integrity and order of messages sent by group members;
- The *Consistency* service maintains the distributed shared memory consistency, i.e. ensures that all the cooperative application objects, that are replicated on all sites, are identical for each user. This service interacts with the communication service to define messages to broadcast in order to maintain system consistency. It uses the *Regulation* module programmed by the application designer to determine object management strategies;
- The *Audio/Video* service is used for continuous stream transport (audio, video). The buses that are CORBA 2 compliant do not efficiently manage the broadcast of continuous media. To provide this functionality, we developed the *Audio/Video* service of CAliF Multimedia that implements an OMG proposal for standardization of continuous stream transport [5].

Manuscript received October 5, 2025
Manuscript revised October 20, 2025
https://doi.org/10.22937/IJCSNS.2025.25.10.1

The control commands (creation, stop, pause,...) are carried through the communication service, but the stream goes out of the CORBA bus because of performance and easy management of video and voice synchronization;

- The *QoS* service allows the users to translate quality of service requests of applications and physically reserve necessary resources for it. For example, the QoS service is used by the *Audio/Video* service during the creation of connections for continuous stream transport.

We can distinguish services for continuous stream transport (*Audio/Video* and *QoS*) and services for cooperation (*Group Communication* and *Consistency*). In this paper we place special emphasis on these last services and especially on the protocols involved.

The first part is composed of a description of our motivations. The second part of this paper presents the Pilgrim algorithm that is the main component of the CAliF consistency service. We finish this first part by a Pilgrim optimization proposal. The main default of a token ring type protocol is to visit all sites including no-producers, it is the reason why we have developed the Chameleon protocol, which allows the virtual topology to be reconfigured using two communication techniques: a rotating sequencer for one site and a symmetric approach for another. This paper ends by a further work part that explains our new research ways.

2. Communication in Cooperative Work

2.1 Presentation

Cooperative work applications allow distant users who may have different roles and rights to be interactive. Many factors can evolve during the life of this kind of application.

- The workload can vary significantly with the time. For example, all the cooperative members may have to react simultaneously to an event, and then to remain inactive for a long time.
- The workload can also vary in space. Two members can work together and the others can observe their exchanges and react when necessary.
- The roles and rights of participants can change. An observer member (inactive) may become an actor after a particular event. It is the case for a referee member who is active only when a fault is committed.
- The Quality of service can be modified, for example if a video connection between two sites is introduced.

All these variable parameters show that it is difficult to work with a fixed virtual topology for communications in the cooperative world. It seems important to be able to change this topology dynamically according to the application requirements.

2.2 Related Work

In a cooperative application, site actions have to be broadcasted in order on the group. In [7] we distinguish three types of protocol for group communication:

- Asymmetric protocols: messages are sent to a central sequencer node that multicasts them in order.
- Symmetric protocols: all sites have the same role; the management of the message order becomes very costly.
- Rotating sequencer protocols: they are placed between the symmetric and asymmetric protocols. The two major subclasses are the sequencing acknowledgement strategy and the token strategy.

We will study this last type of protocol, and more particularly the token strategy: possession of the token gives a site the right to emit.

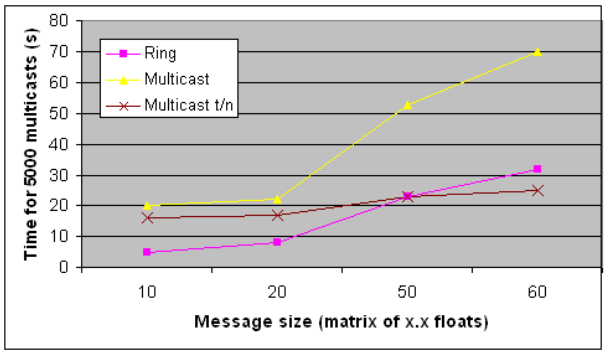


Fig. 2 Multicast Time.

The Pilgrim algorithm [7,8] uses a virtual ring topology with a token that contains some information such as the modifications of the Distributed Shared Memory (DSM) [9,10]. This type of protocol allows us to manage at low cost the shared data consistency. Furthermore, it is efficient for cooperative applications. Figure 2 compares the use of a symmetric technique (multicast) with a rotating sequencer protocol technique (ring) with four sites on a 100Mb Ethernet network. The multicast t/n curve represents a multicast technique with messages n times smaller than for the ring technique with a token size of t. Indeed, all modified information by the n persons involved in a cooperative task are placed on the token. This corresponds to the multicast of a message, which is n

times smaller than the token, since each application member sends only its own modifications: this size is in reality more important to ensure the consistency of shared object with such a technique. We can observe that the token technique is always better than the multicast one. It is also better than the multicast t/n technique until a token size of 10 Kbytes. A token exceeds rarely such a size in a discrete media application, indeed, it transports only modifications performed by active sites in one ring turn. Furthermore, we do not take into account the cost of treatments involved by the consistency management, which is greater with symmetric protocols. Then, the technique used by the Pilgrim algorithm is well adapted to the transport of discrete media in cooperative applications. However, this algorithm contents itself with working on a fixed topology, that prevents it from adapting its behavior to the dynamicity of cooperative work communications.

In the Horus Total protocol [11] the token does not stay at one sequencer, nor does it cycle through all the members. It cycles through the current set of senders, so it can be adapted to the application requirements. In the Hybrid protocol [12] some processes order messages using a symmetric approach (passive mode), and others use a token-site approach (active mode).

The use of multiple types of transmission seems to be an efficient way to meet with cooperative applications requirements. It must be possible to change communication type both dynamically and efficiently. Some systems [13] build a new topology for each protocol change. This is costly, making the reconfiguration of group exchanges more limited. The use of different transmission types makes system integrity more difficult to maintain. Indeed, when two sites do not use the same communication protocol, it becomes difficult to manage the message order.

3. Consistency Protocol: The Pilgrim Algorithm

We tried to find a protocol that would reduce the number of exchanged messages through the network, and with which it would not be necessary to have a message for each reading or writing. We wanted the user of a cooperative application developed with CAliF not to feel the latency due to shared memory management.

Token ring protocols successively give a token to each processor. It is not necessary to request the token. The token ring model could become penalizing if the system is made up of many nodes some of which are inactive: the token continues to travel all around the ring and the time between two turns is wasted. For such a case, other studies have optimized the critical section management in distributed systems [14,15,16,17]. In cooperative work the number of sites is rather low. For example, it is not acceptable to edit a cooperative document with more than one hundred members.

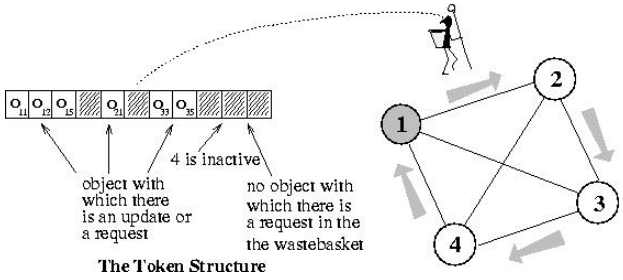


Fig. 3 Pilgrim large view.

In this situation, we developed an algorithm, which uses the token technique (Figure 3). But in our algorithm, the token is not a simple variable that is successively transmitted to each node of the ring, but a more complex data structure that contains the updates of shared data. These data are carried through the ring on the token, named *Pilgrim*. When the Pilgrim arrives at a site, it delivers updates from other sites and it recovers values, which have been modified during the last turn to carry them through the ring. We present this first protocol in four sections: the informal description, the model and proof with a finite state automaton and model checker Spin, performance and then optimization with overlapping technique.

3.1 Pilgrim Overview

First, we define the major characteristics of our protocol:

- This protocol allows the consistency management of distributed shared memory using a type of object-based replicated memory. **Each** object is only stored on each site which **uses** it.
- At time *t* each object has only one owner that can write on the object.
- The Pilgrim protocol is a simple writer protocol with regard to a single object but a multiple writers protocol with regard to all of the shared memory.
- During execution, only messages, which allow token circulation are sent: token sending and token acknowledgement.

Memory Representation

There is an instance of the memory on each site and the token carries memory updates.

- *The Structure of the Token*

The Pilgrim is an object array: the objects which need to be updated. In this array, objects from different sites are distinguished by separators.

- *Local Representation of the Shared Memory*

CAliF manages a replicated memory. An instance of the memory is on each site as a double link list of objects.

The Structure of Objects

An object o_{ij} owned by a site i is composed of four fields: one command, the parameter of the command, the rank j of the object in the list i and the data. The command can have one value among the following: N(null), D(delete object), C(change object ownership), Q(question to earn object ownership), A(accept to give up object ownership), R(refuse to give up object ownership). The parameter is optional, for example it indicates the number of the asking site. The rank is the identifier of the object o_{ij}. Data size depends on the cooperative application developed.

Operations on the Shared Memory

We can distinguish five cases:

- *Two writing cases*: An object owner wants to access the object in read-write. It can modify the object, but since it is not the owner, it has to earn the ownership first.
- *A single reading case*: If a site (owner or non-owner) wants to read the object, it reads the local value. Thus, a reading always returns the value of the local memory.
- *Two delete cases*: If a site wants to delete an object, it has to be the owner, otherwise, it has to first earn the ownership of the object.

Change of Ownership

When a site S_i wants to write on an object o of which it is not the owner, S_i has to send a request command for the object o via the token.

If the ownership of the object has already been requested by another site, this request is delayed. If no request for object o is in progress, the request command can be sent.

Not all requests are necessarily accepted. We have given priority to the active owner: if the owner writes on the object during the last turn of the token, it has to refuse the request. If not, it has to accept it. When site S_k's request for an ownership change on object o_{ij} is accepted, it is responsible for sending the owner change command via the token.

Data Deletion

In a cooperative system, several members work toward the same goal and they share data. If a member can delete all the data it owns, some useful data may unfortunately be deleted. That is why we added another list *WB* (*Wastebasket*) where deleted object are temporarily stored. It is still possible to recover an object stored on the *WB* list.

Token Treatment

When the token arrives, two types of operations are possible: operations on data, and operations on ownerships.

- *operations on data*:

When the token arrives on a site S_i, it is composed of the updates made from other sites. The data owned by S_i, if modified during the last turn of the token, are placed on the token that carries the new values.

- *operations on ownerships*:

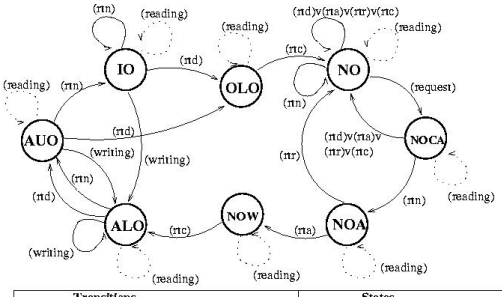
It is possible to modify the ownership of an object. All commands concerning ownership management are sent through the ring via the token.

When the token arrives, two treatment phases are needed: *the reading of the token* which allows the recovery of information from the other sites of the system; *the writing on the token* which allows updates, creations, deletes, ownership changes or answers about request commands to be sent to the other sites.

3.2 Proof and Validation

To validate our distributed algorithm, we have used two classical techniques: a finite state automaton to proof some Theorems and the model checker Spin to show that our protocol has no deadlock.

Pilgrim Properties



Transitions	States
(request)	ALO: Active Locked Owner
(reading)	AUO: Active Unlocked Owner
(writing)	IO: Inactive Owner
(rtn): receives token with no command	OLO: Owner which Loses the Ownership
(rtd): receives token with request command	NO: Non−Owner
(rtr): receives token with refuse command	NOCA: Non−Owner which Could Ask for the ownership
(rta): receives token with accept command	NOA: Non−Owner which Asks
(rtc): receives token with ownership change command	NOW: Non−Owner which Wins the ownership

Fig. 4 Pilgrim Finite State Automaton.

We propose a model of our algorithm with a finite state automaton made up of 8 states and 8 transitions (Figure 4).

Let *E* be the set of states containing *n* elements and *T* the set of transitions(events) containing *m* elements, the automaton edges represent an application from *ExT→E*. The structure created is theoretically a graph with *n.m edges*. In our example *n=8* and *m=8*, so the mathematic definition would impose *64 edges*. But some of them are impossible. If we strictly abide by this definition, we have to create a trash state which would receive all impossible edges. To simplify our graph we have not created this state. In [18], we discuss the consequences of each event. For each state we also state 10 rules defining the automaton, and especially the *impossible* events.

We demonstrated in [18] the theorem of mutual exclusion for writing on a shared object and a second theorem called the liveliness theorem. If we are working on local object lists, the first theorem (mutual exclusion) demonstrates that an object is only on one list (it has only one owner), while the second theorem (liveliness) demonstrates that an active object is necessarily on a list. We call an object active when it has been created and not deleted: in a cooperative drawing editor it is a visible object. In this case, an object always has an owner; otherwise it cannot be deleted or modified.

Mutual Exclusion Theorem:
The Pilgrim protocol guarantees mutual exclusion for writing on a shared object.
Liveliness Theorem:
An object always has an owner or has one after finite time.

Pilgrim Validation

To validate our protocol, we used the SPIN model checker [19]. We have built a model using PROMELA, SPIN's input language. Our work could be decomposed into two parts: the simple model and the complete one. Two types of message are carried through the ring : the Pilgrim and the Acknowledgement (figure 5).

- Simple model

The first model is composed of three sites. These sites share only one byte. At the initial state, node number 1 owns the variable. Each site may request to earn the ownership, and then can write in the variable.
When a site receives the token, it updates its local variable instance and eventually the token. When the token treatment is done, the site sends the token to its successor in the virtual ring and then sends an acknowledgement to its predecessor in the virtual ring. After a given amount of time, if no acknowledgement is received, the site sends the token to the successor of its successor and so on. Complet model

- The second model is more complete, it is composed of four sites which share three bytes. Functionalities are identical to those of the first model.

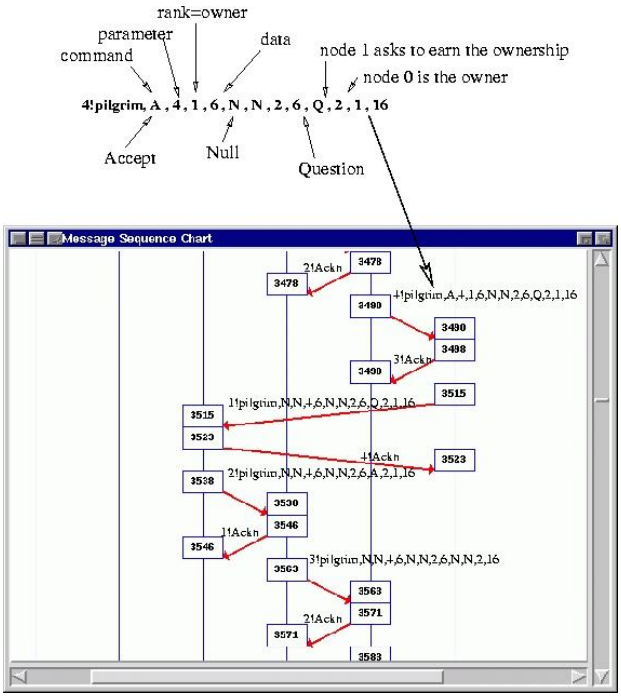


Fig. 5 Spin Simulation.

Figure 5, we show an abstract of the message sequence chart given by XSPIN. We can observe an owner changing phase: the owner of the third variable changes.

- At state 3515, node 1 receives the pilgrim, node 2 asks to earn the ownership of the third variable.
- At state 3523, node 1 accepts.
- At state 3563, node 2 changes the ownership of the third variable and it becomes the new owner of the third variable.

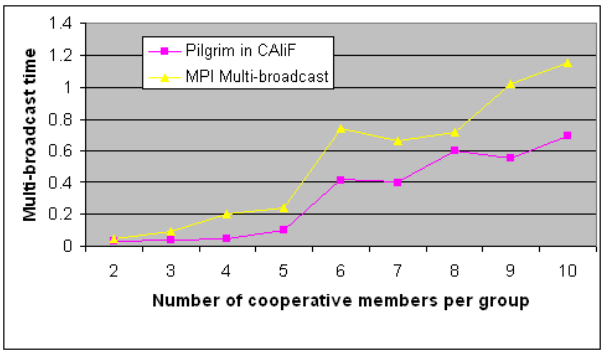


Fig. 6 Multi-broadcast.

These two models, verified using a maximum search depth = 1000, was found errorless. Therefore, we could think that our protocol was validated by SPIN.

3.3 Pilgrim performance

In order to determine the value of creating groups in an operation such as broadcasting, we have carried out tests on MPI.

Multi-broadcast is an important operation in a cooperative application because all participating members broadcast simultaneously.

This operation is very expensive, and the user should not feel the latency of its management. In [20], reaction time, ensuring the clarity of the operation, is defined as less than a second.

On the curves shown in Figure 6, multi-broadcast is implemented using MPI routines and the Pilgrim algorithm over MPI. Each node broadcasts 512 bytes. We can observe that if there are more than 10 participating members, broadcasting time exceeds one second. In this case, it would be useful to create restricted cooperation groups in which this multi-broadcast operation would be more efficient. Indeed, the network load decreases with this technique.

Partitioning into groups enables this multi-broadcast operation to increase its performances and makes the operation clear to the users.

3.4 Pilgrim Optimization: Overlapping Technique

Description of the Optimization Implementation

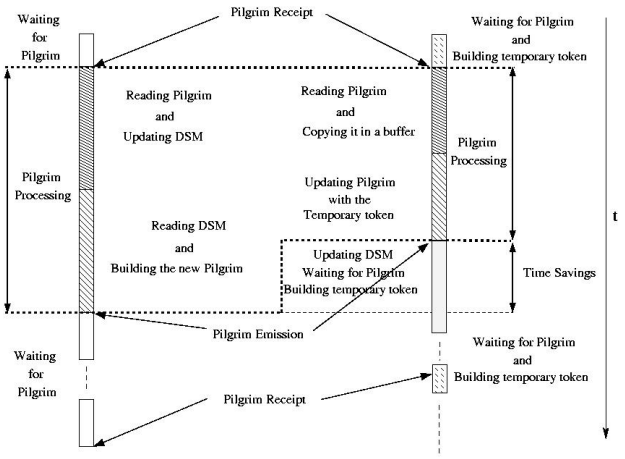


Fig. 7 Optimizations of Pilgrim Processing.

We have implemented the Pilgrim algorithm over CAlifCom. This algorithm use a virtual ring to broadcast information on the DSM (Distributed Shared Memory) and to manage its consistency. In such an algorithm, the latency between an emission and a reception is proportional to the number of sites, the communication times and the Pilgrim token processing. In our implementation we have focused on reducing of this processing.

A conventional approach is shown in the left part of Figure 7. We see that implementing the Pilgrim algorithm can be broken down into several stages:

- Pilgrim receipt
- Pilgrim processing
  - reading the Pilgrim token and updating the DSM with objects it contained. When a modified object is met in the Pilgrim token it is updated in the local DSM.
  - reading the DSM to list all objects modified since the last receipt of the Pilgrim and updating the new Pilgrim with the above list.
- Pilgrim emission

Scanning the Distributed Shared Memory can be a long operation if there is a great number of objects to examine. We can save time by reducing the DSM analysis and by postponing its update: that is the *overlapping* technique. We can see in the right part of Figure 7 that some optimizations are possible for Pilgrim processing:

- The DSM can be updated after the token emission, so this processing is performed between two Pilgrim receipts. When the Pilgrim arrives, the updates it contains are copied in a buffer, and used later.
- Building the new Pilgrim can also be optimized. We use a temporary local Pilgrim in which we continually put the objects modified on the local site since the last receipt of the token. So, when it arrives, a DSM scanning is not necessary, all changes are already available in this temporary token. It is a fast operation, we just need to copy the temporary list in the real Pilgrim. With this method we can build a new temporary Pilgrim during the token processing.

In Figure 7 we suppose that the time between two Pilgrim receipts is longer than the token processing time. If it is not the case, the processing is performed concurrently with the DSM update. So the efficiency of this technique grows with the number of cooperative sites, which increases the latency time and allows the DSM to be updated before a new Pilgrim receipt.

In our implementation we try to hide from the cooperative application users the operations performed to carry out shared object consistency. Operations are buffered, and accesses to DSM objects synchronized. So, there is no latency feel due to pilgrim processing.

Time savings generated by this implementation is shown in Figure 7.

Tests with optimized Pilgrim

We performed all tests on 166 Pentium processors connected by a 10Mb Ethernet network. We want to show that CAliF allows us to develop efficient cooperative applications without using very powerful machines.

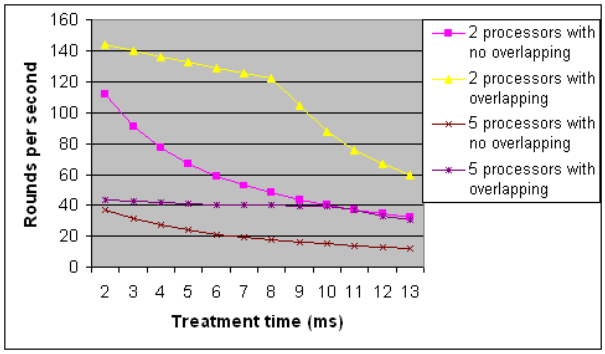


Fig. 8 Overlapping.

Figure 8 shows the difference between our implementation and the conventional approach to Pilgrim processing (described in the previous section).

The comparison is made for 2 and 5 machines. We see that for the conventional approach, the number of rounds performed by the Pilgrim in one second decreases proportionally to the size of DSM. Indeed, when the number of shared objects increases, the time needed for scanning all of them also increases.

When we used the overlapping technique we implemented, application performance are improved. Moreover, when the DSM update time is shorter than the time between two Pilgrim receipts, DSM size increase is not very detrimental to performance. As we can see in Figure 8 the difference between the two techniques grows with DSM size until it reaches a given number of objects. This point corresponds to the critical size, beyond which the amount of time needed to perform the DSM update exceeds Pilgrim travel time. So it arrives during the memory update and cannot be totally processed before the end of this operation, due to consistency problems. Therefore, the difference between both techniques is smaller, but our implementation is still better.

We observe that our implementation is efficient: for very large DSM sizes the number of rounds per second never falls below 12, and this size involves a very large number of shared objects. This is still an acceptable time for a cooperative application, since users cannot feel such a latency.

It seems important to indicate our implementation performance according to Pilgrim size. This size depends on to the cooperative application type. A shared text editor requires a smaller Pilgrim than a cooperative drawing application.

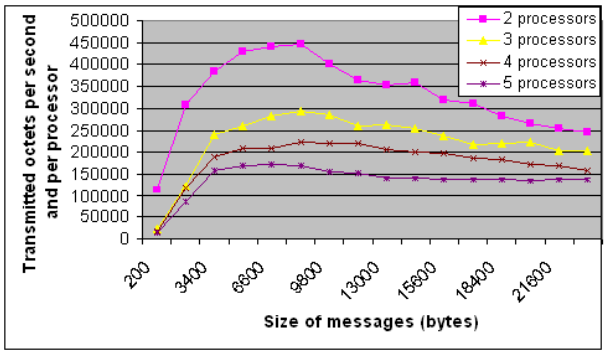


Fig. 9 Optimal Pilgrim Size.

In Figure 9, we see that the optimal Pilgrim size is between 5000 and 10000 bytes. But for 5 cooperative members, performance does not decrease proportionally to Pilgrim size. Pilgrim implementation over CAliFCom is efficient for all types of discrete media applications, where the size of objects is not too great and no synchronization between sites is needed.

However, we cannot efficiently manage continuous media applications which require great outflows and synchronization methods. But continuous media streams do not need consistency management. Therefore, several algorithms can be combined in such applications:

- Pilgrim algorithm for consistency management of shared objects;
- Continuous media management algorithm for audio and video.

4. Communication Protocol: The Chameleon Algorithm

4.1 Description

CAliF Multimedia uses distributed shared memory to manage the consistency of discrete media. The main part of the platform is the communication service. It allows cooperating members to broadcast their information.

All the modified objects of the distributed shared memory are transported by way of this service. The consistency service interacts with this service to maintain the integrity of the application.

We chose to develop a new token strategy based algorithm. The originality of these algorithm, named *Chameleon*, is to allow the virtual topology to be reconfigured using two communication techniques: a rotating sequencer for one site and a symmetric approach

for another, for example. Thus, the token only visits the active sites. Another important characteristic of this algorithm is that the representation of the virtual topology is a distributed shared object and is transported by the token.

Ring topology is more efficient than a full connected network topology for a multi-broadcast operation; however this tendency is reversed when the number of active sites decreases. So, we use the second method (symmetric) for the sites which do not participate actively in cooperative work to complete the ring which links the active sites.

4.2 Site States

Actors of a cooperative application can have different roles and rights, thus implying that sites can have different states:

- *Producer-Consumer* (**PCo**): The site modifies the shared data. It receives information from the other producer sites and sends the result of its operations to the next producer through the token.
- *Tutor Producer-Consumer* (**TPCo**): The site modifies the shared data and sends the results of its operations to its next producer and to the sites for which it is the tutor. A tutor is a member of the virtual ring, whereas a *tutored* site is not a member of the ring but depends on a tutor.
- *Simple Consumer or Tutored site* (**SCo**): The site does not modify the shared data; it only receives information from its tutor via an inactive copy of the token. It never broadcasts a token.
- *Simple Producer* (**SP**): This site is the only producer.

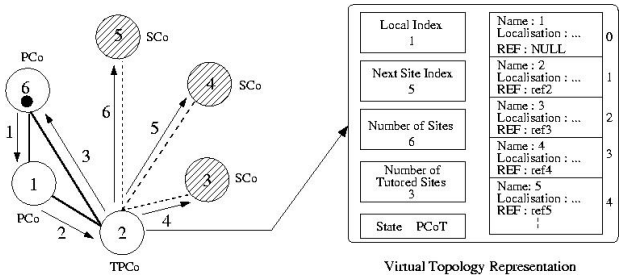


Fig. 10 Token Treatment.

The programmer of a cooperative application has to choose a policy for the management of the site states. For example, if a PCo site is inactive for a given time it becomes SCo. Or, for a given site it may be impossible to be producer: an observer site will always remain SCo.

The evolution of virtual topology is dynamic and does not require reconstruction, so it is not very costly. It is based on a ring topology, but it can evolve either towards a centralized system if there is a *Simple Producer*, or towards the use of several techniques: a ring between *n* sites and a centralized system from a node. The Chameleon algorithm manages the membership of a site whatever its state may be.

4.3 Management of active and inactive sites

The Token

The token is not a simple tool allowing a site to broadcast, it contains the virtual topology representation. The consistency of this topology has to be maintained, to ensure that all the sites have the same view of the system. The nature and the treatment of the token depend on the state of the sites which receive it.

Figure 10 shows the token treatment for SCo and Producer sites. Site 2 is the tutor of sites 3,4 and 5. Each site has a local copy of the virtual topology representation. It knows its local index, the index of the next producer, its number of tutored sites, the total number of sites, its state, and it has an array containing the description and references of the sites corresponding to these indexes.

```
BEGIN
receipt of the token
case state of local site of :
PCo:
      consume token (update of DSM, update of virtual topology ...)
      modify token  (the local modifications are put on the token)
      send token to next site
SCo:
      consume token
TPCo:
      consume token
      modify token
      send token to next producer        (figure 10: stage 3)
      for i = 1 to NbTutored do
           send token to tutored[i]      (figure 10: stages 4, 5, 6)
END.
```

Fig. 11 Broadcast Algorithm.

In the simplified algorithm (Figure 11) we see how the token is treated when there is no change of state, no membership request and no video channel creation request.

We can see that when a TPCo site broadcasts the token, it begins by sending it to the next producer. As broadcast is a blocking operation, the TPCo site has to wait until it is terminated. After that, the token is sent in parallel by the TPCo site to its tutored site(s) and by its next producer to another producer. So, at the same time, two sites are treating the token. There is no problem of consistency management because one of these two sites is SCo and cannot modify the distributed shared memory.

Several events can lead SCo sites to send messages: for example, they may have to become PCo if they have a modification to carry out, or they may have to manage the membership of a new site. In such a case, the SCo site can send messages to its tutor which stocks and treats them when it is in possession of the token. In this way, the

consistency of the virtual topology as well as the order and integrity of sent messages are maintained.

Change of state

The policy changing site states is chosen by the cooperative application designer. When the change of state criterion is reached, the virtual topology has to be modified. This modification depends on the state of the site which causes the change.

When a PCo site has to become SCo, it waits for the token, then it modifies the virtual topology contained in the token and routes this token which will broadcast this change of state. The modification of the virtual topology on the token is performed by a calculation function which builds a new topology according to the current topology and to all waiting requests (membership, departure...). If this site wants to become a producer again, it has to send a request which will not be transported by the token (a SCo site cannot send a token); this request will be treated by its tutor. Then, if a TPCo (tutor) site becomes Simple Consumer, it has to:

- find new tutor(s) for its tutored site(s);
- find a tutor for itself;
- modify the virtual topology on the token;
- send the token to the next producer site;
- send the token to its tutored site(s).

Each modification of a site state leads to a dynamic reconfiguration of the virtual topology. In this case several policies are possible for the new configuration calculation. For example, the positioning of the tutored sites can be either centralized (a site is the tutor of all the SCo) or balanced (a site is the new tutor if it has less tutored sites than the other producers). These policies are chosen by the programmer according to the type of application he is building.

4.4 Fault Tolerance

We have introduced additional controls to limit the errors in case of fault on a site and to obtain a fault tolerant system.

With a system composed of producer sites, when a token is sent, the sender site waits for an acknowledgement which allows the emitter to control the good circulation of the token on the virtual ring. It means that the successor has received the token and also that it has treated this one. The successor emits the acknowledgement only when it has treated the token and just before to send it to its own successor.

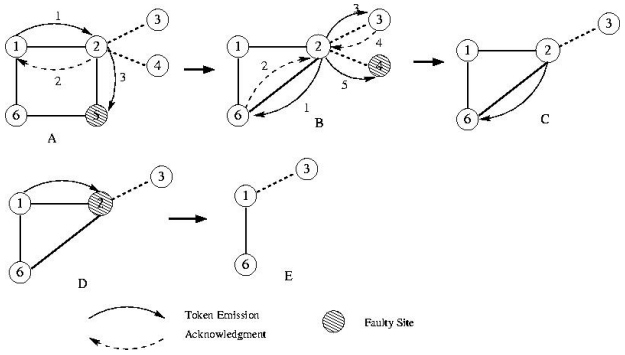


Fig. 12 Faults Treatment.

The delay overrun is calculated according to the token size and to the known characteristics of the network. We deal only with site faults, not with network problems. When the time limit between an emission and a reception is reached, we suppose that the receptor site is faulty. The predecessor site of the faulty one is then in charge of restoring the objects consistency, in particular if there are some ownership notions. We work on a replicated shared memory, thus, all sites have a local copy of the objects. Furthermore, a new communication topology has to be calculated and installed. Then, the protocol takes off the faulty site and gives the token to its successor.

The faults treatment becomes more complex when Simple Consumer sites are involved. We distinguish several cases:

- If the producer which follows the tutor is faulty (5), the new topology has to be calculated and the token has to be sent to the new following producer (6) but also to the tutored sites (3 and 4) in order to inform them of the new topology (Figure 12 A-B),
- If a tutored site is faulty, its tutor waits the next turn to correct and broadcast the new topology. The tutor can not treat a new token as it has not received all the acknowledgements it was waiting for, (Figure 12 B-C),
- If the tutor site is faulty, its predecessor launches the calculation of a new communication topology to remove this site and to find a new tutor for the Simple Consumer sites (Figure 12 D-E).

4.5 Chameleon Proof and Validation

We have used the SPIN [HOL91] model checker as validation tool, in particular the X version: Xspin.

Spin allows us to detect dead-locks or possibly bad assertions. SPIN algorithms are expressed in PROMELA which is a no-determinist language. We have proceeded in 3 stages on a 6 sites model [Gar01]:

- a first simplified version of the Chameleon with a static virtual architecture. In this version, 6 processes

exchange the token: 3 Tutor-Producer-Consumer sites (TPCo) and 3 Simple-Consumer sites (Sco). The aim is to put in place communications in the virtual architecture. On Figure 13 we can see an execution sequence of this first version. On this figure, the labels of the modeled architecture (3!token...) correspond to the sending of the token sequences. We can observe their order on the Xspin simulation. The arrows represent the token emission, the first column match with site 1, the second with site 2...

- a second PROMELA version of the Chameleon algorithm proposes a system where the architecture is dynamic. It is then interesting to verify that there is no dead-lock and that each site can pass in one of the state Pco, TPCo, Sco or SP.
- In a last version we have tested the faults treatment. When a site sends the token, if it does not receive an acknowledgement, it deletes the faulty site, it becomes the new tutor of the sites tutored by the deleted one, and then it transmits the token to its new successor.

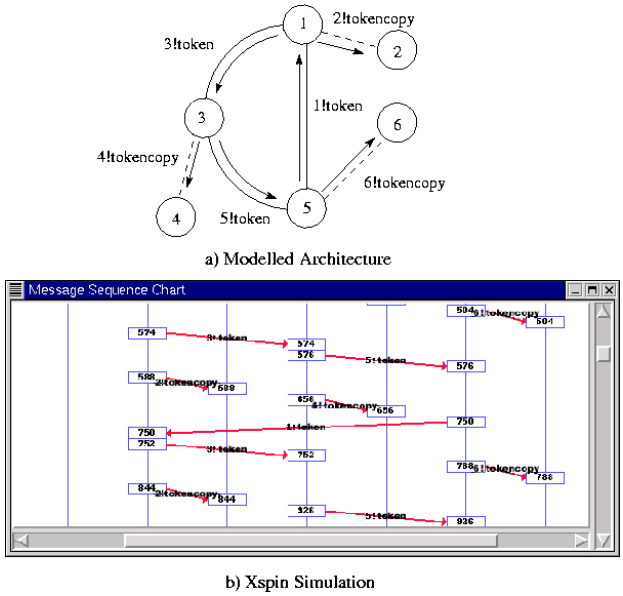


Fig. 13 Xspin and Modelling.

For the last PROMELA version, the delay overrun has been managed as a message: if a site is faulty when it receives the message, it sends to its predecessor in the logical ring a timeout message instead of an Ackn message. The different versions of the Chameleon protocol we have tested have not cause some errors. We have done these verifications with a 10000 depth.

4.6 Chameleon Performance

We have implemented the Chameleon algorithm in the CAliF Multimedia group communication service. This algorithm allows us to manage communications, consistency of the virtual topology representation on each site, and the establishment of transport channels for continuous media. The communication service of our platform is implemented with Orbacus 3.2.1 [21] and C++. The tests are performed on a 100 Mbit LAN with a PentiumIII PC cluster.

Communication

A first test presents gains obtained when a PCo site becomes Sco, and allows us to see the impact of site speed on the group behavior (Figure 14). In this test sites called Fast sites are equipped with 100Mb/s Ethernet cards and slow sites with 10Mb/s cards.

For the start configuration with 4 fast active sites (PCo), we see that the time taken by the token to perform 3000 turns increases with its size. For a token containing 2500 elements of 4 bytes (10 Kbytes) it is 12.5 seconds. When one of the active sites becomes inactive (SCo), performance is improved from 8 percent for a 10-element token to 20 percent for a 2500-elements token. The gain depends on the token treatment time. If this time is great (in terms of the token size), the obtained gain will also be greater because the treatment of the token is performed in parallel on two sites.

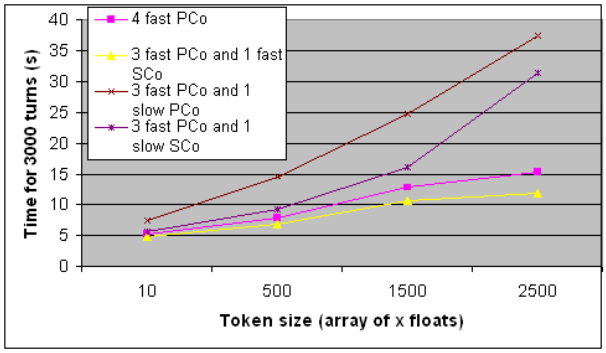


Fig. 14 Introduction of different kinds of salve sites.

For the start configuration with 3 fast PCo sites and 1 slow PCo, it is efficient to place the slow site in SCo state for a token size below 1500 elements (6Kbytes). Beyond that point, performance decrease due to the congestion point represented by the slow site. Indeed, the tutor of the slow site has to wait that this last has performed its treatment before receiving the new token.

These tests give good results considering that most cooperative applications use a small (less than 5 Kbytes) token to manage discrete media consistency. Group auto-organization needs to answer several questions, "when and how to modify group topology?...", and needs also to

know the impact of such a modification according to site power, active/inactive active sites ratio. So, we can propose a group service on top of CORBA, which offers functions to manage operations in the group, but some parameters have to be adjusted to meet particular cooperative application requirements.

These tests give good results considering that most cooperative applications use a small (less than 5 Kbytes) token to manage discrete media consistency.

Placement

We can study the different ways to place tutored sites. The two main policies are represented on Figure 15:

- In the first case (a), a producer site is appointed to be the tutor of all SCo sites.
- In the second case (b), the new tutored sites are allocated to low loaded producers in order to balance the treatments.

It is also possible to use intermediary solutions: random for example.

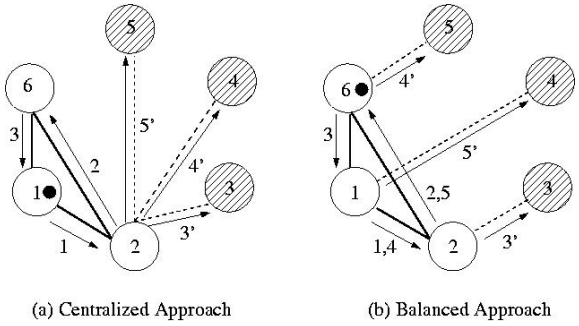


Fig. 15 Tutored Sites Placement.

We see on Figure 15 that the balancing of tutored sites is logically better. Indeed, in 5 logical stages, the token which starts up from site number 1 performs a complete ring turn and then is located on site number 6, while in the case of a centralization of tutored sites it only performs one ring turn.

The way to place tutored sites has to be chosen by the cooperative application programmer according to its needs.

Our tests show (Figure 16) the effect of the number of tutored sites and the choice of their tutors on the performance. We work with a virtual topology with 6 sites, and can see that the increase of the SCo site number improves performance. The optimal number is not, however, the maximum one. Indeed, when it becomes greater than the number of producer sites, there is a problem of congestion. The token comes back to a TPCo site before the tutored sites have received and treated their previous copy. As broadcast is a blocking operation (for fault tolerance and synchronization), the TPCo site has to wait for the SCo site release. If multicast is not implemented in the system used, the balancing technique is better than the centralized one. With 3 TCPo and 3 SCo sites the centralized technique is 25 percent slower than with balanced tutors. Indeed, when there is only one tutor the TPCo site have to wait one stage more for its tutored site release.

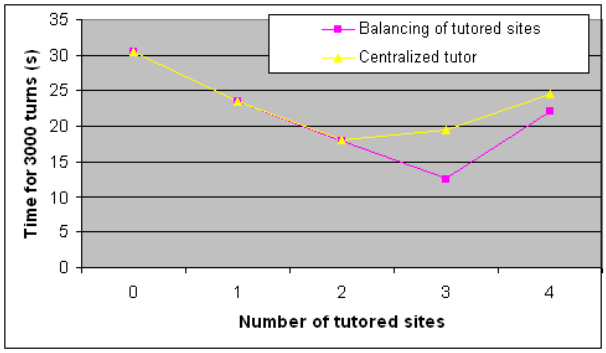


Fig. 16 Balancing of Tutored Sites.

These tests are made to help the cooperative application programmer using CAliF Multimedia to choose the best policy for the change of site criterion or for tutor balancing. This policy is linked to the type of application to be written and to the available technology: a centralized tutor is better if multicast exists on the system used.

5. Conclusion and Further Work

In this paper we have presented, Pilgrim and Chameleon algorithms which allow consistency management in collaborative work. Indeed, in this type of applications, several users handle simultaneously shared objects. We have also presented the need of adaptability of this type of system, due to work load fluctuations.

Some existing group communication protocols allow the transmission mode to evolve dynamically, other algorithms offer a distributed shared memory consistency management support. The Chameleon algorithm combines these two characteristics. It is used in the CAliF Multimedia group communication service. It manages communications and consistency of the virtual topology representation. It offers a support for distributed shared memory consistency management. It also manages the connections for continuous media transport. Tests show that it is efficient, even on high speed networks because token treatment time is not linked to transmission time. Users of this service have to take into account time savings obtained with the application of different policies and the

type of application they want to build: telemedicine or cooperative telemaintenance have not the same cooperation criteria.

Pilgrim algorithm allows us to manage consistency of distributed shared memory. It uses the principle of property on shared objects. Some optimizations have allow us to implement a better version of this protocol on a token ring topology. This type of fixed topology limits its efficiency for collaborative applications. A complementary use of the Chameleon algorithm is an interesting alternative to this problem, it allows us to take advantage of robustness an performances of these two combined algorithms.

We are validating the Chameleon algorithm using a finite state automaton and a Model Checker. We also are implementing a telemedicine and a telemaintenace [22] application using CAliF Multimedia.

References

[1] JW. Hong, YM. Shin, MS. Kim, JY. Kim and YH. Suh. Design and Implementation of a Distributed Multimedia Collaborative Environment. *Special Issue on Multimedia Collaborative Environments of Cluster Computing: Networks Software Tools, and Applications, Baltzer Science*, Fall, 1998.

[2] H. Guyennet, E. Garcia and J-C. Lapayre. Multimedia Integration in Cooperative Work. *Proceedings of the 5th International Conference on Information Systems Analysis, ISAS'99*, Orlando, USA, pages 442-447, July, 1999.

[3] E. Garcia, H. Guyennet, J-C. Lapayre and N. Zerhouni. A new Industrial Cooperative tele-maintenance Platform. *Special Issue of Computers & Industrial Engineering, Elsevier Science*, vol. 46(4), pages 851-864, 2004.

[4] Object Management Group. Real-Time CORBA, joint Revised Submission. *OMG TC Document orbos*, March, 1999.

[5] Object Management Group. Control and Management of Audio/Video Streams. *OMG, CORBA Telecom*, June, 1998.

[6] R.K. Budhia. Performance Engineering of group communication protocols. *Thesis of the University of California*, August, 1997.

[7] H. Guyennet, J-C. Lapayre and M.Trehel. A New Consistency Protocol Implemented in the CAliF System. *IEEE Proceedings of the HiPC International Conference*, Bangladore, India, December 1997.

[8] E. Garcia, J-C. Lapayre and G. David. Pilgrim Performance over a New CAliF Communication Layer. *IEEE Proceedings of the The International Conference on Parallel and Distributed Systems, ICPADS'00*, Iwate, Japan, August, 2000.

[9] B. Nitzberg and V. Lo. Distributed Shared Memory: a survey of issues and algorithms. *IEEE computer*, vol. 24(8), pages 52-60, September, 1991.

[10] L. Brunie, L. Lefevre and O. Reymann. Execution Analysis of DSM Applications: A Distributed and Scalable Approach. *ACM Proceedings of the the SIGMETRICS Symposium on Parallel and Distributed Tools SPDT'96*, May, 1996.

[11] R. Van Renesse, K.P. Birma and S. Maffeis. Horus: A flexible group communication system. *Communications of the ACM*, vol. 39(4), pages 76-83, April, 1996.

[12] L.E.T. Rodrigues, H. Fonseca and P. Verissimo. Totally ordered multicast in large-scale systems. *Proceedings of the 16th International Conference on Distributed Computing Systems*, Hong-Kong, pages 503-510, May, 1996.

[13] D. Brattli. A Survey of Dynamic Configurable Protocol Stacks. *http://www.cs.uit.no/~dagb/dynamic-protocols/dynamic-protocols.html, University of Tromso, Faculty of Science, Department of Computer Science*, Norway, 1998.

[14] L. Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *Communication ACM*, vol. 21, pages 558-565, 1978.

[15] A.K. Agrawala and G. Ricart. An Optimal Algorithm for Mutual Exclusion in Computer Networks. *Communication ACM*, vol. 24, pages 9-17, 1981.

[16] M. Maekawa. A $\sqrt{n}$ Algorithm for Mutual Exclusion in Decentralized Systems. *ACM Transactions on Computer Systems*, vol. 3(2), pages 145-159, 1985.

[17] M. Naimi and M. Tréhel. An Improvement of the Log(n) Distributed Algorithm of Mutual Exclusion. *IEEE Proceedings of the 7th International Conference on Distributed Computing Systems*, pages 371-375, 1987.

[18] H. Guyennet, J-C. Lapayre and M. Tréhel. The Pilgrim Consistency Protocol for Distributed Shared Memory. *Technical Report 01-R-02*, Besançon, France, June, 2002.

[19] G. Holzmann. Design and Validation of Computer Protocols. *Prentice Hall International Editions*, 1991.

[20] C. Ruhla. Pédagogie par l'image et le son. *Association belge des professeurs de sciences francophones Congress*, 1990.

[21] OOC. ORBACUS for C++ and Java. *http://www.ooc.com/ob*, 1999.

[22] J. Szymanski and al. PROTEUS - a European Initiative for E-Maintenance Platform Development. *9th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA2003*, Lisbonne, Portugal,16-19 September, 2003.