Optimistic Access Control Model for Collaborative Editing Systems

Presented by Asma Cherif

Cassis Team, Loria

November 29, 2010

Introduction

Distributed Collaborative Editors (DCE) provide computer support for modifying simultaneously shared documents:

- articles,
- wiki pages and
- programming source code

by dispersed users.

Example

- Google Docs
- Coword, Copowerpoint
- *etc*.

DCE Requirements

DCE must consider human factors

- High responsiveness
- High concurrency
- Consistency
- Decentralized coordination
- Scalability

Examples of Real-time Collaborative Editors

- centralized (global and unique order of execution):
  - SOCT4 [Vidot00], GOT [Sun98]
    (-) Client-Server architecture
- decentralized (arbitrary order of execution):
  - adOPTed [Ressel:96], SOCT2 [Suleiman98], GOTO [Sun98], and SDT [Li04]
    (-) fixed number of users (use of state vectors to detect causality relationship)
  - OPTIC [Imine08]: dynamic groups

Objective

Build a generic access control layer:

- suites existing editing solutions characterized by
  - replication
  - log usage
  - undo procedure
- responds to DCEs requirements

without adding overhead

Challenges

- Ensuring security is a challenging problem in these applications:
  - Balancing the computing goals of collaboration and access control to shared information,
  - Users can join and leave at any time so we have to allow for dynamic change of access rights (scalability requirement).
  - High performance: access rights management should be replicated (responsiveness requirement).

Our Coordination Model

Our coordination model is composed by two shared and replicated objects:

- Data Object
- Policy Object

Shared Data Object

The data object is modeled by a list of

- characters, paragraphs, pages, pixels, XML nodes, ...

Operations altering data object state: cooperative operations

Example

- Insert
- Delete
- Update

Shared Policy Object

We specify an authorization policy by three sets:

- set of subjects (*i.e.* users)
- set of objects
- set of access rights

Policy

A policy is a function that maps a set of subjects and a set of objects to a set of signed rights.

Shared Policy Object

Authorization rule

$$(\{s_1, s_2, \ldots, s_n\}, \{o_1, o_2, \ldots, o_p\}, \{r_1, r_2, \ldots, r_q\}, +/-)$$

Ownership

each user s_i has its own policy P_i to administrate his objects $\Rrightarrow$ Administration distribution

Shared Policy Object

Two kinds of administrative operations

- AddAuth
- DelAuth

Global Policy Object

The set of all owner policies

Collaboration Protocol

1. Check a local operation against the appropriate owner policy object.

Collaboration Protocol

1. Check a local operation against the appropriate owner policy object.
2. Once granted and executed, local operations are broadcast to other users.

Collaboration Protocol

1. Check a local operation against the appropriate owner policy object.
2. Once granted and executed, local operations are broadcast to other users.
3. When received by a user, remote operations are checked against the appropriate owner policy object.

Collaboration Protocol

1. Check a local operation against the appropriate owner policy object.
2. Once granted and executed, local operations are broadcast to other users.
3. When received by a user, remote operations are checked against the appropriate owner policy object.
4. When an administrator modifies its owner policy object, modifications are sent to other users in order to update their local copies.

Is convergence maintained in presence of administrative operations

Consistency and Security Issues

- Out-of-order Execution of Cooperative Operations.
- Out-of-order Execution of Cooperative and Administrative Operations.

Out-of-order Execution of Cooperative Operations

Cooperative operations may arrive in different order at different sites.

Issue

This can lead to undesirable situations in collaborative editors: data divergence.

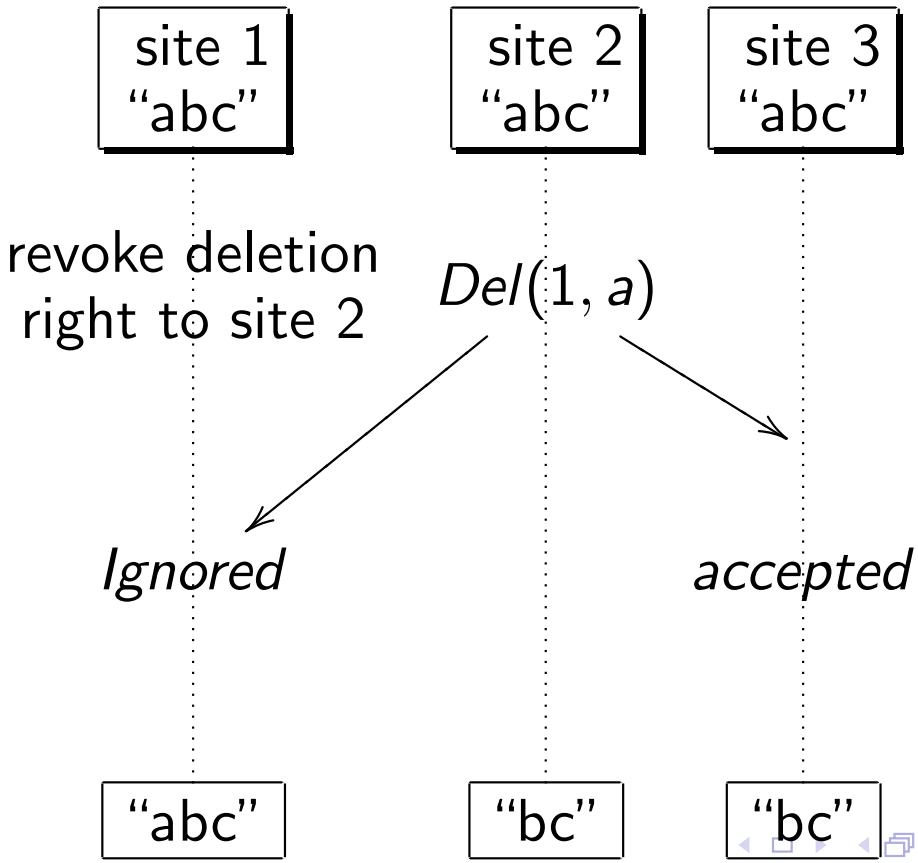
Use OT approach to overcome this problem

Out-of-order Execution of Cooperative and Administrative Operations

Performing cooperative and administrative operations in different orders at every user site :

- Security holes
- Data divergence

Divergence caused by introducing administrative operations (Scenario1)



Divergence caused by introducing administrative operations (Scenario1)

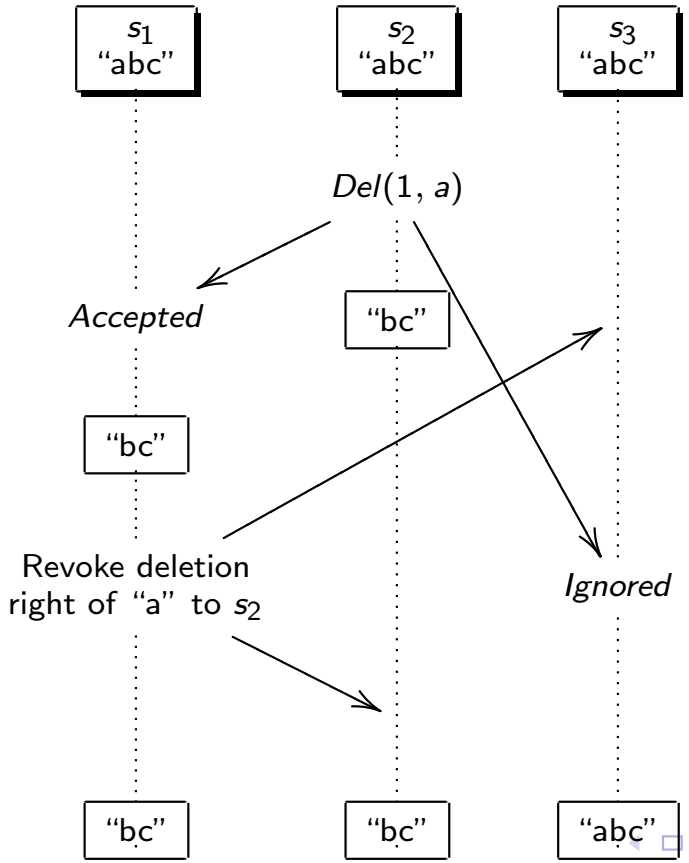
Issue

Detect causality between administrative and cooperative operations?

Solution

- Give priority to administrative operations
- Temporary policy violation: optimistic access control
- Recover the last correct state : Undo process

Divergence caused by introducing administrative operations (Scenario 2)



Divergence caused by introducing administrative operations (Scenario 2)

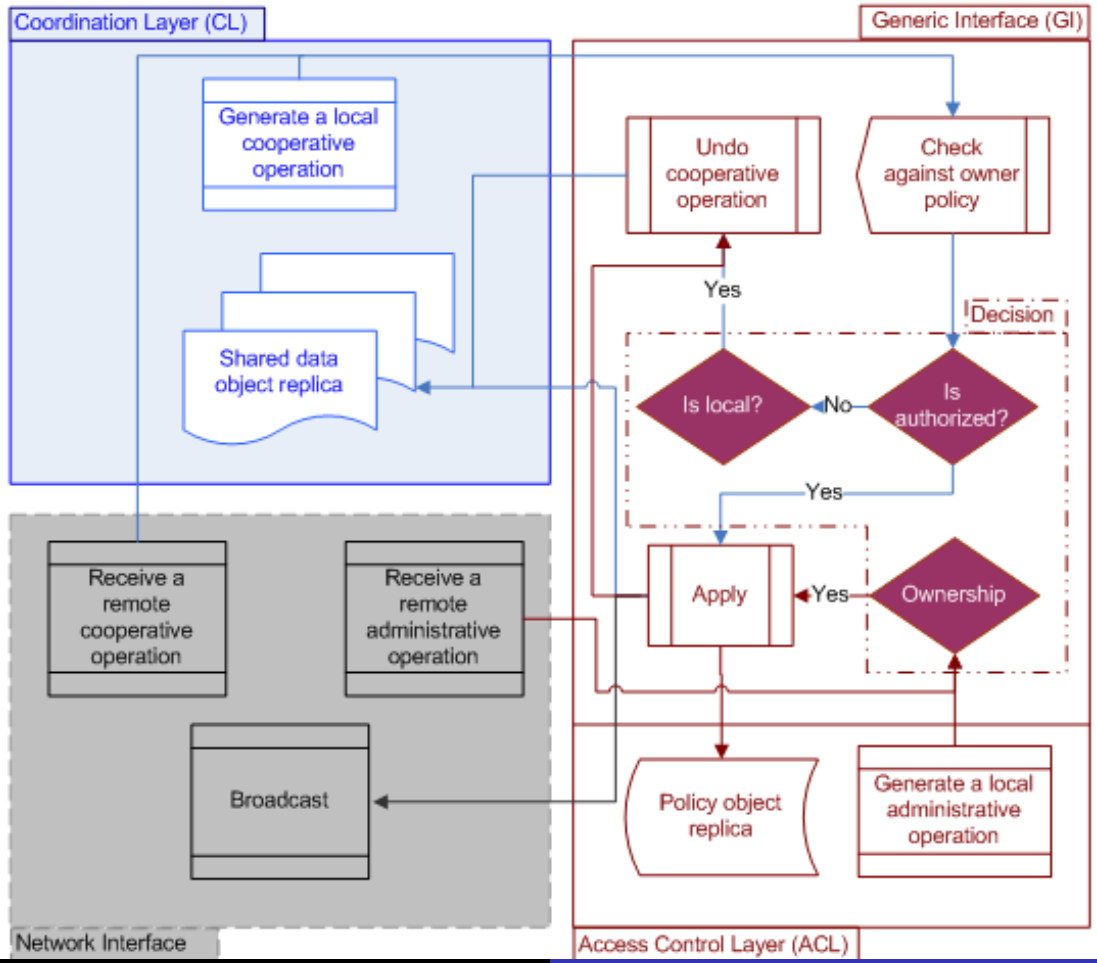
Issue

Detect causality between an administrative and cooperative operation?

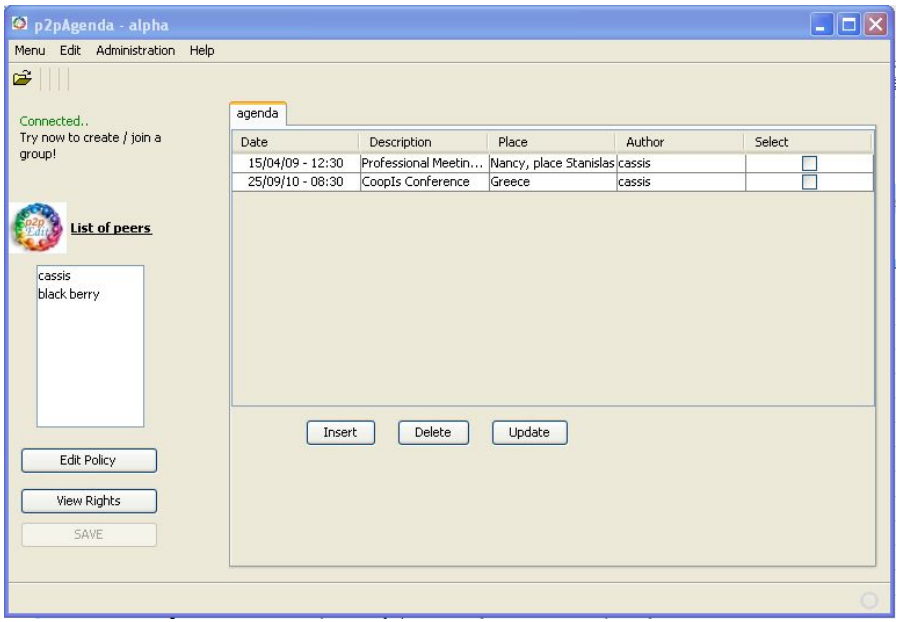
Solution

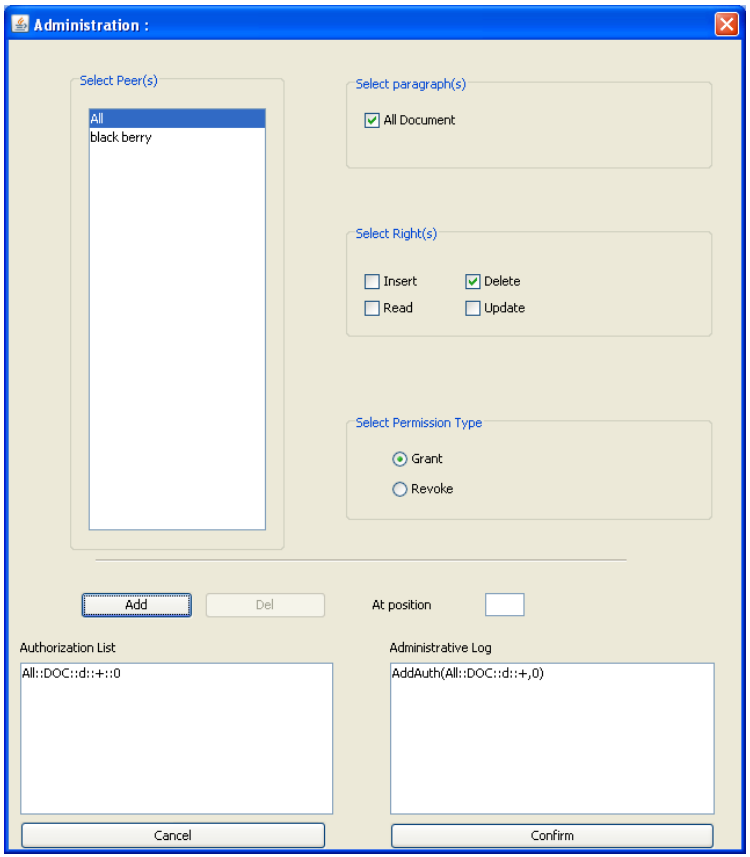
- Each modification seen by the object administrator is considered as valid

Access Control Flow Chart



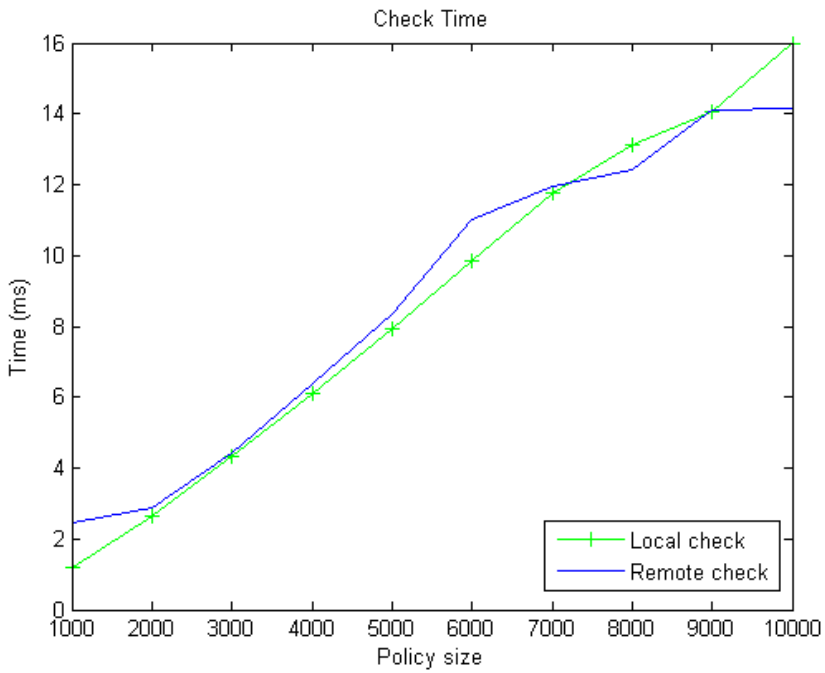
Implementation: p2pAgenda (shared calendar)





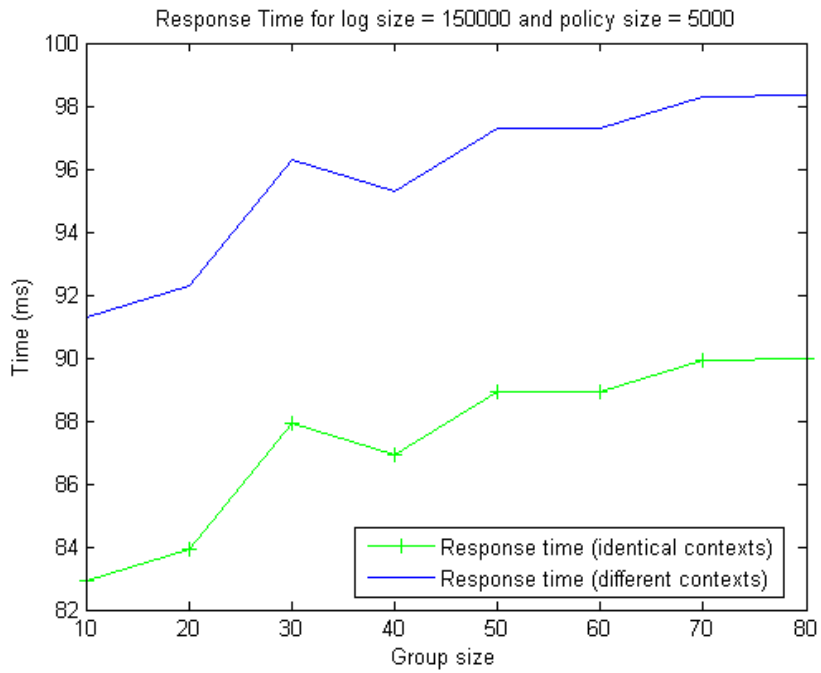
Evaluation: response time

- Worst case: last rule matches the cooperative operation.
- check time is linear
- $\leq$ 16 ms (local operations)
- $\leq$ 14 ms (remote operations)
- use of Hashset



Evaluation: response time

- Worst case: integrate a remote insert.
- Response time is $\leq 100ms$ for
- log size = 150000 (100%*ins*)
- Owner policy size = 5000
- peers number= 80



A Garbage Collector for Collaborative Editors

Motivations

- Mobile Devices such as PDAs and cell phones are becoming more and more pervasive (PDAs, iPhones, ...).
- Several works try to integrate desktop applications on these devices.

Challenges

- mobile devices: low space memory, slow wireless connections
- collaborative editing applications: log usage (convergence) $\Longrightarrow$ large storage space
- Big logs $\Longrightarrow$ high delays

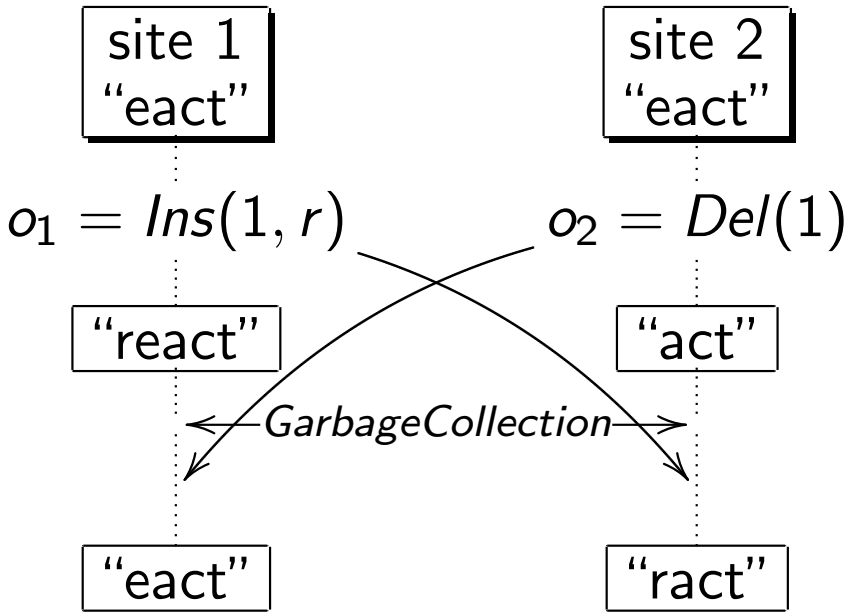
Principle

- Garbage $\Longrightarrow$ global view on the collaborative system (clean while maintaining convergence)
- Existing solution [Sun98]:
  - state vector usage $\Longrightarrow$ scalability issue
  - identical logs

Key idea

Build a global view on the collaboration state in a scalable fashion.

Divergence caused by garbage collection applied on different contexts



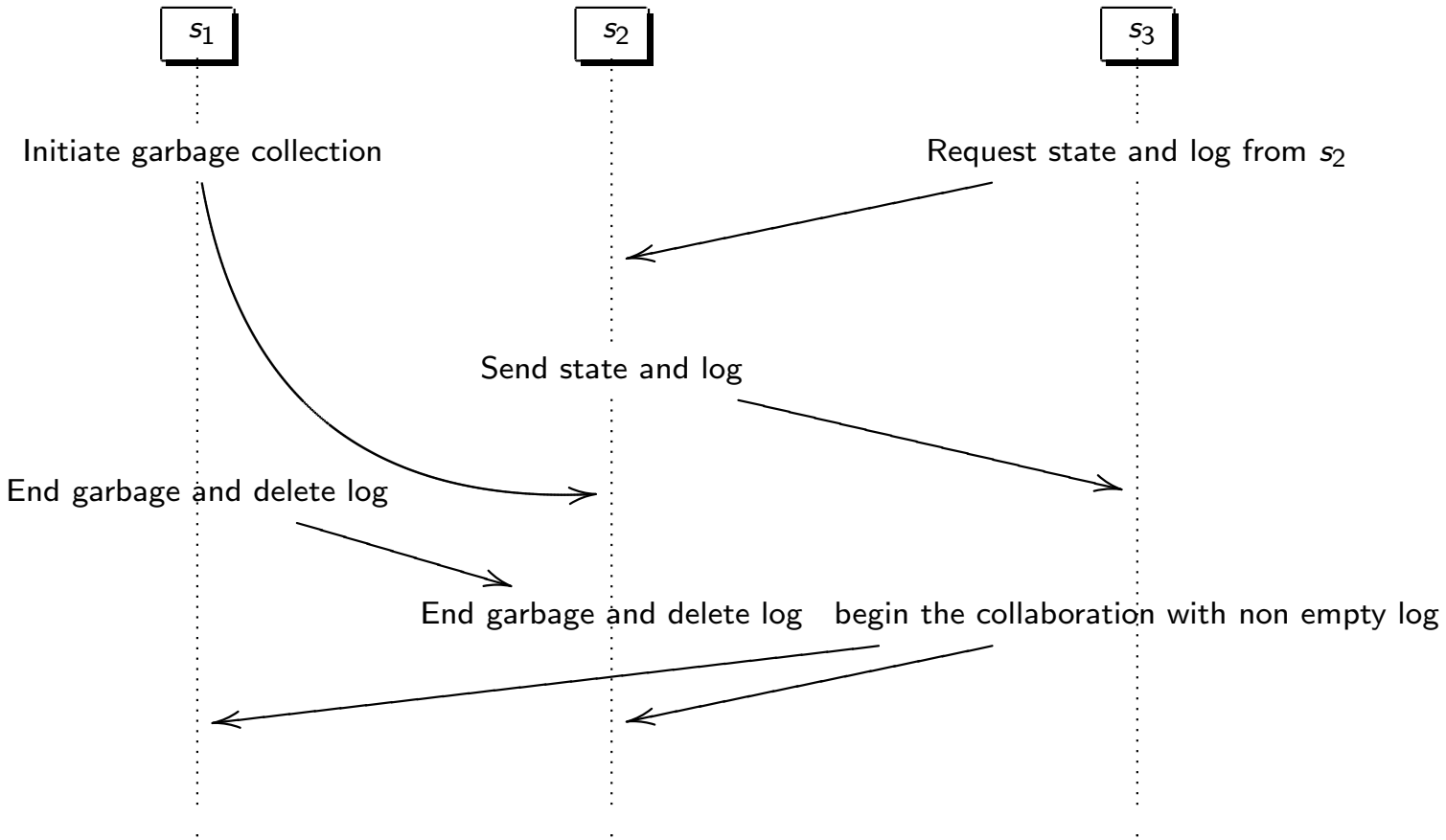
Issue

Execution order between garbage operations and cooperative operations

Solution

Wait until agreement about logs (global view)

Divergence caused by a new user joining the group



Issue

New user is not aware about garbage initiation

Solution

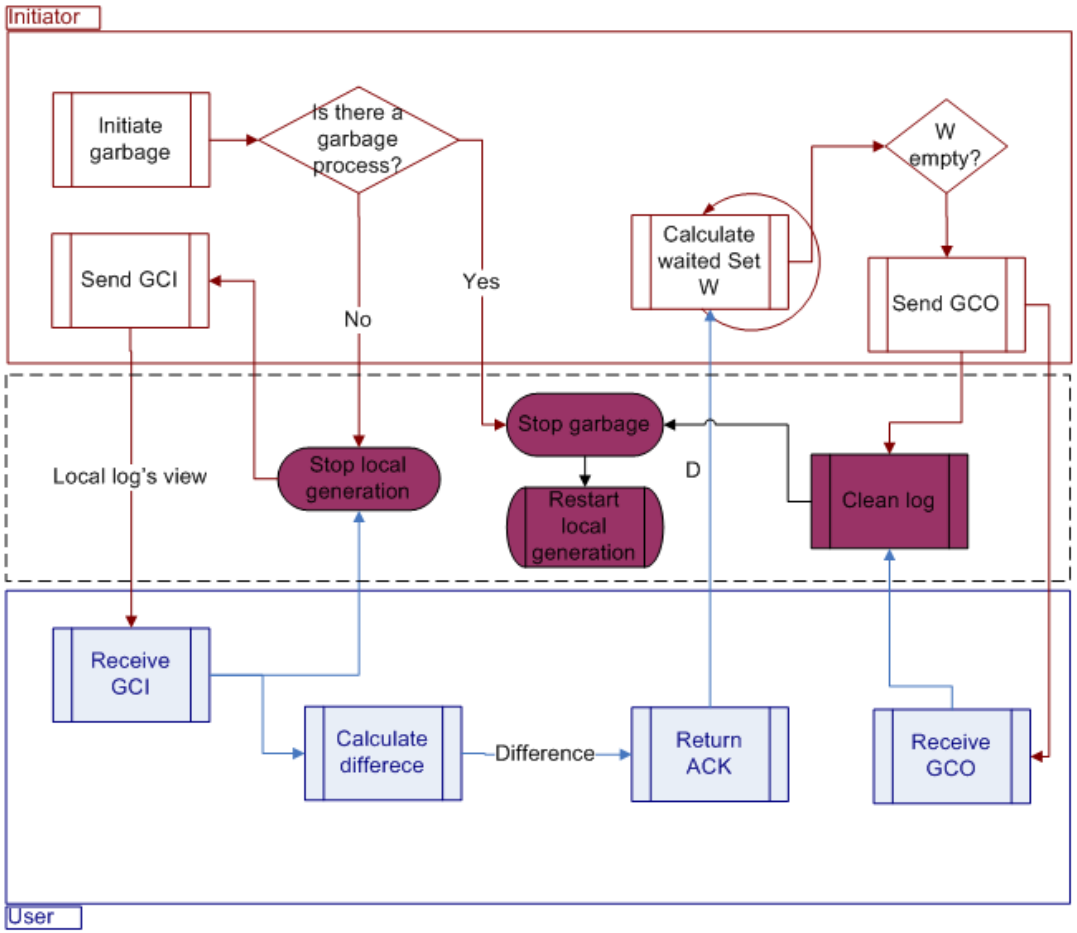
Enforce new or disconnected users to wait until garbage ends

Garbage messages

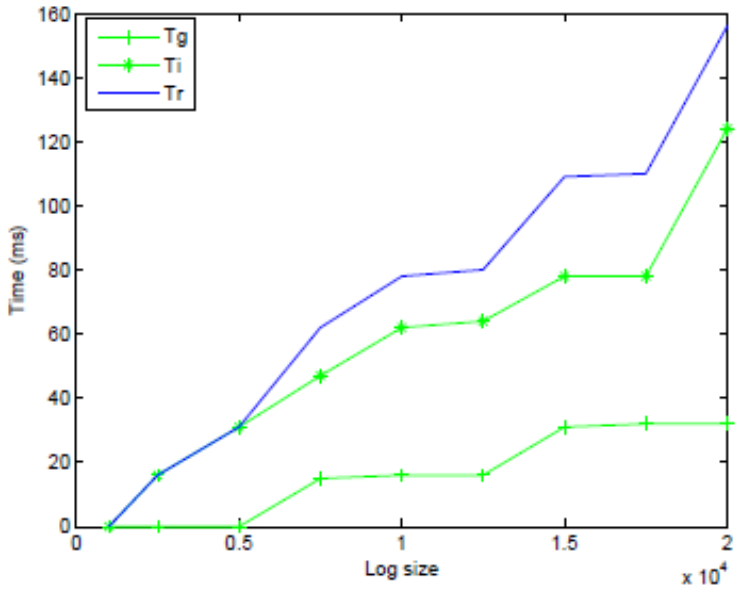
Exchange garbage collection messages

- Initiation (GCI): initiator identity + local view of log.
- Acquirement (ACK): difference between local and initiator views of log.
- Ordre (GCO): garbage order.

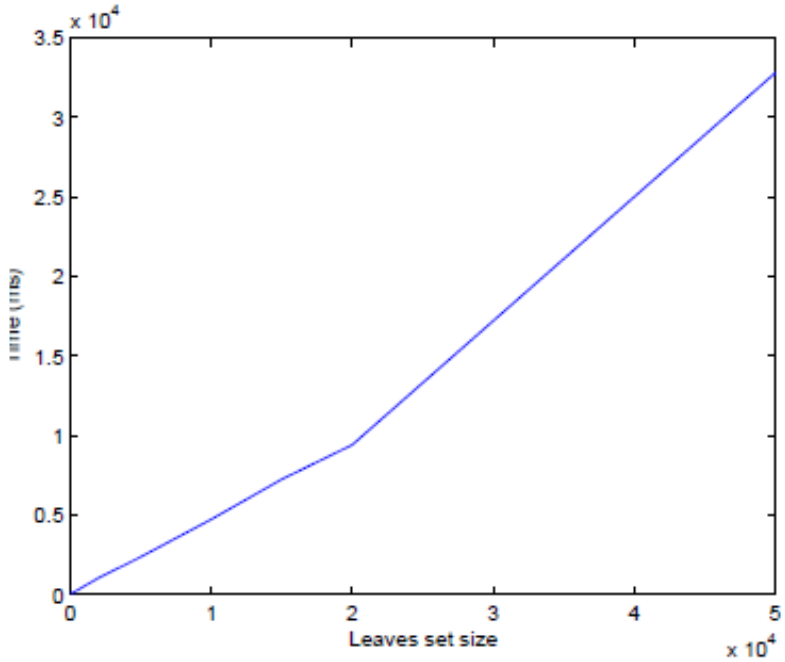
Garbage protocol



Implementing OPTIC with garbage collector on CDC mobile phones



Response Time for the CDC environment



Garbage collection time

Pros & cons

- (+) scalable
- (+) cleaning equivalent logs
- (+) good performance
- (-) blocking solution: local generation is stopped during garbage collection

Contribution

A first step towards garbage collection in collaborative editors to extend them for mobile devices.

Conclusion

- Contributions:
  - New framework for controlling access in collaborative editing work based on Optimistic Access Control.
  - Description and checking of our authorization policies are very simple (to maintain high responsiveness).
  - Good behavior with large logs.
  - Propose an approach to garbage logs.
  - Performance evaluation with a large scale distributed platform grid5000.
- Perspectives: Deploy the model for social network applications
  - Existing applications are centralized
  - Allow every user to manage access control for his objects