Semantic and Locality Aware Consistency for Mobile Cooperative Editing*

André Pessoa Negrão, João Costa, Paulo Ferreira, and Luís Veiga

INESC-ID/Instituto Superior Técnico
Rua Alves Redol 9, Lisboa, Portugal
{andre.pessoa,joao.da.costa}@ist.utl.pt,
{paulo.ferreira,luis.veiga}@inesc-id.pt

Abstract. This paper presents CoopSLA (Cooperative Semantic Locality Awareness), a consistency model for cooperative editing applications running in resource-constrained mobile devices. In CoopSLA, updates to different parts of the document have different priorities, depending on the relative interest of the user in the region where the update is performed; updates that are considered relevant to the user are propagated frequently, while less important ones are postponed. As a result, the system makes a more intelligent usage of the network resources, since 1) fewer accesses to the network are issued, 2) bandwidth savings are obtained by merging the delayed updates, and 3) reduced bandwidth available is used more efficiently by propagating more relevant updates sooner. These properties are of vital importance in the mobile environments we are addressing, in which devices have limited bandwidth and battery power. We have implemented a collaborative version of Tex editor TexMaker using the CoopSLA approach. We present evaluation results that support our claim that CoopSLA is very effective in reducing the overhead of replica synchronization without imposing limitations to application models.

Keywords: Cooperative Editing, Optimistic Replication, Data Consistency, Interest Management, Divergence Bounding.

1 Introduction

Cooperative editing applications enable geographically distributed users to concurrently edit a shared document space over a computer network[3]. Recently, these applications experienced an increase in popularity as a result of the expansion of the Internet and the rapid proliferation of mobile devices, such as smart phones, PDAs and tablets[10]. These modern devices are now sophisticated enough to allow its users to execute cooperative editing applications and participate in editing sessions alongside more powerful devices – like desktops or laptops – possibly mediated by cloud infrastructures.

* This work was partially supported by national funds through FCT – Fundação para a Ciência e Tecnologia, under projects PTDC/EIA-EIA/102250/2008, PTDC/EIA-EIA/108963/2008, PTDC/EIA-EIA/113993/2009 and PEst-OE/EEI/LA0021/2011.

R. Meersman et al. (Eds.): OTM 2012, Part I, LNCS 7565, pp. 380–397, 2012.

A critical technique to support these new heterogeneous environments – that mix resource constrained and powerful devices, interacting over wired and wireless networks – is to replicate the application data at the users' devices and resort to optimistic protocols to manage the consistency of the shared state. Optimistic replication[12] has the potential benefit of improving performance, availability and usability by allowing faster (local) access to the data. It also makes a more efficient use of the resources since it does not require constant access to the network for synchronization purposes. Optimistic mechanisms have been extensively applied to cooperative editing, in particular the Operational Transformation paradigm[2,15,14,7] and, more recently, Commutative Replicated Data Types[9,8,18,11,19].

While the state-of-the-art solutions provide a fair compromise between consistency and performance, they still neglect two important aspects that can be leveraged to improve the performance and overall usability and experience of cooperative editing applications. First, they do not consider the variable and highly dynamic characteristics of group work, in which different users are interested (and work on) different parts of the document space: i) a user is more interested in the zone(s) of the document that he is editing and a few other *observation points*, rather than the whole document space equally, and ii) his interest in the different sections of the document space varies over time. Second, optimistic systems based on eventual consistency are typically prone to some level of uncertainty and disruption: while the system is ensured to converge in the future, there is limited or no support to determine how current is the data observed by the user, and to establish and enforce clear bounds or guarantees on that currentness.

In this work we argue that it is possible to make a more efficient and scalable usage of the network resources by taking the users' interest into account. To address this issue, we propose CoopSLA (Cooperative Semantic Locality Awareness), a consistency model that unifies several well-know concepts of the distributed systems field: interest management, locality-awareness and bounded divergence. In CoopSLA updates are assigned a per-user priority level based on its *semantic distance* to the user's *observation point(s)* (Interest Management). Updates to regions closer to the observation points are considered more relevant and, thus, are awarded higher priority; priority decreases as the distance to the observation point increases (Locality-Awareness). In each priority level, updates are managed according to a parametrizable, multidimensional consistency space that determines when they must be propagated, establishing clear and well-known bounds to the divergence between the different replicas of the system (Bounded Divergence).

With CoopSLA, we are able to make a more intelligent and semantically meaningful usage of the network resources: by postponing updates with less priority, the system can merge and aggregate them, minimizing accesses to the network and reducing bandwidth; as a result of message aggregation, the latencies of the more important messages are reduced at the expense of the less relevant ones. These properties are of particular importance when mobile devices are in use,

because wireless networks provide low bandwidth with high latency, which has a significant impact on performance and interactivity[6], and frequent access to the network resources greatly increases battery consumption[4]. Another important aspect of CoopSLA is that it establishes bounds on the amount of replica deviation allowed, providing users with stronger guarantees regarding the actual consistency state of the document space.

We designed and implemented a middleware layer that enforces the CoopSLA model on behalf of the applications. This allows current single-user applications to be more easily adapted to support collaborative features and relieves programmers from the daunting task of designing and programming complex network and replication protocols. Using the CoopSLA middleware, we implemented a collaborative version of the popular Tex editor *TexMaker*. We present experimental results that support our claim that CoopSLA is very effective and flexible in reducing the overhead of replica synchronization without imposing limitations to application models and traditional semantics.

The paper is organized as follows. Section 2 introduces relevant concepts and describes the main assumptions of our work. Section 3 describes the CoopSLA consistency model in detail. Section 4 presents the main architectural aspects of the CoopSLA middleware. Section 5 overviews the implementation of our solution. Section 6 presents and discusses the experimental results. Related work is presented in Section 7. Finally, Section 8 concludes the paper.

2 System Model

In a cooperative editing session, multiple geographically distributed users concurrently edit a shared *document space* – for example, a Latex project, a wiki or a Word document. Common to these scenarios is a hierarchical structure of *semantic regions*, which are logical sub-divisions of the document space (like a folder, a file, or a $\backslash$*section* of a Latex document). Also, semantic regions may have logical references to other regions (e.g., a Latex $\backslash$*ref* or a link on a webpage). When considering the interest of a user, both the structure of the document space and the references between its components must be taken into account.

We model the document space as a rooted directed graph $G = (V, E)$. Each vertex V corresponds to a *semantic region* of the document space and there is an edge (v_i,v_j) between vertices v_i and v_j *iff* there is a *relation* between the two. We consider two types of relations: a *structural* edge connects two vertices that have a parent/child relation that is part of the hierarchical organization of the document (in a Latex document, for example, a $\backslash$*chapter* has a structural relation with each of its child $\backslash$*section*); a *semantic* edge is a non-structural edge that connects two vertices that have an application-specific reference between them (e.g., a $\backslash$*ref* in a Latex document or a link in a web page). Structural edges define a subgraph S of G corresponding to the tree structure of the document space.

We define a function $d : V \times V \rightarrow \mathbb{N}_0$ over the graph that represents the *semantic distance* between two vertices of the graph. The semantic distance

indicates the degree of correlation between two semantic regions of a document according to some application-dependent and user-aware criteria. A lower value denotes high correlation, while a higher value denotes low correlation.

We denote the participants of a cooperative editing session as *cooperation group* and each participant is called *node*. Each node of the cooperation group has a local *view* consisting of a full *local replica* of the shared application state that may have bounded inconsistencies with relation to the latest state of the application. Each replica consists of the document space graph G in which each vertex also holds the contents of the corresponding semantic region. In this context, we refer to a vertex of the graph as an *object*. Each object has one primary/master replica that holds its most recent value and one or more secondary replicas that may have stale values. The consistency model makes no restrictions as to which nodes of the cooperation group can be the master of an object.

3 A Semantic and Locality Aware Consistency Model

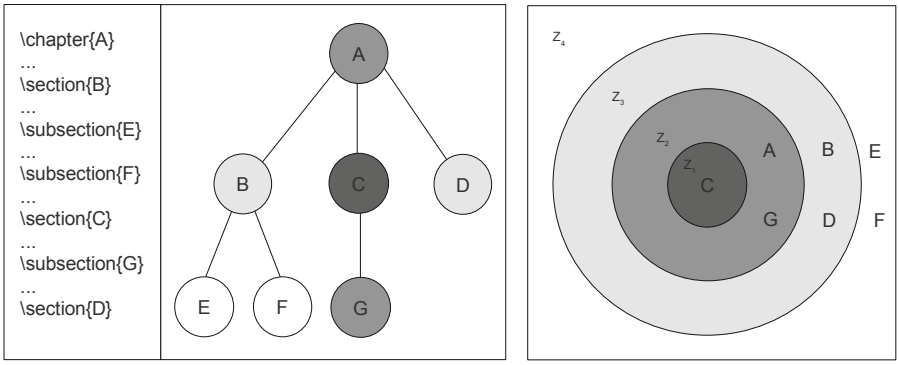
To capture the interest of a user in the different regions of a document, the CoopSLA model incorporates the notion of a *pivot*. A pivot is a special object that corresponds to a user's observation point and according to which the consistency requirements of the user's view is managed. Broadly speaking, the pivot determines, on a per-user basis, i) when an update to an object is allowed to be postponed, and ii) under which conditions previously postponed updates are required to be propagated to the user.

Different users have different pivots and each user may have multiple pivots, each corresponding to a semantic region in which he is interested. In this section, we describe the pivot-based CoopSLA consistency model in detail. For clarity, we first describe the main concepts of the CoopSLA model considering only one pivot (Sections 3.1 and 3.2). We briefly describe a generalization of the model for multiple pivots in Section 3.3.

3.1 Consistency Field

Each user's pivot p has a position in the application graph ($p \in V$). p can change throughout the execution of the application, mirroring the dynamic interest of the user in the document space. Moreover, a pivot generates a discrete consistency-field composed of n *consistency zones* $z_1, ..., z_n$ where: z_1 is the inner zone of radius r_1 and centered in p; each zone $z_i \in \{z_2, ..., z_{n-1}\}$ corresponds to the area with outer radius r_i and inner radius r_{i-1} (the radius of zone z_{i-1}); the outer zone z_n corresponds to the area beyond r_{n-1}, the radius of zone z_{n-1}. It follows that the consistency zone z_o of object o at semantic distance $d(p, o)$ from pivot p in the consistency-field is given by

$$z_o = \begin{cases} z_1 & \text{iff } d(p,o) \leq r_1 \\ z_i, 1 < i \leq n-1 & \text{iff } r_{i-1} < d(p,o) \leq r_i \\ z_n & \text{iff } r_{n-1} \leq d(p,o) \end{cases} \quad (1)$$



(a) Example document structure and corresponding graph representation with consistency requirements.

(b) Corresponding consistency zones.

Fig. 1. Example of Consistency Zones in a document structure

Figure 1 shows a simple example of a consistency field. The user's descending order of interest, based on which the consistency field is defined, is described in Table 1; the table explains how, taking into account current pivot position, each section of document is mapped to a given consistency zone.

In this example, the user is editing section C of a Latex document (left side of Figure 1(a)); as a result, the pivot is placed in the corresponding C vertex of the application graph (right side of Figure 1(a)). The consistency field generated (left side of Figure 1(b)) assigns the highest priority to updates to section C, the current editing section. Updates to sections A e G (respectively, the parent and child sections of C) have lower priority than C, but higher than sections B and D – the sections that are two graph hops away from C. Updates to section E and F have the lowest priority.

3.2 Consistency Requirements

Each consistency zone z_i has a corresponding *consistency degree* c_i that specifies the consistency requirements of the objects located within that zone. Consistency degrees respect the property $c_i > c_{i+1}$, meaning that consistency degree c_i of consistency zone z_i enforces stronger consistency than degree c_{i+1} of zone

Table 1. Example user interest

Priority	Description	Zone
1	Current editing section	z_1
2	Parent and child sections of current editing section	z_2
3	Close sections – sections two graph hops away.	z_3
4	Remaining sections	z_4

z_{i+1}. It follows from this property that consistency degrees (and, consequently, requirements) become weaker as their semantic distance to the pivot increases. Consistency degrees are described by 3-dimensional *consistency vectors* (κ) that limit the maximum divergence between the local replica of an object and the latest state of the object:

- *Time* (θ): Specifies how long (in seconds) an object is allowed to remain without being refreshed with its latest value.
- *Sequence* (σ): Defines the maximum number of updates to an object that are allowed to be postponed (missing updates).
- *Value* (ν): Specifies the maximum percentage divergence between the contents of the local replica of an object and its primary replica. *Value* is an application-dependent metric calculated by a special purpose function defined by the application's programmers.

CoopSLA guarantees that an object is updated whenever at least one of the previous criteria is about to be violated. Consider, for example, a consistency vector $\kappa = [0.25, 6, 20]$; an object within the consistency zone corresponding to κ is guaranteed to be, at most, 0.25 seconds outdated, 6 updates behind the primary replica or with contents diverging 20% from the object's latest value.

3.3 Model Generalization

CoopSLA allows the definition of multiple pivots for each user. For example, if a user is editing multiple files, each one of the editing points in the different files may correspond to a different pivot. Furthermore, different pivots may have different consistency requirements, as it is natural that the current editing point is more relevant to the user. In a multi-pivot setup, an object's consistency zone is assigned with relation to its closest *pivot*.

The model also allows the definition of multiple views per user, which allows different sets of objects to be characterized with different consistency requirements regarding the same *pivot*. Consider a pivot that corresponds to the paragraph currently being edited by the user. In this scenario, a user may be more interested in sections he created than in sections created by other users, regardless of how close they are to the user. With multiple views, user created objects may be assigned more strict consistency requirements.

4 Architecture

CoopSLA is implemented by a middleware layer that abstracts the programmers from the aspects related to network communication and consistency enforcement. The middleware follows a client-server architecture, in which clients edit the shared document space and the server propagates updates to each client according to its consistency specifications. In this section we describe the main architectural aspects of the CoopSLA middleware. We start by presenting an overview of the system, after which we describe how it represents the document space internally and how the consistency model is enforced.

4.1 Overview

Following the CoopSLA replication model (see Section 2), the server holds a full replica of the shared application state (i.e., the application graph G). The server stores the primary replica of every object in the system and, thus, always has the most recent version of the objects. When the server receives a client update it applies it to its primary replica immediately; in contrast, it postpones propagating the received updates to the clients, as long as their consistency requirements are respected.

A client consists of the editing application stacked on top of the middleware. It receives the input from the user, applies it to its local replica and submits the corresponding update to the server. Clients do not communicate directly with each other; update propagation is exclusively performed by the server. Each client holds a full replica of the data; however, unlike the server, these are secondary and, as a result, may have stale values that are managed according to each client's consistency specification.

The main task of the server is to enforce the CoopSLA consistency model. This requires it to continuously monitor client updates and collect information about the current consistency state of each client. Periodically, and when updates are received at the server, it executes a validation algorithm that uses the collected data to verify if the consistency requirements of the clients are still met; if not, the server propagates the postponed (possibly merged) updates to the clients that would, otherwise, violate their consistency specification.

4.2 Data Representation

The CoopSLA middleware represents the contents of each object of the graph as a TreeDoc Commutative Replicated Data Type (CRDT)[9]. By doing so, we enable replicas to converge without the need for complex conflict resolution protocols, which further enhances the relaxed synchronization properties provided by CoopSLA. As a result of using TreeDoc, our system follows an *operation transfer* design. This means that the update messages exchanged during an editing session consist of *add* or *remove* operations, instead of the actual data.

Updates that have not yet been propagated to a client are stored at the server in a per-client *update queue*. When adding a new update to a queue, the server automatically merges add/remove operations that cancel each other. Even by just using this mechanism, the results (Section 6) proved to be very encouraging. Alternative (or complementary) merging solutions are still being implemented and are out of the scope of this paper.

4.3 Monitoring Client Activity

To enforce the consistency model the server stores, for each client c_i, the client's consistency specification (pivots, zones and degrees) and *consistency state* table ψ_{c_i}. The latter stores, for each object o_i: 1) the time elapsed since c_i last received updates regarding o_i ($\psi_{c_i}[\theta, o_i]$), 2) the number of updates to o_i that have not

yet been sent to c_i ($\psi_{c_i}[\sigma, o_i]$), and 3) the *value* of o_i the last time updates to it were sent to c_i ($\psi_{c_i}[\nu, o_i]$).

Enforcing each client's consistency specifications requires the server to keep track of the following critical events:

Content Updates. When the server receives a content update (and *add* or *remove* request) it adds it to the update queues of the clients and updates the *sequence* state $\psi_{c_i}[\sigma]$ of every client c_i. Next, it verifies if the new value of the *sequence* metric of c_i has reached the bound specified in c_i's consistency specification; if so, it marks o_i as *dirty* in c_i's *dirty table*. When processing the next round of the validation algorithm, the server verifies that o_i is dirty and, as a result, refreshes the client's state by propagating the updates needed to ensure the client's consistency specifications are met.

Structure Updates. Modifications to the structure of a document change the distances between its regions. As a result, the placement of the objects within the consistency fields of the clients change and new consistency requirements have to be considered; thus the server is required to update its internal data structures accordingly. Furthermore, because a structure update is also an update to the document, the server also updates and re-evaluates the *sequence* state $\psi_{c_i}[\sigma]$ of each client for every object that moved to a different consistency zone.

Pivot Movement. As with structure updates, when a pivot moves the composition of its consistency zones change, resulting in new consistency requirements. As a result, the server has to update the internal data structures representing that client accordingly, as well as re-evaluating, for every object that moved to a different consistency zone, the *sequence* state of the client. Note that in this case $\psi_{c_i}[\sigma]$ is not updated, since the movement of the pivot does not modify the document space.

4.4 Consistency Enforcement

In each periodically executed round of the consistency validation algorithm, the server checks if any update received since the last round resulted in a violation of a client's consistency specification. If so, the identified updates are propagated to the client.

The validation algorithm verifies, for each client c_i and each object o_i, if the object is within the limits imposed by the consistency zone defined by the client's pivot(s). Because c_i may have multiple views and multiple pivots, the server must first identify which pivot p_i enforces the strongest consistency requirements for o_i. Then, it identifies the consistency zone of p_i where o_i lies, retrieving the corresponding consistency vector κ_i.

Next, each dimension of the identified κ_i is tested. Verifying *time* (θ) and *sequence* (σ) is straightforward: for σ, the server simply checks if the object has

been previously marked as dirty (Section 4.3); for θ, it tests if the time elapsed since the last time c_i was updated with the latest version of o_i exceeds θ_{κ_i}.

To verify ν, on the other hand, the server has to compare the current value of o_i with the client's $\psi_{c_i}[\nu, o_i]$, which would require it to store, for every client, a copy of every object. To avoid the memory overhead of such a solution, the server takes a snapshot of an object whenever updates regarding that object are propagated to a client. Before taking a snapshot, however, it first verifies if a snapshot of the object already exists; if it does, the server uses it, avoiding an unnecessary copy of the object and saving memory.

5 Implementation

We implemented a prototype of the CoopSLA middleware and extended the Linux version of the Latex editor *Texmaker*[1] on top of it. In this section we describe the main implementation details of the middleware (Section 5.1) and the extension to Texmaker (Section 5.3) and explain how programmers interact with the middleware and specify the CoopSLA consistency settings (Section 5.2).

5.1 Middleware

Each semantic region of the application graph is represented by an SRegion object that contains a list of children subregions and a TreeDoc with the contents of the region it represents. SRegions are uniquely identified by the server; this identifier is used by clients to access the semantic region represented by the object. The list is ordered by the semantic order of the children subregions in the document space. SRegions also hold a programmer provided DataUnit object containing application-specific information. It may be used, for example, to implement links and references between SRegions.

Implementing the object snapshot approach described in Section 4.4 requires rounds, snapshots and objects to be *versioned*. The round number r_v is an integer number that is incremented in every round. Snapshot and object versions are assigned based on round versions: snapshot versions correspond to the r_v of the round in which the snapshot was taken; object versions correspond to the r_v of the round in which the object was last updated. To dispose of snapshots that are no longer referenced we hold a list of the clients that reference the snapshot and collect the latter when the list becomes empty.

To save memory, the per-client pending updates queues do not store actual updates. Instead, it points to the updates stored in the global queue. Thus, the global queue holds the updates received since the last round, as well as any update that has not yet been sent to a client (i.e., is still referenced by a client's queue). When an update is no longer referenced by any client's queue, it is discarded.

[1] `http://www.xm1math.net/texmaker/`

5.2 Interfacing with Programmers

In our current implementation, programmers describe the consistency requirements using an XML file. When the client application starts, it invokes an API registration function with the path to the XML file as its argument. The CoopSLA client then parses the file and sends a registration request to the server.

Programmers control the structure of the document by adding or removing semantic regions using API functions (addSRegion/removeSRegion). Both functions receive the parent of the new region, the region *type* and, optionally, a set of semantic links to other regions. The region type is an application-dependent string value used to identify the objects consistency zone for a particular pivot (explained later in this section).

Programmers must provide two additional functions to be called by the middleware when checking a client's consistency: **valueDiff** and **getConsistencyZone**. The **valueDiff** function is used to verify the *value* metric. It returns the (application-specific) percentage difference between two versions of an object. **getConsistencyZone** is a functions that, given a pivot, a graph object and the graph path between the pivot and the object, returns the consistency zone of the object regarding the pivot.

5.3 CoLaTex

To validate our system, we have extended the Tex editor *Texmaker* with cooperative capabilities using its add-ons feature. Our add-ons consist of simple functions that intercept the user's modifications to the local replica of the shared document, insert the updates received from the server into the Latex document and track the user's editing position.

We defined one pivot for each open file, each corresponding to the semantic region being edited by the user within that file. We implemented an add-on that allows the user to manually assign a region of the document to a consistency zone. Our **valueDiff** function returns the percentage difference in number of characters between two versions of an object. Table 2 describes the consistency zones we used defined; the **getConsistencyZone** function returns the consistency zone of an object based on the following considerations:

– *Consistency Zone 0* includes the region in which the pivot is placed and its direct children, i.e., the ones that are one graph-hop below the pivot.

Table 2. Consistency Zones

Zone	*Time* (θ)	*Sequence* (σ)	*Value* (ν)
1	1 sec.	1 update	1%
2	10 sec.	15 updates	5%
3	40 sec.	100 updates	30%
4	2 min.	750 updates	60%
5	5 min.	1000 updates	90%

- *Consistency Zone 1* contains the direct parent – the region one graph-hop above the pivot – and indirect children – identified by traversing the graph downwards from the pivot, excluding the direct children – of the pivot.
- *Consistency Zone 2* comprises the regions that belong to the same *chapter* as the pivot. If there is no explicit \chapter defined, we consider that the document has one implicit chapter of which every section is a part of.
- *Consistency Zone 3* includes the top-level sections of the remaining chapters (*chapter*) of the document. If the document does not have chapters, zones two and three are merged and zone four is awarded the consistency specifications originally defined for zone three.
- *Consistency Zone 4* contains the regions that do not belong to any of the previous zones.

6 Evaluation

We conducted a series of tests to experimentally validate our claim that CoopSLA makes a more efficient use of network resources by exploiting the *locality of interest* of users. In particular, we intended to quantify the savings that CoopSLA obtains regarding the overall bandwidth required to propagate the updates generated during an editing session and the access frequency to the network resources. In the following sections we detail the experiments conducted. We first describe the configuration of the simulation environment, and then present and analyse the results obtained.

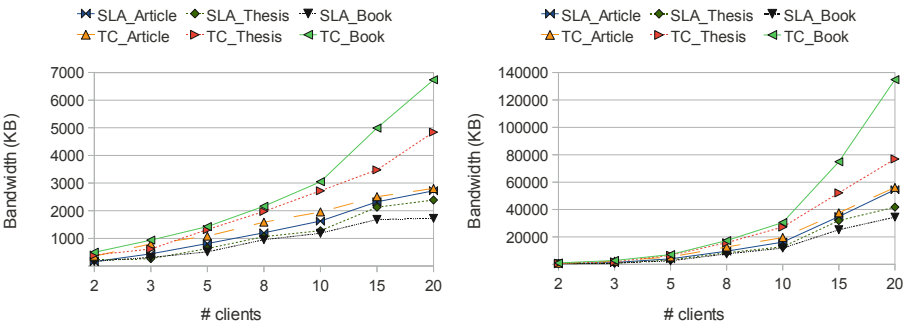
6.1 Simulation Environment

Clients are simulated by running a predetermined number of parametrized *editing bots*. Editing bots perform text insertions (write or paste), deletions (erase or cut) and browse through the document space. Table 3 shows the decision tree that models the behaviour of the bots used in our experiments.

Each simulation consisted in a five minute run during which bots executed according to their decision tree, propagating the corresponding updates to the server. The server monitored inbound and outbound messages, storing per-client and overall values regarding bandwidth and number of exchanged messages. Unless told otherwise, the results presented were obtained using the consistency requirements described in Table 2. For simplicity, we chose to have only on pivot per-client in the experiments.

Table 3. Bot decision tree

	Add		**Remove**		**Move**	
Read	**Write**	**Paste**	**Erase**	**Cut**	**To sides**	**Up/down**
60%	15%	3%	8%	3%	8%	3%



(a) Average per-client inbound bandwidth. (b) Total server outbound bandwidth.

Fig. 2. Used bandwidth

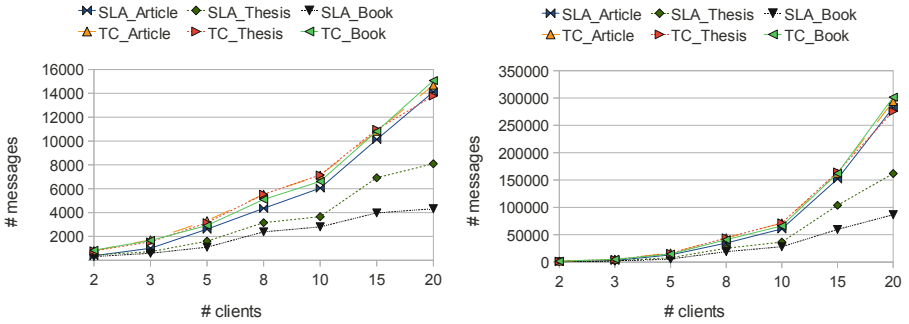
We compared CoopSLA with a baseline TreeDoc implementation that propagates updates to every user as soon as they arrive at the server. Throughout this section we refer to this system as *Total Consistency* (TC), due to its eager propagation approach.

The tests were conducted on Intel Core 2 Quad machines with 8GB Ram running Ubuntu Linux. The server executed on a dedicated machine with no other user-level application running; clients were deployed on up to three machines. The computers were connected through a LAN.

6.2 Evaluation Results

In the first set of experiments, we measured bandwidth and number of accesses to the network at both the clients and the server. In these simulations we varied the number of clients and the type of document space edited. We considered three types of documents that differ mainly in size and structural complexity: an article, a PhD thesis and a book.

Figure 2 presents the results obtained regarding bandwidth. We plotted the results of both CoopSLA and TC for an increasing number of users and the three types of documents we consider (article, thesis and book). The figures show that CoopSLA is able to effectively reduce bandwidth usage both at the client and the server side. Moreover, it shows that as the number of clients and the size of the documents increase, CoopSLA is increasingly more efficient in obtaining bandwidth savings. This behaviour shows the scalability of the system, and is especially relevant considering that as an editing project grows in size, it is more likely that more users will cooperatively access it. The main reason for these results is that in larger documents the average distance between the editing regions of the users is higher; as a result, the probability of postponing and, eventually, merging updates also increases. This fact is particularly evident if we make a pairwise comparison between CoopSLA and TC for each document type. With the article (CoopSLA_Article and TC_Article), the bandwidth savings obtained by CoopSLA are minimal, because the probability that an update



(a) Average number of messages received per-client.

(b) Total number of messages sent by the server.

Fig. 3. Messages exchanged

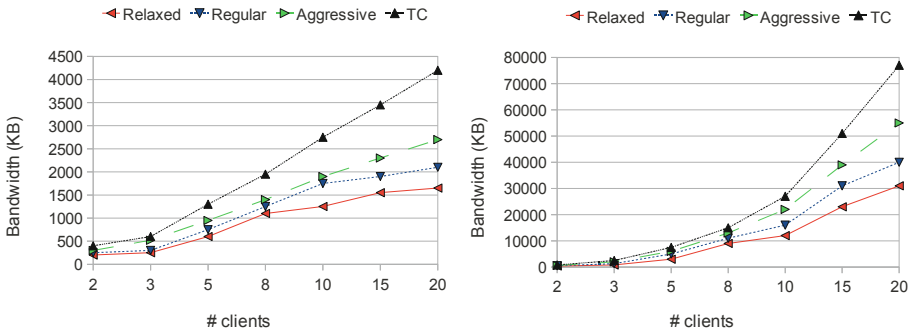
occurs in a client's pivot region (and, consequently, is propagated immediately) is high[2]. If the document grows in size and complexity, the bandwidth savings obtained by CoopSLA increase greatly: approximately 45% less bandwidth with the thesis document (CoopSLA_Thesis and TC_Thesis) and 65% with the book (CoopSLA_Book and TC_Book).

Another important conclusion that can be inferred from Figure 2 is that CoopSLA is able to efficiently minimize one of the main drawbacks of the CRDT approach, the size overhead of path identifiers. When a document is represented as a CRDT, the size of the TreeDoc path identifiers increases as the document grows. Because update messages exchange path identifiers, when a document grows in size, the update messages follow the same pattern. Without CoopSLA, the larger the document is, the larger update messages are; as a result, the bandwidth required to update clients increases. With CoopSLA, on the other hand, we take advantage of the accumulation of postponed messages at the server to merge them before propagating them to the clients.

While the overall traffic generated by the clients is influenced by the specific characteristics of TreeDoc, the number of update messages issued by each client depends only on the editing pattern of the users. By measuring the number of messages exchanged over the network, we are able to clearly isolate and analyse the individual contribution of CoopSLA. Figure 3 shows the results of these measurements for both CoopSLA and TC.

The results further confirm the ones regarding bandwidth. Figure 3 shows that CoopSLA is able to reduce the number of messages received by each client and those savings increase with the size and complexity of the edited document. Again, these results are a direct consequence of CoopSLA's ability to leverage the accumulation of low-priority postponed messages by merging those that cancel each

[2] Note, however, that we could have obtained better results by configuring the consistency requirements of the pivot region to a less strict setting. We analyse the influence of varying the parameters of CoopSLA later in this section.



(a) Average per-client inbound bandwidth. (b) Total server outbound bandwidth.

Fig. 4. Bandwidth usage with different consistency requirements

other. As a result, CoopSLA discards unnecessary messages that would have, otherwise, been propagated immediately to each client. This behaviour is desirable not only because it has a direct influence on the reduction of the bandwidth requirements, but also because it means that mobile devices using CoopSLA make a less demanding usage of the network resources, which contributes to reduce battery consumption. Furthermore, these results provide an encouraging indication of how CoopSLA would work with different *operation-transfer* strategies, like OT or other CRDT implementations.

The results presented so far misleadingly indicate that when a document is small, there is no advantage in using CoopSLA. However, CoopSLA allows application programmers to specify consistency requirements arbitrarily, as they see fit for their applications. As long as possible, a programmer should try to relax the consistency requirements, always ensuring the application provides the required levels of interactivity. If necessary, however, the programmer can define more demanding requirements. To show the flexibility of CoopSLA, we measured the bandwidth usage of three CoopSLA consistency specifications that differ in update propagation aggressiveness, as described in Table 4. The *Relaxed* specification provides the weaker guarantees; in particular, it does not not require high-priority updates to be propagated immediately. *Aggressive* is the most demanding specification; it requires high-priority updates to be immediately propagated and

Table 4. Consistency Zones

Zone	Relaxed	Regular	Aggressive
1	$\{\theta$=5,σ=10,ν=5$\}$	$\{\theta$=2,σ=5,ν=5$\}$	$\{\theta$=1,σ=1, ν=1$\}$
2	$\{\theta$=20,σ=30,ν=10$\}$	$\{\theta$=10,σ=10,ν=5$\}$	$\{\theta$=5,σ=5,ν=5$\}$
3	$\{\theta$=40,σ=100,ν=50$\}$	$\{\theta$=40,σ=100,ν=30$\}$	$\{\theta$=15,σ=15,ν=20$\}$
4	$\{\theta$=120,σ=750,ν=60$\}$	$\{\theta$=90,σ=300,ν=60$\}$	$\{\theta$=30,σ=50 ν=50$\}$
5	$\{\theta$=300,σ=1500,ν=90$\}$	$\{\theta$=180,σ=750,ν=80$\}$	$\{\theta$=60,σ=150,ν=50$\}$

the remaining zones to be updated frequently. *Regular* is an intermediate specification that provides more relaxed consistency than *Aggressive*, but stricter than *Relaxed*.

Figure 4 shows the results obtained; the bar labelled TC corresponds to the version of the baseline TreeDoc implementation that does not use CoopSLA, while the remaining bars correspond to the three specifications of Table 4. The measurements were made using the Thesis document. As expected, the figure shows that CoopSLA is more efficient when the consistency requirements are more relaxed. This happens because relaxed consistency requirements allow for a larger volume of updates to be retained at the server for longer periods; as a result, the probability that two updates can be merged increases. Conversely, when we increase the aggressiveness of the requirements, we reduce the volume of updates that are retained at the server, thus reducing merge efficiency. However, even considering that CoopSLA is less effective with stronger consistency requirements, the results show that even a fairly strict set of requirements is able to obtain more than 20% bandwidth savings over the version that does not use CoopSLA.

7 Related Work

In this section we discuss prior work on the two topics that are more closely related with our work: divergence bounding and consistency in cooperative editing.

7.1 Divergence Bounding in Optimistic Replication

Designers of replicated systems typically choose between pessimistic and optimistic consistency models[12]. In many cases, however, neither the performance overheads imposed by strong consistency neither the lack of limits for inconsistency are acceptable to applications. An interesting alternative called *divergence bounding* consists in allowing updates to be managed optimistically, but define under which conditions replicas are required to converge and how to enforce that convergence. *Real time guarantees*[1], for example, allows replicas to remain stale for a specified maximum time, before they are required to synchronize. *Order bounding*, another simple solution, limits the number of updates that can be applied to a local replica without synchronization[5].

The TACT[21] framework proposes a multi-dimensional approach to divergence bounding that unifies in a single model three metrics: real-time guarantees, order bounding and a novel metric called *Numerical Error* that bounds the total number of updates, across all replicas, that can proceed before replicas are forced to synchronize. Our work distinguishes from TACT by embodying the notion of locality-awareness into the consistency model. This allows our system to implicitly assign different priorities to different updates that may vary throughout execution.

Vector Field Consistency (*VFC*) [13,17] is a consistency model for mobile multiplayer games that enables replicas to define their consistency requirements

in a continuous consistency spectrum. The novelty of the VFC model is that it combines multi-dimensional divergence bounding with *locality-awareness* to improve the availability and user experience while effectively reducing bandwidth usage. Consistency between replicas strengthens as the distance between objects decreases. To define these mutable divergence bounds, around pivots there are several concentric ring-shaped *consistency zones* with increasing distance (radius) and decreasing consistency requirements (increasing divergence bounds). Then, in each zone, like in TACT, programmers define a 3-dimensional vector: *time*, *sequence*, *value*.

7.2 Consistency in Cooperative Editing

The issue of maintaining replica consistency in cooperative applications has been extensively studied in the last two decades. The most representative solutions fall into the *Operational Transformation* (OT)[2,15,14,7,16,20] category. In OT, each locally generated operation is associated with a timestamp and broadcast to the remaining sites. Then, each remote update received is transformed (e.g., by adjusting its insert/delete index) in order to commute with concurrent operations already applied to the shared document. As a result, transformed operations can be executed without re-ordering previous applied operations.

OT transforms updates in order to make them commute. A recently introduced alternative is to make every operation automatically commutative by representing the document as a *Commutative Replicated Data Type* (CRDT)[9,8,18,11,19]. The CRDT approach considers that a document is composed of a sequence of immutable and uniquely identified elements that can be any non-editable component of a document, like a character or a graphics file. Commutativity is achieved by designing an identifier space that ensures that it is always possible to create a new identifier between two existing ones[9].

To the best of our knowledge, neither approach (OT or CRDTs) considers the dynamically changing interest of the users in the different semantic regions of a document; instead, they propagate every update with the same static priority. Moreover, CoopSLA can use either OT or CRDT as building blocks for update propagation. As described in Section 4.2, in our current implementation we used CRDTs.

8 Conclusion

In this paper we presented a semantic and locality aware consistency model for cooperative editing applications. Our model, named CoopSLA, explores the heterogeneous and dynamic interest of users in different regions of a document space in order to reduce communications between the participants of an editing session. CoopSLA assigns, on a per-user basis, different priorities to different updates, based on the semantic distance between the place in the document where the update is performed and the places in which the user is more interested. Updates with high priority are sent frequently to the client, while low priority

updates are postponed and, when possible, merged. Each priority level is characterized by a multidimensional consistency degree that defines how many and how long updates to a particular object are allowed to be postponed.

We implemented a middleware layer enforcing CoopSLA and extended the popular Tex editor *TexMaker* with cooperative features using it. We conducted a series of tests to experimentally evaluate the performance of CoopSLA. The results presented in this paper support our claim that CoopSLA is very effective in reducing the overhead of replica synchronization without constraining application models and respecting their consistency need.

References

1. Alonso, R., Barbara, D., Garcia-Molina, H.: Data caching issues in an information retrieval system. ACM Trans. Database Syst. 15(3), 359–384 (1990), http://doi.acm.org/10.1145/88636.87848
2. Ellis, C.A., Gibbs, S.J.: Concurrency control in groupware systems. In: SIGMOD 1989: Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data, pp. 399–407. ACM, New York (1989)
3. Ellis, C.A., Gibbs, S.J., Rein, G.: Groupware: some issues and experiences. Commun. ACM 34, 39–58 (1991), http://doi.acm.org/10.1145/99977.99987
4. Imielinski, T., Badrinath, B.R.: Mobile wireless computing: challenges in data management. Commun. ACM 37(10), 18–28 (1994)
5. Krishnakumar, N., Bernstein, A.J.: Bounded ignorance: a technique for increasing concurrency in a replicated system. ACM Trans. Database Syst. 19(4), 586–625 (1994)
6. Li, D., Anand, M.: Majab: improving resource management for web-based applications on mobile devices. In: Proceedings of the 7th International Conference on Mobile Systems, Applications, and Services, MobiSys 2009, pp. 95–108. ACM, New York (2009), http://doi.acm.org/10.1145/1555816.1555827
7. Li, R., Li, D.: A new operational transformation framework for real-time group editors. IEEE Transactions on Parallel and Distributed Systems 18(3), 307–319 (2007)
8. Oster, G., Urso, P., Molli, P., Imine, A.: Data consistency for p2p collaborative editing. In: CSCW 2006: Proceedings of the 2006 20th Anniversary Conference on Computer Supported Cooperative Work, pp. 259–268. ACM, New York (2006)
9. Preguiça, N., Marques, J.M., Shapiro, M., Letia, M.: A commutative replicated data type for cooperative editing. In: ICDCS 2009: Proceedings of the 2009 29th IEEE International Conference on Distributed Computing Systems, pp. 395–403. IEEE Computer Society, Washington, DC (2009)
10. Preguiça, N., Martins, J.L., Domingos, H., Duarte, S.: Data management support for asynchronous groupware. In: Proceedings of the 2000 ACM Conference on Computer Supported Cooperative Work, CSCW 2000, pp. 69–78. ACM, New York (2000), http://doi.acm.org/10.1145/358916.358972
11. Roh, H.G., Jeon, M., Kim, J.S., Lee, J.: Replicated abstract data types: Building blocks for collaborative applications. J. Parallel Distrib. Comput. 71(3), 354–368 (2011), http://dx.doi.org/10.1016/j.jpdc.2010.12.006
12. Saito, Y., Shapiro, M.: Optimistic replication. ACM Comput. Surv. 37(1), 42–81 (2005)

13. Santos, N., Veiga, L., Brandt, F.: Vector-Field Consistency for Ad-Hoc Gaming. In: Cerqueira, R., Campbell, R.H. (eds.) Middleware 2007. LNCS, vol. 4834, pp. 80–100. Springer, Heidelberg (2007)
14. Shao, B., Li, D., Gu, N.: A fast operational transformation algorithm for mobile and asynchronous collaboration. IEEE Transactions on Parallel and Distributed Systems 21(12), 1707–1720 (2010)
15. Sun, C., Ellis, C.: Operational transformation in real-time group editors: issues, algorithms, and achievements. In: CSCW 1998: Proceedings of the 1998 ACM Conference on Computer Supported Cooperative Work, pp. 59–68. ACM, New York (1998)
16. Sun, D., Sun, C.: Context-based operational transformation in distributed collaborative editing systems. IEEE Trans. Parallel Distrib. Syst. 20(10), 1454–1470 (2009), http://dx.doi.org/10.1109/TPDS.2008.240
17. Veiga, L., Negrão, A., Santos, N., Ferreira, P.: Unifying divergence bounding and locality awareness in replicated systems with vector-field consistency. J. Internet Services and Applications 1(2), 95–115 (2010)
18. Weiss, S., Urso, P., Molli, P.: Logoot: A scalable optimistic replication algorithm for collaborative editing on p2p networks. In: ICDCS 2009: Proceedings of the 2009 29th IEEE International Conference on Distributed Computing Systems, pp. 404–412. IEEE Computer Society, Washington, DC (2009)
19. Wu, Q., Pu, C., Ferreira, J.: A partial persistent data structure to support consistency in real-time collaborative editing. In: 2010 IEEE 26th International Conference on Data Engineering (ICDE), pp. 1707–1720 (March 2010)
20. Xia, S., Sun, D., Sun, C., Chen, D., Shen, H.: Leveraging single-user applications for multi-user collaboration: the coword approach. In: CSCW 2004: Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work, pp. 162–171. ACM, New York (2004)
21. Yu, H., Vahdat, A.: Design and evaluation of a conit-based continuous consistency model for replicated services. ACM Trans. Comput. Syst. 20(3), 239–282 (2002), http://doi.acm.org/10.1145/566340.566342