Vector-Field Consistency for Cooperative[1] Work

João Filipe Ferreira da Costa

INESC-ID/Instituto Superior Técnico
Distributed Systems Group
joão.da.costa@ist.utl.pt

Abstract. Although cooperative work is widely used in a daily basis, nonetheless cooperative work tools are not being used as their potential would suggest. Even though the reasons behind this fact are not completely clear, the truth is that users see current cooperative work tools as a burden to their working activities. To work around this problem, the notion of locality-awareness can be extremely helpful. Additionally, if we adapt consistency models to the cooperative activities being performed by each individual user, then we will be able to improve their experience without compromising the overall application performance. For example, we can decide which updates each user is most interested in and which ones he should be unaware of. The VFC algorithm does just that but in the environment of ad-hoc gaming. We propose VFC for Cooperative Work (VFC4CW), an adaptation of the original VFC consistency model to the environment of cooperative work, namely document-based cooperative work. In this work, we first study the state of the art of consistency maintenance in cooperative work tools. Then we propose an architecture which combines a generic VFC4CW middleware layer with a cooperative work application. At last, we discuss how to evaluate a future implementation of the proposed architecture.

Keywords: VFC4CW, continuous consistency model, locality-awareness, cooperative work, document-based.

1 Introduction

In everyday life, it is more and more common to perform *cooperative work* to carry out some task. For example, the cooperative production of documents (e.g. articles, presentations, financial reports) is a daily chore in enterprises [Noël04]. Also, wikis, the massive cooperative editor by excellence, are between the most used platforms on the Internet.

If writing is always a long and complex process, *cooperative writing* is an even more complex and difficult process. Cooperatively writing a text can indeed shorten the duration of the writing task if teams work well and so members do not hinder each others' work. If on the contrary teams do not function well by themselves, the additional effort needed to make them work may not pay off.

On contrary to what may be general belief, *cooperative work tools* are not as used as their potential suggests [Noël04]. Causing this, may be the general impression that these tools represent more of a burden than a relief. Although this is true in some cases, the major

[1] Although some authors suggest that there is in fact a difference in this field between the terms *cooperation* and *collaboration*, in this work there will be no distinction.

Fig. 1: Typical example of cooperative edition of a document.

obstacle is still the reluctance of experienced users in changing their everyday tools and methods of collaboration.

A suggested approach for cooperative tools to achieve the desired success is the idea of enhancing the existing work tools to supply them with functionalities addressing cooperation. Even though this may seem a promising technique, it is also a dangerous one. First, because modern mature work tools have so many functionalities, adding extra ones can compromise the tools' usability. Also, due to the great complexity of the existing functionalities, it is not always obvious how to provide them with cooperation faculties.

Although investigation has not yet come to a conclusion about the best method for designing a successful cooperative tool there are already some good examples. Some popular examples of success are the ones of *Google Docs* and *Google Spreadsheets*[2]. Forgetting about the fact that they benefit from the popularity of their developer company, the key to success may lay in their simplicity. With not so many functions but with very familiar ones, these tools manage to create a good developing environment. Another example is of course *Wikipedia*, the most popular wiki tool in the world, where users collectively produce text at a rate of 10 words per second[3]. Again, the simplicity and easy access of this tool are without a doubt reasons for its triumph. Finally, the *Google Wave*[4] example shows a completely different idea of what a cooperative tool should enable. Additionally to the cooperative work *per se*, Google Wave provides easy communication channels between members.

Although cooperative tools are little by little becoming more accepted and users are starting to see their potential, the *replication schemes* behind popular tools are still very conservative. Following the example of Wikipedia, it is implemented above a roughly centralized infrastructure and with no complex conflict resolver mechanism. Supporting such infrastructures can have huge costs when scaling these tools to a great number of users [Oster06, Weiss07, Morris07, Oster09].

This article proposes the application of a consistency model more coherent with the cooperative tools paradigm. First, it proposes the appliance of a *continuous consistency model* to better meet the requirements of these tools. Also the notion of *locality-awareness* will be combined with the continuous consistency model to adapt the consistency requirements to the current edition location of each user, within the document, and other points of registered

[2] http://docs.google.com/
[3] http://en.wikipedia.org/wiki/Wikipedia:Statistics
[4] http://wave.google.com/

interest. To implement these concepts, the *Vector-Field Consistency* (VFC) algorithm will be used and adapted to the desired context.

Since VFC was designed to the gaming environment, it is necessary to port it to the world of cooperative work, namely translate the concepts of location and distance between users' locations, which are pretty straightforward in its original form. Following, *location* is defined as *the place in the document semantics where the user is editing* and *distance* as *the distance between the semantic regions being edited and other points of interest*.

An example of how the distance between users influences the consistency boundaries is the one given in Fig. 1. In this picture, both user A and user B are editing the first section but on different paragraphs. User C on the other hand, is editing the second section. Thus, user A is probably working on a different subject than user C. As for users A and B, they must be careful because their writing about the same topic. Hence, consistency can be weakened between user C and the remaining but strengthened between users A and B. This way, both A and B know with great precision what the other one is changing, but not what C is modifying

In this work, as seen in the example of Fig. 1, the focus will be the *edition of text documents*. But the adaptation of the VFC model should be sufficient generic to be easily ported to other cooperative work environments like *spreadsheets* and *presentations*.

In the next section, the objectives of this work are presented. In section 3, we start by exploring the main issues related to consistency maintenance in distributed systems, and then we overview the importance of cooperative work tools followed by a description of some examples. In section 4, the proposed architecture for the problem in this work is explained. Later in section 5, a series of quantitative, qualitative and comparative parameters are presented for a future analyzes of the success of this work. Finally, in section 6, some conclusions are taken.

2 Objectives

The main goal of this work is to adapt the *Vector-Field Consistency* algorithm to the cooperative work scenario, namely document-based cooperative work. The second is to integrate the new adapted algorithm, to which we call *VFC for Cooperative Work* (VFC4CW), in a cooperative work tool. This integration should improve performance (w.r.t. network usage) and enhance the usability of such tools in the cooperative scenario.

The work will begin with a study of consistency enforcement in distributed systems for cooperative work. This will provide a perspective of the benefits and flaws of the studied techniques, which will be used to propose an architecture that fits the requirements of the problem stated in the previous section.

The most important part of this work will be the adaptation of the VFC algorithm to a new environment. This will be achieved by porting the VFC core concepts, which now suit the ad-hoc gaming environment, to a new scenario where notions like distance and user position are not adequate in their original definition.

The adapted algorithm, which will be referred to as VFC4CW, will be incorporated in an established cooperative work tool, and/or used to create a generic middleware layer specific to the algorithm. To measure the success of this work, the resulting product will be evaluated using a number of criteria (listed below and in more detail in section 5).

In sum, the objectives of this work are:

- Study of the state-of-the-art of consistency enforcement in distributed systems and cooperative work;
- Adapt the VFC algorithm to the cooperative work scenario (VFC4CW);
- Integration of VFC4CW in a cooperative work tool and/or design of a VFC4CW generic middleware layer to enforce the algorithm in text-based documents (e.g. plain text, wiki, HTML, ODF[5]);
- Evaluate the developed system in terms of the following criteria:
    - *Qualitative*: conformity with the model, maintain user expected properties and functionality;
    - *Quantitative*: performance results, application and user interaction overhead, bandwidth usage;
    - *Comparative*: using some of the above criteria against a selection of other approaches found in literature.

3 Related Work

The development of cooperative distributed tools implies a need to share data across several replicas. In general, distributed systems, like cooperative tools, need to ensure the consistency of replicated data.

The following sections address the two principal themes studied to determine the state of the art of the Cooperative Work in Distributed Systems field. In section 3.1, the topic of *Consistency in Distributed Systems* [Saito05] is presented. The differences and usefulness of a series of models and algorithms in this area are discussed. In section 3.2, some of the existing *Cooperative Work Tools* [Rodden91, Rama06] are introduced as well as their strengths and weaknesses. This set is mainly composed by tools and algorithms that focus on edition of either plaintext files, structured documents or shared wikis.

3.1. Consistency in Distributed Systems

In a Distributed System, higher availability and performance is often accomplished using data replication. Data replication enables applications to work even when some replicas are unavailable. Moreover, several users can access data at the same time and without the need to contact some remote and possible busy location. The downside of data replication is the possibility of replica divergence/ inconsistency. The balance between consistency and availability is a characteristic that separates systems in two opposite classes: *optimistic replication systems* [Saito05] (section 3.1.2) and *pessimistic replication systems* (section 3.1.1). However, sometimes neither the pessimistic nor the unbounded optimistic approaches are acceptable to applications. Thus, it may be beneficial to explore the semantic space between the two alternatives. Hence, in section 3.1.5, the existing *continuous consistency models*, their virtues and flaws are presented. Finally, in section 3.1.6 the idea of mixing the state-transfer and operation-based approaches to provide efficient operation representations is overviewed.

[5] Open Document Format

3.1.1 Pessimistic Approach

Pessimistic replication systems provide single copy consistency by allowing only one user at a time to perform alterations to a replica. Therefore, the remaining users block until the first user finishes. This approach prevents the existence of conflicts and offer guaranties of strong consistency between replicas since the systems do not speculate whether it is safe or not to update data. Although the use of pessimist replication algorithms [Bernstein86, Dietterich94] can work well in local-area networks, when ported to a wide-area network such as the Internet, these algorithms cannot provide good performance and availability.

3.1.2 Design Choices in the Optimistic Approach

In opposition to Pessimistic techniques, Optimistic Replication assumes that conflicts will be extremely rare and can be fixed later whenever they appear. For this reason optimistic algorithms do not require *a priori synchronization* with the other replicas to perform an update. In result, these systems offer greater *availability*, *flexibility*, *scale better* and enable *asynchronous collaboration* even in wide-area environments. As for consistency, in optimistic algorithms, replicas may only converge eventually and so, controlling the differences between them in each moment is of the most importance. In result, these algorithms can only be deployed in systems that can support partially inconsistent data, even if only temporarily.

In optimistic replication [Saito05], a series of design choices that define these optimistic systems can be considered. Those choices are defining characteristics that should reflect the consistency requirements of optimistic replication systems. They are: *number of writers*; *definition of operations*; *scheduling*; *handling of conflicts*; *propagation strategies and topologies*; *consistency guarantees*. The analysis of these criteria provides a great overview of the most important issues of optimistic replication.

Number of Writers: single-master vs. multi-master.

Single-master or *caching* systems are those in which only one replica can submit an update. Although easy to implement, these systems have very little availability. In comparison, *Multi-master* systems [Xia04, Weiss07, Preguiça09] are much more complex but offer greater availability. They let various replicas update content simultaneously and exchange the modifications in background. Multi-master systems, in contrast with the single-master, need to deal with the issues of scheduling and conflict handling.

Definition of Operations: state-transfer vs. operation-transfer.

Considering how modifications are propagated and exchanged between replicas, *State-Transfer* [Satyanarayanan90, Santos07] systems are a special class of systems that considers every operation to an object as a modification of the entire object. Although this may seem ineffective (and in fact, most times it is), there is some interesting potential behind this technique: for instance, replicas can easily converge by receiving the most recent contents without the need to apply every missing update. This use of the state-transfer approach can be very useful for example when a site stays offline for too long.

In opposition to State-Transfer systems, *Operation-Transfer* [Ellis89, Cart07, Shapiro07, Weiss09] systems offer a more flexible conflict resolution and reduced bandwidth requirements at a cost of higher algorithmic complexity. To achieve that, these systems transfer one or more operations that correspond to the user's changes instead of transferring the entire replicated object. This technique is particular useful when dealing with large and high level objects.

Nevertheless, when none of the above approaches meet applications requirements, hybrid solutions [Muthitacharoen01, Barreto05, Veiga05] may be useful. In section 3.1.6, the benefits and costs of such solutions will be analyzed in more detail.

Scheduling: syntactic vs. semantic.

Eventual consistency is one of the most defining characteristics of optimistic replication. This concept means that all replicas will eventually come to the same value when users stop the commitment of new updates. To reach eventual consistency, replicas have to produce equivalent schedules (which produce equivalent final state). There are a number of systems that employ different types of scheduling. On one hand we have the *syntactic scheduling* [Satyanarayanan90] approach, which tries to define a total order of operations based on their submission timing and location. Coda [Satyanarayanan90] for example implements this type of scheduling using scalar timestamps.

On the other hand we have the *semantic scheduling* [Sun96, Preguiça09, Oster06] approach, which exploits operation semantics to either transform (section 3.1.3) or commute operations (section 3.1.4). This approach has the advantage of reducing rollbacks and augmenting scheduling freedom in order to find the "best" (in the application point of view) possible schedule well-suited with the existing constraints. Since semantic scheduling requires systems to have access to some sort of semantic knowledge about objects, it can only be used in operation-transfer systems.

Conflict Handling: syntactic vs. semantic detection & manual vs. automatic removal

Another important characteristic that separates various optimistic replication systems is the way they *handle conflicts*. Conflicts happen whenever operations do not respect their *preconditions*. Managing conflicts has two phases: *detection* of the conflict and *resolving* it. Numerous systems choose not to do any type of conflict handling, although at the cost of an increase in number of lost-updates and of truly difficult data management. As in scheduling schemes, also conflict management techniques cover the spectrum between syntactic and semantic approaches. *Syntactic conflict detection* finds conflicts through the use of the *happens-before* [Lamport78, Raynal96] relationship. This means that every group of concurrent operations will be flagged as in conflict. *Semantic conflict detection* policies on the other hand use semantic information about the replicated objects to determine if there is in fact a conflict. This method, although not as efficient as the syntactic conflict detection, can benefit a lot from the lack of false conflicts.

Once a conflict has been detected, it is necessary to resolve it. In the resolution of conflicts, the objective is to rewrite or abort operations in order to remove conflicts and it can be either *manual* or *automatic*. Manually resolving conflicts is not much of a challenge since it simply presents two versions of the replicated object and let the user decide the final version. As for automatic resolution of conflicts, there are several methods to handle this: an interesting one is the Bayou [Terry95] example, which attaches to each operation a precondition and a merge procedure. Then, before applying an operation, if the precondition is being violated, the merge procedure is executed and the necessary fix-ups preformed. Although this can be an appealing approach, studies showed that writing these merge procedures is truly difficult in most situations.

Repeatedly, a problem with conflict management is to determine when an operation is stable, that is, when its outcome result is no longer in a tentative state. This knowledge is useful for applications because, once an operation gets stable (committed) it can be removed from logs. In this context, *commitment protocols* play an important role. Some of these

protocols apply what is generally called *agreement in background*; this solution however is based on vector clocks and equivalent structures and so do not scale well. Another much more complex solution is to apply a *consensus* algorithm to agree on the order in which operations are committed [Çetintemel03].

Propagation Strategies and Topologies: pushing vs. pulling & fixed vs. ad-hoc

The *propagation of updates* can be analyzed in terms of the *communication topology* and *degree of synchrony*. When dealing with fixed topologies, solutions can be extremely efficient, however, if the network is not structured or it changes dynamically the best solutions rely on *epidemic propagation* algorithms. Regarding the degree of synchrony, there are pull-based systems, i.e. systems in which each replica requests some set of the remaining replicas for their most recent updates, and push-based systems, i.e. systems in which each replica proactively sends their own updates to other replicas. Although this depends on each case, in most scenarios the quicker the propagation, the lower the conflict rate is.

Conclusions

A few conclusions can be extracted from this section: First, *Single-Master* systems are good for read-intensive or single-writer applications. Also, *Multi-Master State-Transfer* systems are fairly simple and have low memory requirements and so they're adequate for most replicated systems. Additionally, their bandwidth needs do not increase with the rate of updates since they can be easily merged into one. These systems have the disadvantage of not dealing well with conflict resolution because of their all or nothing update approach. The problems with semantically rich conflict resolution can be solved using *Multi-Master Operation-Transfer* systems, although there is a price to pay in terms of algorithmic complexity and log space overhead.

Next, a review of the principal characteristics of the two major *semantic scheduling* schemes is made in sections 3.1.3 and 3.1.4. The first, the most mature of these techniques, is *Operational Transformation*. Then, *Operation Commutativity*, a younger but very promising technique.

Finally, two types of *mixed approaches* are presented: in section 3.1.5, an approach that offers the possibility of bounding consistency in a continuous spectrum between *pessimistic* and *optimistic* consistency; in section 3.1.6, a combination of the *transfer-based* and the *operation-based* approach, in order to achieve more efficient representations.

3.1.3. Operational Transformation

Operational transformation is a *semantic scheduling* technique originally designed for consistency maintenance in group editors. These systems have very specific constraints, namely short response times and support for free and concurrent editing (similar to *Google Docs & Spreadsheets*), which can be fulfilled by the use of this technique.

The OT approach, pioneered by the *Grove* [Ellis89] system in the late 80's, consisted in the principle that an update should by immediately executed in the local replica after its creation and only then propagated to the remaining replicas. Then the remote replicas transform the operation right before its execution without needing to reorder the previous operations.

Grove made the contribution of identifying two inconsistency problems that would happen if operations were executed in remote locations without a previous transformation step and as soon as they arrive: *divergence* and *causality-violation*. First, considering that operations are not commutative between then, if executed in different orders, the final editing

result will be different. This is the divergence problem. To understand the second problem, out of causal order execution, one can imagine the situation were replicas transmit their operations without synchronization. In this case, since operations are executed immediately after arrival, the execution order might not respect the natural causal order.

Grove then defined two correctness criteria to solve the previously identified consistency problems: (1) *convergence property*: all generated operations have been executed at all sites; (2) *precedence property*: if one operation O_a causally precedes another operation O_b, then at each site the execution of O_a happens before the execution of O_b. Grove system has two main components: the state-vector time stamping scheme for guaranteeing (2); and what became known as the *distributed OPerational Transformation* (dOPT) algorithm for ensuring (1). With the dOPT algorithm, every operation that obeys to the first criteria is transformed against an already executed operation which is independent with the first. This transformation should be done in such way that every site that executes the same set of operations produces the same end state.

Later, *REDUCE* [Sun96] identified a third inconsistency problem, called *intention violation,* which consists on the execution of an operation that, if executed after another concurrent operation, may not change the content as expected by the user. This problem may seem similar to the divergence problem but, as proven by its solution, it is not. In fact, the divergence problem can be solved just by employing a serialization protocol, but the intention violation problem cannot. This happens because, even though replicas are consistent, the final state might not reflect the users' expectations. Considering the intention violation problem, REDUCE defined the subsequent consistency model, which is commonly called *CCI consistency model*:

A cooperative editing system is consistent if it always maintains the following properties:

- *Convergence*: when the same set of operations has been executed at all sites, all copies of the shared document are identical.

- *Causality-preservation*: for any pair of operations O_a and O_b, if $O_a \rightarrow O_b$, then O_a is executed before O_b at all sites.

- *Intention-preservation*: for any operation O, the effects of executing O at all sites are the same as the intention of O, and the effect of executing O does not change the effects of independent operations.

In opposition to Grove, in the REDUCE approach an undo/do/redo scheme is used for achieving convergence. As for intention-preservation, a new operation transform algorithm was applied, the *Generic Operational Transformation* (GOT) control algorithm. This consistency model and its implementation successfully solved the scenario in which Grove didn't work.

An optimization to the GOT algorithm by the authors of Grove and REDUCE, called *GOT Optimized* (GOTO), can be found in [Ellis89]. This algorithm allows a similar approach to the one used in REDUCE but without the need for the undo/do/redo scheme. Also, by performing transformations on both the creation and execution contexts, this algorithm is able to reduce the number of transformations.

Since the appearance of the Grove and REDUCE systems, many other algorithms, optimizations and even scenarios of application were presented for Operational Transformation. Numerous systems, such as Jupiter [Nichols95], TreeOPT [Ignat03], CoWord

[Sun04, Xia04], MOT2 [Cart07], UNO [Weiss08] and FEW [Bento06] successfully applied this technique especially in the field consistency maintenance. An interesting extension to the original Operational Transformation scheme, which was designed by CoWord, is the addition of support for the update primitive (originally, all user alterations were translated to sets of insert and delete primitives). In MOT2, the OT approach was successfully ported to the P2P environment. In fact, with MOT2 any site can reconcile its copy at any time with any other site that owns a copy of the object while achieving overall convergence of copies. Finally, UNO not just ports OT to the P2P environment, it also proposes a new extension to support the undoing of any applied operation.

3.1.4 Operation Commutativity

In the cooperative editing environment, it's natural that replicas diverge if they don't execute operations in the same order. To address the problem of replica converge there are several techniques. One of them is the previously presented *Operational Transformation* (section 3.1.3) but, as said in [Preguiça09], OT is too "*complex and error-prone*". An alternative solution, and the one in focus in this section, is *Operation Commutativity*, the condition that every pair of operations is in commutative relation.

Operation Commutativity aims to the *automatic convergence* of replicas, i.e. convergence without the need of any complex concurrency control (e.g. lock or serialization). To achieve automatic convergence, it is sufficient the use of a *Commutative Replicated Data Type* (CRDT), as it was baptized [Shapiro07].

The CRDT approach considers that a document is formed as a sequence of *atoms* each univocally described by an identifier that does never change during the life span of a document. An atom can be any non editable element like a character or a graphics file. The identifiers' space must be dense and their total order must reflect the order of appearance of atoms in the document. These requirements suggest that rational or real numbers could be used as identifiers, however, they would require infinite precision which is not viable.

The *TreeDoc* [Shapiro07, Preguiça09] is an implementation of the identifier's structure based on binary trees that represent the document elements/atoms. The total order of those elements can be translated from walking the tree in infix order, which means that the identifiers can be obtained from the tree paths. Since binary trees cannot by themselves support concurrent edits, *(major) nodes* in the identification tree can contain several *mini-nodes* to represent those edits. These structures must be identified inside the major-node by a *disambiguator* which may be either a *unique disambiguator* (UDIS: $<site_{id}, site_{counter}>$) or a simple *site disambiguator* (SDIS: $site_{id}$). Even though the first may seem inappropriate since it has a greater overhead, the latter imposes the use of tombstones, which can only be safely deleted when all sites are alive [Saito05]. To fully understand some of the former concepts a more careful reading of [Shapiro07, Preguiça09] is advised. The proposed TreeDoc algorithm for generating new identifiers can be improved by combining it with algorithms to balance the identifier tree.

Another replication mechanism that was developed earlier is presented in [Roh06]. Although the solution is similar with the CRDT approach, it relies on tombstones and vector clocks and so it has problems to scale to massive collaboration environments. Additionally, this solution has the problematic issue of being destructive and losing work. Thus, it cannot be applied to cooperative environments.

One more system that implements operation commutativity is the *Wooky* [Weiss07] system which was based on the *Woot* [Oster06] framework (Section 3.2.2). Even though this system was developed independently from TreeDoc, they have many similarities. In fact, Wook also bases its design in the idea of creating non-destructive operations with unambiguous

identifiers that live as long as the document does. One significant difference between these two approaches is the way Woot structures the documents and identifiers: a linear structure stores elements in their order of appearance in the document, each one indentified by <$site_{id}$, $site_{counter}$>. Even more, Woot does not allow the removal of the identifiers of deleted elements. This results in a constant growth of the supporting data-structure.

The same authors that created Woot [Oster06] and Wooky [Weiss07] later developed the *Logoot* [Weiss09] model. The objective of this model is to ensure CCI consistency (see section 3.1.3) and scalability both in number of users and updates so it can be adapted to massive collaboration environments (e.g. over a P2P network). In consequence, one key design goal of this model is not to be tied to tombstones because of their scalability and performance problems. Like the previous examples of TreeDoc and Woot, this model is based on non-mutable and totally ordered identifiers [Weiss09]. Since the identifiers are totally ordered, they can be removed without affecting the order or the remaining identifiers.

3.1.5 Continuous Consistency Models

Designers of replicated systems conventionally choose between strong and optimistic consistency models. Although sometimes, neither the performance overheads imposed by strong consistency neither the lack of limits for inconsistency are acceptable to applications. In such cases, it's appropriate to explore the semantic space between these two alternatives. The fundamental idea behind these *continuous consistency models* is that this space is a continuum parameterized by the *distance* between replicas. This distance is zero for strong consistency and infinite for optimistic consistency. The distance measure can be used to provide a per-replica consistency based on the expected amount of conflicting updates.

Leverage of the consistency space continuum allows systems to correctly balance applications availability and consistency. This balance is affected by factors like application workload, read/write ratios, probability of simultaneous writes, network latency, bandwidth, error rates, etc. Some of the work developed in this field is based on bounding consistency along a single dimension, e.g.: "*maximum time without being made consistent*"; "*maximum number of uncommitted updates*" [Krishnakumar94]. Even though studying single dimension consistency bounding may be interesting, this section will focus in more comprehensive models [Yu00, Santos07, Barreto09] with greater expressive power.

The TACT framework

In [Yu00] is presented a middleware called TACT, which enables applications to quantify their consistency requirements. Once those requirements are defined, the TACT framework only lets one operation proceed locally if the consistency bounds are not currently being violated. Otherwise, TACT blocks the operation and synchronizes with the remaining replicas until the fulfillment of the pre-defined consistency requirements.

With TACT, applications need to specify their *conits*, i.e. the physical or logical unit of consistency. For example, in an airline reservation service, the conit could be an individual flight, or a block of seats or even a single seat. Defining the granularity of conits depends solely on the application necessities. The quantification of divergence boundaries is made on a per-replica basis through the use of three metrics, *Numerical Error*, *Order Error*, and *Staleness*. The first metric, *Numerical Error* bounds the difference between the local value of the conit and the value of the "final image", i.e. the image in which all tentative updates of all replicas have been applied. The implementation of this metric can be tricky since, if application conits don't represent a numerical value, a numeric representation (a weight) must be specified. Also, this metric relies on the cooperation of all replicas and thus, cannot be

dynamically changed without a consensus like protocol. On the other hand, *Order Error* can be controlled using only local data, since it bounds the maximum number of tentative writes at a replica. In other words, it bounds the number of local writes that may still be rolled backed. Finally, *Staleness* is the maximum value of time between the last seen local write and the current time. To bound this metric each replica holds a real-time vector, in which each entry corresponds to the real time passed since the last seen update from each replica.

This model enables the definition of per-replica consistency bounds, which allows systems to greatly adapt to their consistency requirements. For instance, one replica with limited network access may relax its consistency limits. Oppositely, in a replica with faster links it may be viable to impose a stronger consistency. Another interesting application of TACT is the routing of clients to different replicas (with different divergence bounds) according to their profile. Balancing consistency to meet each replica needs [Yu00, Yu06] can have serious impacts in system performances. In fact, most times, applications can have significant performance improvements without compromising correction by slightly relaxing consistency.

The Vector-Field Consistency model

Like the previously presented TACT framework, *Vector Field Consistency* (*VFC*) [Santos07] is a consistency model that enables replicas to define their consistency requirements in a continuous consistency spectrum. Other than simply *bounding divergence* on a per-replica basis, VFC allows more powerful consistency enforcement policies. For example, using VFC, the maximum allowed difference between two replicas can be dynamically changed during the execution of an application.

Indeed, the novelty of the VFC model is that it combines divergence bounding with the notion of *locality-awareness* to improve the availability and user experience while effectively reducing bandwidth usage. To understand how locality-awareness affects this model, the concept of *pivot* must be introduced. Pivots roughly correspond to each user's observation points within the data and their position (w.r.t. some set of coordinates) is expected to be volatile. Also, consistency between replicas should strengthen as the distance between their pivots shortens. 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 the TACT framework, programmers use a 3-dimensional vector: *time*, *sequence*, *value*. These boundaries should be specified in a way that does not compromise the user's experience. In other words, all the required information is presented to users with enough quality and they cannot perceive much difference in their use of the applications.

The VFC model is extremely flexible with regard to the possibility of specifying different consistency boundaries. This flexibility allows VFC to be used in a wide variety of systems with very different consistency enforcement policies. Moreover, VFC is simple and intuitive in such way that developers can easily express and parameterize their consistency requirements in accordance with the application requirements. Also, VFC effectively reduces the network bandwidth requirements by selectively choosing which updates are more important to which replicas and delaying less important ones, possibly omitting some superseded by later updates.

In comparison with VFC, the TACT framework can enforce the same consistency scenarios but, since it does not support locality-awareness, it cannot be applied to scenarios where consistency depends on the notion of the user's position within the data.

Although Vector-Field Consistency was initially design to fit the environment of distributed game development, the concepts of pivots, consistency zones and distance between

replicas are sufficiently general to be easily applied to other distributed environments like cooperative work tools based on shared data.

Data-aware Connectivity

The interest about the concept of *data-aware connectivity* system [Barreto09] is the use it makes of a continuous consistency model. The continuous consistency model is used to determine the *quality* of a replicated object. Then, unlike in the previously presented works, using the ascertained current value of quality, the system *regulates connectivity* to enforce the divergence/quality bounds. Only when all connections were explored without success it forbids access to the replica.

This way, the connectivity costs can decrease because only the exactly required connections will be used to achieve the imposed quality, which can be very useful in environments with great constraints like mobile environments.

These systems have two key components: the *quality monitor* to estimate the quality of each available replicated object; the *connectivity regulator* to enforce a certain level of replica quality. Quality is influenced, among other criteria by its *freshness*, *consistency* and *possibility of rapid commitment*, which are represented by a group of variables: (1) time since the last synchronization from each replica; (2) number of local tentative updates; (3) commit weight of currently inaccessible replicas; (4) number of known concurrent updates; and (5) recent update activity by other replicas to the object. The first (1) variable represents how fresh the replica is. The second (2) can be used to determine consistency. The remaining (3-5) determine the probability of a quick and successful update.

Another important issue about being up to the system to determine replica quality is the fact that these metrics, although intuitive to the attentive user, are not easy to evaluate by the human user. Thus, it is less error-prone to let the system deal with connectivity issues and provide the user an indicative value for the quality of replica so he can decide if he wants to perform a certain task.

3.1.6 Efficient Update Representations

The *definition of updates* has two well-known solutions: *state-transfer* and *operation-transfer*. While the first can be very easily and transparently adapted to existing commercial solutions, the second solution promises high concurrency and lower conflict rates. The idea of mixing the two solutions to get a transparent and yet highly available middleware for distributed systems can be very attractive.

The proposal of *Chunks* [Muthitacharoen01, Barreto05] and *Semantic Chunks* [Veiga05] is to exploit content similarity with the intention of reducing the *bandwidth and memory requirements* and in the case of semantic chunks the amount of *update-conflicts* due to false-sharing.

Chunks were implemented in the *LBFS* [Muthitacharoen01] and later in the *Haddock-FS* [Barreto05] with the primary objective of seriously reducing the occupied memory and bandwidth. To accomplish this goal, an *application transparent* middleware was developed, which takes advantage of cross-file and cross-version similarities. In these systems, every file is partitioned in several parts called chunks, which, in order to be resilient to insertions and deletions within the file, are divided by a content-based approach. By applying the *SHA-1* [Eastlake01] function to each chunk, they can be unambiguously identified. Using these unique identifiers replicas can then create a chunk repository to reduce memory. Also, by previously exchanging these identifiers, replicas can determine exactly which chunks must be transferred, thus reducing data sent over the network. Due to the great network and bandwidth

reductions achieved, the use of chunks allows distributed file systems to be ported to scenarios of slow or wide-area networks.

Semantic-Chunks on the other hand, are semantically richer than Chunks. Since they can be independently considered for consistency issues, an increase in concurrency and a reduction (in number and cost) of update conflicts is expected. The Semantic-Chunks System consists in a middleware in between the OS/VM and the Office Applications that stores the semantic structure of the documents requested by those applications. These semantic structures (semantic-chunks) are aggregations of either lower-level semantic-chunks or chunks of data. This way, it is possible to dynamically obtain the documents as their sub-regions are required by the system or as new updates are submitted. Whether the data chunks are paragraphs, sentences or even characters, it is not defined. This decision can even be made dynamically so the system can adapt to various consistency requirements. The semantic-chunks middleware is completely transparent to the above applications.

3.2. Cooperative Work Tools

Although the design of cooperative software is an ancient and well-studied field, being a comprehensive and somewhat, fairly subjective area, naturally it is not one of much consensus. To address this issue, this section will start with an overview of the positive and negative aspects of *cooperative writing* [Tammaro97, Noël04] as well the desired properties of good *cooperative writing tools* [Grudin88, Noël04]. Later, the concept of *CSCW* and their core issues will be introduced [Greif88, Grudin88, Bannon91, Rodden91, Rama06]. Finally, some of the most relevant *distributed document editors* and *replicated wiki* systems will be examined (section 3.2.1 and 3.2.2).

Cooperative Writing

The process of writing an article, a review, a book or even a simple draft may be long and complex. To ease the workload and consequently reduce the time consumed by writing tasks and achieve better results, writers tend to group together. Also, sometimes is natural that the work chores, like participation in reporting committees or in a research cooperative project, involve projects of joint writing.

If any cooperative face-to-face project is difficult, due to the problems of team management, a cooperative writing project is even more since it usually involves people in remote locations working together in an almost independent way. In these scenarios, the lack of communication and acknowledgement of other writers' changes, can easily lead to problematic events like writing the same thing more than once and accidentally deleted contents.

Nonetheless, the cooperative writing has very encouraging benefits, namely: *reducing task completion time*; *reducing errors*; *combining different viewpoints and skills*; and *obtaining an accurate document* [Tammaro97, Noël04]. Also, writers tend to feel more involved with the outcome of the project and thus, contributing with *more time and effort* to the writing process.

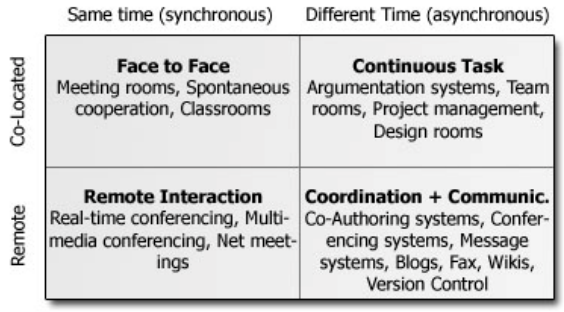

Fig. 2: Time/Space Matrix

Cooperative Writing Tools: What users want?

Although, as discussed, group editing has many great benefits, cooperative writing tools are nowadays still used only in a small part of cooperative writing projects. Most collaborative writers use simple personal word processors [Noël04].

This lack of usage of group writing tools is in part due to the idea that they difficult the cooperative tasks instead of helping, which can be in fact true because systems tend to be designed without taking in consideration how groups really collaborate. Also, experienced users tend to be extremely reluctant about changing their writing software.

To mitigate the existing inertia in changing from non-cooperative to cooperative tools, a suggestion is, instead of developing a cooperative writing editor, to integrate cooperation features into the already existing and utilized software. On the other hand, some authors suggest that adding cooperation features to the already huge set of features offered by these editors, would create such a mess that they would become useless [Noël04].

Enquired users [Noël04] said that the cooperative features they would like to see in their writing tools were: *easy perception of recent modifications*; *easy addition of notes* (distinct from normal text); *incorporated communication channels*; and *text locking functions*. Curiously, on the opposing side, some users completely refused the idea of using a collaborative writing tool: "*The system is not nearly as important as the people with whom one writes. (…) I think a phone, a fax, or email are perfectly suitable*".

As we can see, investigators have not yet come to a conclusion about what should be a cooperative writing tool. There are already suggestions about interesting features but there are still no so many good examples of successful tools.

Computer Supported Cooperative Work (CSCW)

The idea of CSCW [Greif88] was initially introduced by *Irene Greif* and *Paul Cashman* in 1984 to refer to the set of concerns about *supporting multiple individuals working together with computer systems*.

Nowadays, the term of CSCW is somewhat a confusing term. In [Bannon91] a simple analysis to the words of the acronym can explain the wide-coverage of this field. The expression *cooperative work* refers to multiple persons working together to produce a product or service. On the other hand, the second term, *computer supported*, does not limit so much the forms of interaction and organization. In result, CSCW is now a huge field that generically refers to the understanding of the way people work in groups and the associated network technologies, hardware, software, services and techniques.

As CSCW is a broad field, it may be beneficial to focus in just some part. Well, a term similar to the CSCW (some authors even considered them to be the same) is *Groupware*. Groupware is more focused on the enabling technologies which allow work in group, rather than the psychological, social, and organizational impact [Rama06]. From now on the term CSCW will be referring to the same as Groupware, unless when addressing some specifics of one of them.

Due to the great variety of CSCW systems, to help distinguish the ones that matter from those that do not. An interesting division can be made by considering two key characteristics: *space* and *time* (Fig. 2). The first is about the *form of interaction*, i.e. if systems work synchronously or asynchronously. The second is about the *geographical nature of the users*, i.e. if working groups are co-located or remotely located. Using these distinction criteria, CSCW systems can be roughly divided in: *Message Systems*, *Computer Conferencing*, *Meeting Rooms* and *Co-Authoring and Argumentation Systems* (these are the designations presented in the '91 *Survey of CSCW System* [Rodden91]). However, nowadays this division does not create disjoint groups because, has systems evolved, so did their original characteristics.

Message Systems are an evolution from the electronic mail systems, from which they inherit the message exchange paradigm for cooperative work. Like in electronic email, message systems work *asynchronously* and *remotely*. The main difference between these systems and their ancestors is the evolution from a simple exchange of text messages to richer and structured message objects with specific attributes.

A *Computer Conferencing* system is a collection of *conferences/conversations*, each with a group of interested members and (desirably) a single topic. Also, usually each user can know what is new in each conversation he follows. Computer Conferencing started as simple *asynchronous* systems with shared textual information. They later *evolved to synchronous* systems to deal with real-time cooperative scenarios like crisis management. They also evolved to the so called multi-media conferencing systems, which due to network bandwidth improvements were able to integrate audio, text and video in the previous scheme.

Meeting Rooms are a special form of cooperative working paradigm where the participants work in a *synchronous* way and are *co-located* in a room equipped with information display technology and a workstation per member. Most of these systems were created for the proposed of decision making and for this reason they often incorporate statistical and analytic decision models. One interesting development was a multi-user interface with similar views for every participant. These solutions come although with a series of problems, like the distraction caused by the display of other participants' activity and the high bandwidth requirements of such scenarios.

The objective behind *Co-Authoring and Argumentation Systems* [Xia04, Veiga05, Weiss07, Oster09] is to support the development of co-authored documents. The idea is to have each author working on a portion of the whole document in an *asynchronous* manner *independently of their physical proximity*. This model of cooperation can be highly improved if dealing with structured documents since they permit easier isolation of user changes and conflict resolving. Also, as this field of investigation matured, it became obvious that the initial solutions where every user accessed a shared storage system did not work and decentralized solutions were established. In consequence the notion of private and master views/contexts appeared.

Although the proposed classes of systems were intended to divide systems, in most cases systems do not belong uniquely to one class. In fact, the most common situation is that these cooperative work systems combine a variety of characteristics and functionalities from each of the classes presented.

In the next sections some interesting CSCW systems related to text edition will be introduced. In section 3.2.1, *simple document edition systems* will be analyzed. Later, the section 3.2.2 will rather focus on *distributed wiki systems*. Finally, in section 3.2.3 we will discuss the importance of *intra and inter-document distance* in cooperative writing tools.

3.2.1. Distributed Document Editors

The Transparent Adaptation approach: CoWord

Single-user computer applications (text editors, graphics drawing tools, word processors, spreadsheets, presentation tools, web design tools, etc.) are widely present in our daily lives and work. Thus, since the concept of cooperative work is raising its visibility, it is natural to think about adding cooperation support to these applications.

Although the existing cooperation techniques were vastly studied by researchers, they were only applied to prototypes. Hence, it still remains to be determined the true applicability of such solutions in daily used mature applications. In [Xia04] the authors propose how this leveraging between mature single-user application and state-of-the-art cooperative techniques can be carried out.

The continuous improve of current cooperative prototypes to achieve the same richer environment provided by single-user applications is not a good solution as users will not likely be willing to learn and use a new application just because of some less frequently-used functionalities [Grudin94]. On the other hand, modifying existent single-user applications to support cooperative features may not be possible, not because of the implementation efforts, but because most of them are closed-source. Additionally, separating single-user functionalities from cooperative issues may be advantageous.

The solution proposed in [Xia04], called *transparent adaptation*, is to provide applications with cooperative abilities without changing their source-code. The goal is to have multiple users work on their normal single-user applications in an *unconstrained cooperation*, which means that users access data in a concurrent and freely way. Ensuring that users can be performing changes in the same region or adjacent areas, without using complex merging schemes, is resolved by the *Operational Transformational* technique.

To implement the transparent adaptation approach, below the single-user applications two new middleware layers must be created: (1) the *Operational Transformation Layer* to provide eventual consistency guarantees; (2) the *Adaptation Layer* to translate the linear operations performed in the application into simple primitive operations perceivable by the OT Layer.

To increase system modularity, reduce design complexity and promote component reusability, the application specific functionalities and the cooperation capabilities were completely separated. Indeed, the OT Layer is generic and application independent. This way, only the Adaptation Layer must be written in accordance to the single-user application; the bottom-most layer is the same for every application.

The transparent adaptation technique was successfully used to create *CoWord* and *CoPowerPoint*, the cooperative versions of MS Word and MS Power Point. However, this solution can only work in applications with a suitable API which can be used to intercept and replay user input events, and whose data and operational models are adaptable to the underlying OT technique. If not, implementing the Adaptation Layer either will not be so straightforward or it will even be impossible. Additionally, this solution does not provide any kind of awareness and so users may not be able to easily coordinate themselves.

The FEW file system

The *Files EveryWhere* [Bento06] is a distributed file system for mobile computing that provides high availability, good performance and low energy consumption. It works with a generic *operational transformation* reconciliation strategy combined with an *epidemic dissemination* algorithm. This means that users will work *asynchronously* and *concurrently* without the need to perform any conflict resolution task. On the other side, the reconciliation mechanism must be adapted to the specifics of each file type.

Updates in the FEW-FS are propagated using first a *best-effort propagation* system that accelerates convergence and then, to guarantee eventual convergence between replicas, they perform *periodic pair wise epidemic propagation sessions*. In these sessions, replicas send and receive al the unseen updates regardless of their origin.

To merge concurrent updates from different users in order to achieve a common final state in every replica, FEW first translate updates into semantically-rich operations. To do this translation it uses a type-specific program, which compares the previous and the new version of the file. Afterwards, the inferred set of operations is propagated to the remaining replica (using either of the previously mentioned strategies). When an operation arrives to a certain replica, it is integrated employing the generic operational transformation algorithm (the GOTO [Sun98] algorithm in this case).

The *reconciliation method* presented in [Bento06] is suitable only for text files. However, the solution can be adapted to other types of files. This approach, allows maintaining multiple versions for each line as users modify them concurrently, much like the CVS approach. Yet, it allows the merging of multiple versions to be postponed and so to continue the modifications even if they include modifying the lines with multiple versions. This way, there are no lost updates and the user's intentions are preserved since operations are not automatically merged.

Additionally to the presented method, the authors propose also two solutions to deal with files regardless of their content. The first consists in choosing one of the versions for all replicas. The second solution will keep all versions originated by concurrent updates so that the user can later access and eventually merge them.

3.2.2. Distributed Wikis

The Wooki

The increase popularity and importance of *wiki systems* in the daily life of Internet users and mainly of institutions (academic, research, non-governmental organizations, etc.) and corporations unfolds a series of critical issues related to availability. In fact, nowadays, most popular wikis are designed in a purely centralized infrastructure. This means that availability can be compromised in case of *failure*, *heavy load* or *offline access*. If, on the other hand, wikis were replicated in a *P2P network* there would not be any problems with *fault tolerance* or *load balancing*. Additionally, even the *hardware costs* could be divided between all the peers.

As the solution of replicating wikis effectively handles the problems of the centralized version, it arises the problem of *maintaining consistency*. The *Woot* [Oster06], overview in section 3.1.4, algorithm can be used to deal with this consistency problem. This algorithm is improved and incorporated in a fully functional P2P wiki system called *Wooki* [Weiss07]. Additionally, to solve the update propagation problem which is not covered in [Oster06], the wooki system uses a *probabilistic dissemination algorithm* combined with an *anti-entropy algorithm* for managing failures or disconnected sites. Also introduced with *wooki* is an

improved version of the woot algorithm called *Wooto* [Weiss07], which achieves better performance and has smaller memory requirements.

In wooki, as usually in wiki pages, users make all their changes to the page and only then save them. Subsequently, a diff algorithm is used to translate these modifications in wooto operations (using the *delete* and *insert* primitives). Then, operations are immediately executed locally and broadcasted using the *lightweight probabilistic broadcast* (*lpbcastI*) [Eugster03]. This algorithm ensures dissemination of messages to all connected nodes. To deal with offline operations, an anti-entropy algorithm was used. This means that sites periodically send missing updates to randomly selected neighbors. Sometimes, if a site stays offline too long, anti-entropy recovery might not be possible. To solve this situation, wooki sites can update themselves from their neighbors in a state-transfer manner.

DistriWiki

As said before, *Wikis* are growing in importance over the Internet. They are one of the best solutions for cooperative publishing [Morris07]. However, the read-only approach of Web sites that dominates the Internet lead to the establishment of Wikis as centralized services, which does not articulate well with their distributed nature.

In [Morris07], the authors propose that Wikis, in order to comply with their distributed usage paradigm, should be implemented over a peer-to-peer infrastructure. The presented prototype, a peer-to-peer wiki called *DistriWiki* provides the same set of functionalities as classic wiki systems but without the reliability problems of centralized solutions.

The work of DistriWiki identified five important features of peer-to-peer wikis: *redundant decentralization*; *unique identification of documents*; *efficient retrieval of documents*; *usability*; *compatibility with wiki concepts*. This means that P2P wikis should distribute data across peers, which should be able to freely change the content of the distributed data. Also, users should still be able to search and retrieve documents independently from the failure of any peer. Finally, the tools should have a similar user interface and similar concepts and semantics as current wiki systems.

One problem with the completely decentralized solution is how to handle conflicts motivated by concurrent changes. Unfortunately, automatic conflict handling was not covered by DistriWiki, which simply leaves for users to handle such conflicts.

UniWiki

Current peer-to-peer systems are mostly used for distributing quasi-immutable content, thus providing high data availability and good performance. On the other hand there are little solutions for ensuring both *scalability and consistency* in the case of highly dynamic content. The architecture presented in [Oster09] is suitable for systems that need to *store huge amounts of data*, make use of *P2P networks*, does not have *any single point of failure* and can handle *concurrent updates* ensuring *eventual consistency*.

This architecture was used to create a peer-to-peer wiki called *UniWiki* (Universal Wiki), which relies on *distributed hash tables* (DHT) and *optimistic replication* [Saito05]. The normal structure of DHTs, which were created for storing actual data, was slightly modified to *store operations* in each node so it could fit the imposed consistency requirement. To ensure eventual convergence of data, like in *Wooky* [Weiss07], the *Woot* [Oster06] algorithm was used. In order to avoid having every node storing the entire set of wiki pages, each of them is responsible for only a well defined chunk of the content. The defined limits of these chunks can change dynamically in case of arrival or departure of nodes. Another important architectural detail is the separation of the DHT storage system from the wiki clients. To allow

this separation, *wiki front-ends* handle client requests and either transform them in Woot operations (update request) or retrieve the wiki content rendered into HTML (page view request). This allows the system to be *transparently integrated* with other existing wiki engines (e.g. *MediaWiki)*[6]. To support the integration, the system intercepts wiki calls using *aspect oriented programming* so that the distributed storage system is used. The extensive description of the interceptors can be seen in [Oster09].

3.2.3 Intra and Inter-document Relationship

The provide *locality-awareness* in *Cooperative Work Tools*, whether in a *document editor*, *wiki*, *spreadsheet editor*, *presentations builder*, etc., it is of the most importance to know the semantic organization of the data under edition. For example, if one user is editing a chapter, he will want to know if another user starts editing that same chapter. One will be even more interested in knowing if another user starts editing the same section one is editing and even more if it is the same paragraph.

However, this concept of distance between users working concurrently based on the semantic structure of the data being edited must be enhanced if we consider some types of files. For example, it is normal in a HTML or Wiki page to have links to regions inside the same or other pages. Also, in spreadsheets formulas almost always refer to cells placed in the same or other sheets or even in another file.

This intra and inter-document references can shorten the natural semantic distance between regions and systems that aim to provide *locality-awareness* must consider them depending or their context.

There is already some work in this field. In the wiki environment, there are several systems, called *semantic wikis* [Buffa06, Völkel06, Kuhn08, Rahhal08], which aim to subtract from wiki pages relationships between data within pages and among pages. The objectives of semantic wikis are however different since they aim for the exportation of the wiki-based knowledge to databases, ontologies, etc. Another example of reference management is the treatment of *dangling references in the web* [Kappe95, Lawrence01, Veiga03], in which links are considered to ensure referential integrity in the web.

4 Architecture

This section starts with an explanation of how some of the core concepts of the *Vector Field Consistency* algorithm, namely *pivot*, *distance* and *consistency zone*, will be adapted to the environment of *document-based cooperative work*. Then, section 4.2, by means of an *architectural view*, will describe the organization of user nodes, their responsibilities and composition. Finally, in section 4.3 we present two issues being considered to integrate the solution.

4.1 Adapting VFC Concepts: pivots, distance and consistency zones

As said before, in VFC, pivots roughly correspond to each user's observation points. In VFC4CW, a *user's pivot* corresponds to *the place in the document semantics where the user is editing*. Depending on the defined granularity, a user's pivot could be defined by the page,

[6] http://www.mediawiki.org/

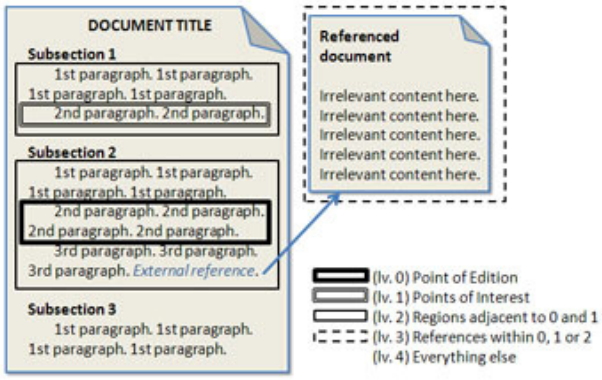

Fig. 3: Consistency zones in the VFC4CW model

section and paragraph which the user is currently editing. To extend this concept to best fit the writing process characteristics, other *points of interest* can be defined. In this section, they will both be referred to only as pivots.

Each user has its own set of pivots which should influence consistency among other users. More specifically, consistency should weaken as the *distance* between users' observation points grows. In VFC4CW, this distance is defined as *the distance between the semantic regions being edited and other points of interest*. Distance should represent how important a certain block of content is to the work being done.

To enforce different consistency levels, around each user's location we define several *consistency zones*. A consistency zone corresponds to a region of a document with a well defined distance from a user location. This distance is zero for the region immediately containing a pivot and increases as we move away from that location. In Fig. 3 we can see several consistency zones around pivots with weaker consistency bounds (represented by thinner lines) as we move far from those locations.

There are numerous regions marked as consistency zones in Fig. 3. In this example we choose to mark the most relevant regions that surround a pivot, but even those are just examples. Relevant regions may change depending on the type of content being edited.

4.2 Architectural components

Every node (application) in a VFC4CW system is equal in terms of implementation, operating broadly as peers. However, some of them will assume special roles in the system as nodes churn and as points of edition change. In Fig. 4, it is presented a possible configuration of system nodes in terms of their role. As we can see, there are two *servers*, two *replicas* and six *clients*. How these nodes with different roles interact and what is their purpose is the subject under discussion in this section.

Clients: As we can conclude from the example of Fig. 4, every node in the system is a client. Clients correspond to the edition applications run by users. They can be explicitly adapted applications or just applications deployed on top of a generic middleware VFC4CW layer. As far as is concerned by the VFC4CW system clients asynchronously edit their documents.

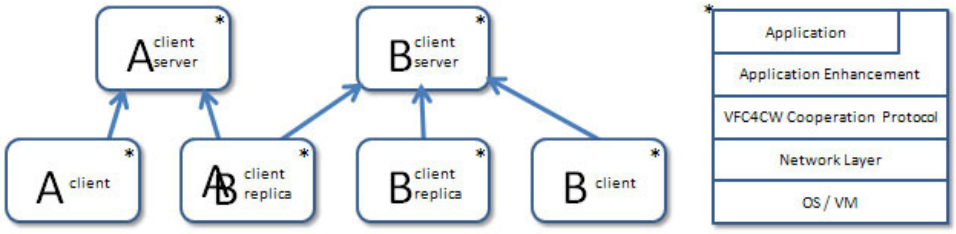

Fig. 4: Main components and main architecture

Servers: Each document under edition has an assigned server which is responsible for monitoring the edition activity and propagating the necessary updates to enforce different consistency levels imposed by the defined consistency zones. It may also be in its duties to warn clients when another client gets close to their pivots.

Such role is assigned dynamically to one of the clients holding and editing the document. Although it is not rigid, the server role may have a more permanent nature, i.e. for the whole life of the document or as long as the document is being edited. Alternatively, this role may be hand over to other nodes.

Replicas: Replicas are nominated by document servers and their only propose is to make the system tolerant to server faults. They are not active replicas in the sense that they only receive and store updates forwarded by servers. If the server for a document fails, one of their replicas is appointed as the new server.

Nodes in the VFC4CW system can assume any role (client, server and replica) and even more than one at the same time, regarding different documents. On the other hand, they must be transparent to the basic user, which is only interested in the cooperative development of his work. Thus, the architecture must incorporate the server and replica capabilities without force the user to interact with any application other then the typical one. To meet these requirements we propose the layered architecture seen in Fig. 4.

Application Enhancement Layer: This layer, which matches the application semantics, translates the user operations into the corresponding updates comprehensible by the VFC4CW consistency protocol. More, it will be responsible for extracting the information about the user location, either by extending the edition tool (if possible) or by analyzing the modifications in the files using a diff algorithm. Also, it may provide extra functionalities to warn the user about concurrent edits.

VFC4CW Cooperation Protocol Layer: This generic layer will implement the VFC4CW protocol for the various roles. It will manage all the updates, which means that it will have the ability to delay or hurry their propagation. This layer should be highly portable whichever the working environment.

4.3 Other architectural details

As studied in the related work section there are two types of definition of operations. Although the original VFC algorithm was implemented as a state-transfer system, it can be perfectly adapted to support the operation-transfer approach. Nevertheless it should be advantageous if there are cases of long disconnection periods to support mixed approaches.

To avoid problems with conflict handling and lost updates, which can be a burden to users, it may be interesting to incorporate semantic-scheduling approaches, namely operational transformation and operation commutativity.

5 Evaluation

As said before this work will result in the integration of the VFC4CW algorithm in a cooperative work tool. To determine if the new algorithm and this integration is in fact a success as we expect, we will perform three forms of evaluation: *qualitative*, *quantitative* and *comparative*.

The *qualitative evaluation* will assess if VFC4CW is transparent to users and what are the benefits of the integration. The VFC4CW model should:

- Provide cooperation capabilities to the usual tools while maintaining the properties and interaction patterns that users expect;
- Fit the model of cooperation without disturbing the user activities;
- Be sufficiently generic to be applied to different cooperative work scenarios and tools;

To perform the *quantitative evaluation* we will analyze a series of criteria to evaluate:

- Bandwidth and memory requirements
    - Number of exchanged messages;
    - Savings in bandwidth and memory;
    - Selection of updates.
- Performance of the algorithm
    - Time to propagate updates;
    - Time to replicate an entire document;
    - Time to obtain a document;
    - Delay imposed to the application use.
- Effectiveness of the algorithm
    - Number of lost updates;
    - Number of detected and removed conflicts.
- Reliability in presence of node failure
    - How many and which nodes can fail at the same time (before the system stabilizes);

Additionally the evaluation process allows making conclusions about whether the model and implemented algorithms are suitable to other environments, like mobile scenarios where there are a series of resource limitations (memory, bandwidth) or web scenarios where there can be a greater number of users simultaneously editing a document.

The *comparative evaluation* will consist in a comparison of some of the obtained quantitative and qualitative results with analogous results from other tools with similar purposes.

6 Conclusions

In this paper we have presented VFC4CW, a *continuous consistency model* which results from the adaptation of the VFC model to the scenario of cooperative work. The VFC model, other than providing a continuum between strong and weak consistency, is combined with the notion of *locality awareness*.

First, we overviewed the state of the art of two important fields: *consistency maintenance in distributed systems* and *cooperative work tools*. Then, we proposed adaptations to best fit the VFC consistency model to the new scenario of cooperative work, namely document based-cooperative work. This adaptation implies porting the original concepts of user's *location* (pivots), *distance* between pivots and *consistency zone* to a new reality. To implement this novel model, we have presented an architecture based on a generic middleware layer, to enforce the VFC4CW protocol, and on an application enhancement layer, adapted to the applications' specifics. A number of criteria were presented for a future analysis of an implementation of this work.

In the future, we will integrate the VFC4CW model with an already established cooperative work tool and later evaluated it.

References

[Applet99] APPELT, W. 1999. WWW Based Collaboration with the BSCW System. In Proceedings of the 26th Conference on Current Trends in theory and Practice of informatics on theory and Practice of informatics

[Bannon91] BANNON, L. J. and SCHMIDT, K. 1991. CSCW: four characters in search of a context. In Studies in Computer Supported Cooperative Work: theory, Practice and Design, J. M. Bowers and S. D. Benford, Eds. North-Holland Human Factors In Information Technology Series, vol. 8. North-Holland Publishing Co., Amsterdam, The Netherlands, 3-16.

[Barreto05] BARRETO, J. and FERREIRA, P. 2005. A highly available replicated file system for resource-constrained windows ce .net devices. In 3rd International Conference on .NET Technologies.

[Barreto09] ARRETO, J., GARCIA, J., VEIGA, L., and FERREIRA, P. 2009. Data-aware connectivity in mobile replicated systems. In Proceedings of the Eighth ACM international Workshop on Data Engineering For Wireless and Mobile Access.

[Bento06] BENTO, M. and PREGUIÇA, N. 2006. Operational transformation based reconciliation in the FEW File System. In Proceedings of the The Eighth International Workshop on Collaborative Editing Systems - integrated in ACM CSCW 2006

[Bernstein86] BERNSTEIN, P. A., HADZILACOS, V., and GOODMAN, N. 1986 Concurrency Control and Recovery in Database Systems. Addison-Wesley Longman Publishing Co., Inc.

[Buffa06] BUFFA, M. and GANDON, F. 2006. SweetWiki: semantic web enabled technologies in Wiki. In Proceedings of the 2006 international Symposium on Wikis.

[Cart07] CART, M. and FERRIE, J. 2007. Asynchronous reconciliation based on operational transformation for P2P collaborative environments. In Proceedings of the 2007 international Conference on Collaborative Computing: Networking, Applications and Worksharing.

[Çetintemel03] ÇETINTEMEL, U., KELEHER, P. J., BHATTACHARJEE, B., and FRANKLIN, M. J. 2003. Deno: A Decentralized, Peer-to-Peer Object-Replication System for Weakly Connected Environments. IEEE Trans. Comput. 52, 7 (Jul. 2003), 943-959.

[Dietterich94] DIETTERICH, D. J. 1994. DEC data distributor: for data replication and data warehousing. SIGMOD Rec. 23, 2 (Jun. 1994), 468.

[Eastlake01] EASTLAKE, D. and JONES, P. 2001 US Secure Hash Algorithm 1 (Sha1).

[Ellis89] ELLIS, C. A. and GIBBS, S. J. 1989. Concurrency control in groupware systems. SIGMOD Rec. 18, 2 (Jun. 1989), 399-407.

[Eugster03] EUGSTER, P. T., GUERRAOUI, R., HANDURUKANDE, S. B., KOUZNETSOV, P., and KERMARREC, A. 2003. Lightweight probabilistic broadcast. ACM Trans. Comput. Syst.

[Greif88] GREIF, I. 1988. Computer-Supported Cooperative Work: a Book of Readings. Morgan Kaufmann Publishers Inc. 1988.

[Grudin88] GRUDIN, J. 1988. Why CSCW applications fail: problems in the design and evaluation of organizational interfaces. In Proceedings of the 1988 ACM Conference on Computer-Supported Cooperative Work.

[Grudin94] GRUDIN, J. 1994. Groupware and social dynamics: eight challenges for developers. Commun. ACM 37, 1 (Jan. 1994), 92-105.

[Ignat03] IGNAT, C. and NORRIE, M. C. 2003. Customizable collaborative editor relying on treeOPT algorithm. In Proceedings of the Eighth Conference on European Conference on Computer Supported Cooperative Work.

[Kappe95] KAPPE, F. 1995. A Scalable Architecture for Maintaining Referential Integrity in Distributed Information Systems. J. UCS, 1(2): 84-104, Feb. 1995.

[Krishnakumar94] KRISHNAKUMAR, N. and BERNSTEIN, A. J. 1994. Bounded ignorance: a technique for increasing concurrency in a replicated system. ACM Trans. Database Syst. 19, 4 (Dec. 1994), 586-625.

[Kuhn08] KUHN, T. 2008. AceWiki: A Natural and Expressive Semantic Wiki. Proc. of Semantic Web User Interaction at CHI 2008: Exploring HCI Challenges, CEUR Workshop Proceedings, 2008

[Lamport78] LAMPORT, L. 1978. Time, clocks, and the ordering of events in a distributed system. Commun. ACM 21, 7 (Jul. 1978), 558-565.

[Lawrence01] LAWRENCE, S., PENNOCK, D. M., FLAKE, G. W., KROVETZ, R., COETZEE, F. M., GLOVER, E., NIELSEN, F. Å., KRUGER, A., and GILES, C. L. 2001. Persistence of Web References in Scientific Research. Computer 34, 2 (Feb. 2001), 26-31.

[Morris07] MORRIS, J. C. 2007. DistriWiki:: a distributed peer-to-peer wiki network. In Proceedings of the 2007 international Symposium on Wikis.

[Muthitacharoen01] MUTHITACHAROEN, A., CHEN, B., and MAZIÈRES, D. 2001. A low-bandwidth network file system. In Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles.

[Nichols95] NICHOLS, D. A., CURTIS, P., DIXON, M., and LAMPING, J. 1995. High-latency, low-bandwidth windowing in the Jupiter collaboration system. In Proceedings of the 8th Annual ACM Symposium on User interface and Software Technology.

[Noël04] NOËL, S. and ROBERT, J. 2004. Empirical Study on Collaborative Writing: What Do Co-authors Do, Use, and Like?. In Comput. Supported Coop. Work.

[Oster09] OSTER, G., MOLLI, P., DUMITRIU, S., and MONDEJAR, R. 2009. UniWiki: A Collaborative P2P System for Distributed Wiki Applications. In Proceedings of the 2009 18th IEEE international Workshops on Enabling Technologies: infrastructures For Collaborative Enterprises.

[Oster06] OSTER, G., URSO, P., MOLLI, P., and IMINE, A. 2006. Data consistency for P2P collaborative editing. In Proceedings of the 2006 20th Anniversary Conference on Computer Supported Cooperative Work.

[Preguiça09] PREGUIÇA, N., MARQUES, J. M., SHAPIRO, M., and LETIA, M. 2009. A Commutative Replicated Data Type for Cooperative Editing. In Proceedings of the 2009 29th IEEE international Conference on Distributed Computing Systems.

[Rahhal08] RAHHAL, C., SKAF-MOLLI, H. and MOLLI, P. 2008. Swooki: A peer-to-peer semantic wiki. In The 3rd Workshop: 'The Wiki Way of Semantics'-SemWiki, co-located with the 5th Annual European Semantic Web Conference (ESWC), Tenerife, Spain. (2008)

[Rama06] RAMA, J. and BISHOP, J. 2006. A survey and comparison of CSCW groupware applications. In Proceedings of the 2006 Annual Research Conference of the South African institute of Computer Scientists and information Technologists on IT Research in Developing Countries.

[Raynal96] RAYNAL, M. and SINGHAL, M. 1996. Logical Time: Capturing Causality in Distributed Systems. Computer 29, 2 (Feb. 1996), 49-56.

[Rodden91] RODDEN, T. 1991. A survey of CSCW systems. Interacting with Computers 3, 3 (1991) 319-353.

[Roh06] ROH, H.-G., KIM, J.-S. and LEE, J. 2006. How to design optimistic operations for peer-to-peer replication. In Int. Conf. on Computer Sc. and Informatics (JCIS/CSI).

[Saito05] SAITO, Y. and SHAPIRO, M. 2005. Optimistic replication. ACM Comput. Surv. 37, 1 (Mar. 2005), 42-81.

[Santos07] SANTOS, N. VEIGA, L., and FERREIRA, P. 2007. Vector-field consistency for ad-hoc gaming. In Proceedings of the ACM/IFIP/USENIX 2007 international Conference on Middleware.

[Satyanarayanan90] SATYANARAYANAN, M., KISTLER, J. J., KUMAR, P., OKASAKI, M. E., SIEGEL, E. H., and STEERE, D. C. 1990. Coda: A Highly Available File System for a Distributed Workstation Environment. *IEEE Trans. Comput.* 39, 4 (Apr. 1990), 447-459.

[Shapiro07] SHAPIRO, M. and PREGUIÇA, N. 2007. Designing a commutative replicated data type. In Institut National de la Recherche en Informatique et Automatique, Rocquencourt, France, Rapport de recherche RR-6320, Oct. 2007.

[Sun98] SUN, C. and ELLIS, C. 1998. Operational transformation in real-time group editors: issues, algorithms, and achievements. In Proceedings of the 1998 ACM Conference on Computer Supported Cooperative Work.

[Sun96] SUN, C., YANG, Y., ZHANG, Y. and CHEN, D. 1996. A consistency model and supporting schemes for real-time cooperative editing systems. In Proceedings of the 19th Australasian Computer Science Conference.

[Sun04] SUN, D., XIA, S., SUN, C., and CHEN, D. 2004. Operational transformation for collaborative word processing. In Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work.

[Tammaro97] TAMMARO, S. G., MOSIER, J. N., GOODWIN, N. C., and SPITZ, G. 1997. Collaborative Writing Is Hard to Support: A Field Study of Collaborative Writing. In Comput. Supported Coop. Work.

[Terry95] TERRY, D. B., THEIMER, M. M., PETERSEN, K., DEMERS, A. J., SPREITZER, M. J., and HAUSER, C. H. 1995. Managing update conflicts in Bayou, a weakly connected replicated storage system. In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*.

[Veiga03] VEIGA, L. and FERREIRA, P. 2003. RepWeb: replicated Web with referential integrity. In Proceedings of the 2003 ACM Symposium on Applied Computing.

[Veiga05] VEIGA, L. and FERREIRA, P. 2005. Semantic-Chunks a middleware for ubiquitous cooperative work. In Proceedings of the 4th Workshop on Reflective and Adaptive Middleware Systems.

[Völkel06] VÖLKEL, M., KRÖTZSCH, M., VRANDECIC, D., HALLER, H., and STUDER, R. 2006. Semantic Wikipedia. In Proceedings of the 15th international Conference on World Wide Web.

[Weiss07] WEISS, S., URSO, P. and MOLLI, P. 2007. Wooki: A p2p wiki-based collaborative writing tool. In Web Information Systems Engineering, Nancy, France, December 2007. Springer.

[Weiss08] WEISS, S., URSO, P. and MOLLI, P. 2008. An undo framework for p2p collaborative editing. In CollaborateCom, Orlando, USA, November 2008.

[Weiss09] WEISS, S., URSO, P., and MOLLI, P. 2009. Logoot: A Scalable Optimistic Replication Algorithm for Collaborative Editing on P2P Networks. In Proceedings of the 2009 29th IEEE international Conference on Distributed Computing Systems.

[Xia04] XIA, S., SUN, D., SUN, C., CHEN, D., and SHEN, H. 2004. Leveraging single-user applications for multi-user collaboration: the coword approach. In Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work.

[Yu00] YU, H. and VAHDAT, A. 2000. Design and evaluation of a continuous consistency model for replicated services. In Proceedings of the 4th Conference on Symposium on Operating System Design & Implementation - Volume 4.

[Yu06] YU, H. and VAHDAT, A. 2006. The costs and limits of availability for replicated services. ACM Trans. Comput. Syst. 24, 1 (Feb. 2006), 70-113.