Synthèse

Nous récapitulons les principales propriétés et différences des modèles de synchronisations pour CRDTs dans Tableau 1.1.

TABLE 1.1 – Récapitulatif comparatif des différents modèles de synchronisation pour CRDTs

	Synchro. par états	Synchro. par opérations	Synchro. par diff. d'états
États forment sup-demi-treillis	✓	✓	✓
Intègre modifications par fusion d'états	✓	✗	✓
Intègre modifications de manière atomique	✗	✓	✓
Résistant aux défaillances réseau	✓	✗	✓
Peut s'affranchir de la cohérence causale	✗	✓	✓
Adapté pour systèmes temps réel	✗	✓	✓

1.2.3 Adoption dans la littérature et l'industrie

- Proposition et conception de CRDTs pour une variété de types de données : Registre, Compteur *Matthieu: TODO : Ajouter IPA*, Ensemble, Liste/Sequence, Graphe, JSON, Filesystem, Access Control. Propose généralement plusieurs sémantiques de résolution de conflits par type de données.
- Conception et développement de librairies mettant à disposition des développeurs d'applications des types de données composés [37, 38, 39, 40, 41] *Matthieu: TODO : Revoir et ajouter Melda (PaPoC'22) si fitting*
- Conception de langages de programmation intégrant des CRDTs comme types primitifs, destinés au développement d'applications distribuées [42, 43]
- Conception et implémentation de bases de données distribuées, relationnelles ou non, privilégiant la disponibilité et la minimisation de la latence à l'aide des CRDTs [44, 45, 46, 47, 48] *Matthieu: TODO : Ajouter Redis et Akka*
- Conception d'un nouveau paradigme d'applications, Local-First Software, dont une des fondations est les CRDTs [49, 50] *Matthieu: TODO : Vérifier et ajouter l'article avec Digital Garden (PaPoC'22 ?) si fitting*
- Éditeurs collaboratifs temps réel à large échelle et offrant de nouveaux scénarios de collaboration grâce aux CRDTs [51, 3]

1.3 Séquences répliquées sans conflits

La *Séquence*, aussi appelée *Liste*, est un type abstrait de données représentant une collection ordonnée et de taille dynamique d'éléments. Dans une séquence, un même élément peut apparaître à de multiples reprises. Chacune des occurrences de cet élément est alors considérée comme distincte.

Dans le cadre de ce manuscrit, nous travaillons sur des séquences de caractères. Cette restriction du domaine se fait sans perte en généralité. Nous illustrons par la Figure 1.10 notre représentation des séquences.

H	E	L	L	O
0	1	2	3	4

FIGURE 1.10 – Représentation de la séquence "HELLO"

Dans la Figure 1.11, nous présentons la spécification algébrique du type Séquence que nous utilisons.

$$
\begin{array}{lcll}
S \in Seq\langle V \rangle & & & \\
 & & & \\
emp & : & & \longrightarrow S \\
 & & & \\
ins & : & S \times N \times V & \longrightarrow S \\
rmv & : & S \times N & \longrightarrow S \\
 & & & \\
len & : & S & \longrightarrow N \\
rd & : & S & \longrightarrow Arr\langle V \rangle
\end{array}
$$

FIGURE 1.11 – Spécification algébrique du type abstrait usuel Séquence

Celle-ci définit deux modifications :

— *ins*, qui permet d'insérer un élément donné v à un index donné i dans une séquence s de taille m. Cette modification renvoie une nouvelle séquence construite de la manière suivante :

$$\forall s \in S, v \in V, i \in [0, m] \mid m = len(s) - 1, s = \langle v_0, ..., v_{i-1}, v_i, ..., v_m \rangle \cdot ins(s, i, v) = \langle v_0, ..., v_{i-1}, v, v_i, ..., v_m \rangle$$

— *rmv*, qui permet de retirer l'élément situé à l'index i dans une séquence s de taille m. Cette modification renvoie une nouvelle séquence construite de la manière suivante :

$$\forall s \in S, v \in V, i \in [0, m] \mid m = len(s) - 1, s = \langle v_0, ..., v_{i-1}, v_i, vi + 1, ..., v_m \rangle \cdot rmv(s, i) = \langle v_0, ..., v_{i-1}, v_{i+1}, ..., v_m \rangle$$

Les modifications définies dans Figure 1.11, *ins* et *rmv*, ne permettent respectivement que l'insertion ou la suppression d'un élément à la fois. Cette simplification du type se fait cependant sans perte de généralité, la spécification pouvant être étendue pour insérer successivement plusieurs éléments à partir d'un index donné ou retirer plusieurs éléments consécutifs.

Cette spécification du type Séquence est une spécification séquentielle. Les modifications sont définies pour être effectuées l'une après l'autre. Si plusieurs noeuds répliquent une même séquence et la modifient en concurrence, l'intégration de leurs opérations respectives dans des ordres différents résulte en des états différents. Nous illustrons ce point avec la Figure 1.12.

Dans cet exemple, deux noeuds A et B partagent et éditent collaborativement une même séquence. Celle-ci correspond initialement à la chaîne de caractères "WRD". Le

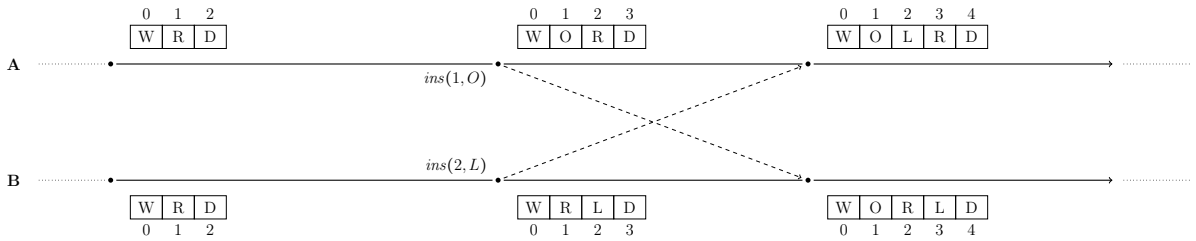

FIGURE 1.12 – Modifications concurrentes d'une séquence

noeud A insère le caractère "O" à l'index 1, obtenant ainsi la séquence "WORD". En concurrence, le noeud B insère lui le caractère "L" à l'index 2 pour obtenir "WRLD".

Les deux noeuds diffusent ensuite leur opération respective puis intègre celle de leur pair. Nous constatons alors une divergence. En effet, l'intégration de la modification $ins(2, L)$ par le noeud A ne produit pas l'effet escompté, c.-à-d. produire la chaîne "WORLD", mais la chaîne "WOLRD.

Cette divergence est dûe à la non-commutativité de la modification *ins* avec elle-même. En effet, celle-ci se base sur un index pour déterminer où placer le nouvel élément. Cependant, les index sont eux-mêmes modifiés par *ins*. Ainsi, l'intégration dans des ordres différents de modifications *ins* sur un même état initial résulte en des états différents. Plus généralement, nous observons que chaque paire possible de modifications du type Séquence, c.-à-d. $\langle ins, ins \rangle$, $\langle ins, del \rangle$ et $\langle del, del \rangle$, n'est pas commutative.

La non-commutativité des modifications du type Séquence fut l'objet de nombreux travaux de recherche dans le domaine de l'édition collaborative. Pour résoudre ce problème, l'approche Operational Transformation (OT) [52, 53] fut initialement proposée. Cette approche propose de transformer une modification par rapport aux modifications concurrentes intégrées pour tenir compte de leur effet. Elle se décompose en deux parties :

— Un algorithme de contrôle [54, 55, 56], qui définit par rapport à quelles modifications une nouvelle modification distante doit être transformée avant d'être intégrée à la copie.

— Des fonctions de transformations *Matthieu: TODO : Ajouter refs*, qui définissent comment une modification doit être transformée par rapport à une autre modification pour tenir compte de son effet.

Cependant, bien que de nombreuses fonctions de transformations pour le type Séquence ont été proposées, seule la correction des Tombstone Transformation Functions (TTF) [57] a été éprouvée à notre connaissance. *Matthieu: TODO : Vérifier que c'est pas plutôt les seules fonctions de transformations qui sont correctes et compatibles avec un système P2P.* De plus, les algorithmes de contrôle compatibles reposent sur une livraison causale des modifications, et donc l'utilisation de vecteurs d'horloges. Cette approche est donc inadaptée aux systèmes Pair-à-Pair (P2P) dynamiques.

Néanmoins, une contribution importante de l'approche OT fut la définition d'un modèle de cohérence que doivent respecter les systèmes d'édition collaboratif : le modèle Convergence, Causality preservation, Intention preservation (CCI) [58].

Définition 7 (Modèle CCI) *Le modèle de cohérence CCI définit qu'un système d'édition collaboratif doit respecter les critères suivants :*

Définition 7.1 (Convergence) *Le critère de* Convergence *indique que des noeuds ayant intégrés le même ensemble de modifications convergent à un état équivalent.*

Définition 7.2 (Causality preservation) *Le critère de* Causality preservation *indique que si une modification m_1 précède une autre modification m_2 d'après la relation* happens-before*, c.-à-d. $m_1 \rightarrow m_2$, m_1 doit être intégrée avant m_2 par les noeuds du système.*

Définition 7.3 (Intention preservation) *Le critère de* Intention preservation *indique que l'intégration d'une modification par un noeud distant doit reproduire l'effet de la modification sur la copie du noeud d'origine, indépendamment des modifications concurrentes intégrées.*

De manière similaire à [59], nous ajoutons le critère de *Scalability* à ces critères :

Définition 8 (Scalability) *Le critère de* Scalability *indique que le nombre de noeuds du système ne doit avoir qu'un impact sous-linéaire sur les complexités en temps, en espace et sur le nombre et la taille des messages.*

Nous constatons cependant que les critères 7.2 et 8 peuvent être contraires. En effet, pour respecter le modèle de cohérence causale, un système peut nécessiter une livraison causale des modifications, e.g. un CRDT synchronisé par opérations dont seules les opérations concurrentes sont commutatives. La livraison causale implique un surcoût computationnel, en métadonnées et en taille des messages qui est fonction du nombre de participants du système [60]. Ainsi, dans le cadre de nos travaux, nous cherchons à nous affranchir du modèle de livraison causale des modifications, ce qui peut nécessiter de relaxer le modèle de cohérence causale.

C'est dans une optique similaire que fut proposé [61], un modèle de séquence répliquée qui pose les fondations des CRDTs. Depuis, plusieurs CRDTs pour Séquence furent définies [62, 63, 59]. Ces CRDTs peuvent être répartis en deux approches : l'approche à pierres tombales [61, 62] et l'approche à identifiants densément ordonnés [63, 59]. L'état d'une séquence pouvant croître de manière infinie, ces CRDTs sont synchronisés par opérations pour limiter la taille des messages diffusés. À notre connaissance, seul [5] propose un CRDT pour Séquence synchronisé par différence d'états.

Dans la suite de cette section, nous présentons les différents CRDTs pour Séquence de la littérature.

1.3.1 Approche à pierres tombales

WOOT

— WOOT [61] fait suite aux travaux présentés dans [57]. Il est considéré a posteriori comme le premier CRDT synchronisé par opérations pour Séquence [14].

14. [64] n'ayant formalisé les CRDTs qu'en 2007.

— Conçu pour l'édition collaborative P2P, son but est de surmonter les limites de l'approche OT. En effet, l'approche OT souffre de la nécessité d'associer à chaque opération des informations de causalité, c.-à-d. un vecteur d'horloges. La taille de cette structure de données augmentant avec le nombre de noeuds du système, cette approche se révèle inadaptée aux systèmes P2P dynamique, c.-à-d. systèmes dans lesquels un grand nombre de noeuds peuvent rejoindre puis quitter la collaboration.

— L'intuition de WOOT est la suivante : WOOT modifie la sémantique de la modification *ins* pour qu'elle corresponde à l'insertion d'un nouvel élément entre deux autres, et non plus à l'insertion d'un nouvel élément à une position donnée. Ce changement, qui est compatible avec l'intention des utilisateur-rices, n'est cependant pas anodin. En effet, il permet à WOOT de rendre *ins* commutative avec les modifications concurrentes. en exprimant la position du nouvel élément de manière relative à d'autres éléments et non plus via un index qui est spécifique à un état donné.

— Afin de préciser quels éléments correspondent aux prédécesseur et successeur de l'élément inséré, WOOT repose sur un système d'identifiants. WOOT associe ainsi un identifiant unique à chaque élément de la Séquence. La modification *rmv* utilise aussi les identifiants pour indiquer l'élément à supprimer.

— WOOT utilise des pierres tombales pour rendre *ins*, qui nécessite la présence des deux éléments entre lesquels nous insérons un nouvel élément, commutative avec *rmv*. Ainsi, une pierre tombale est conservée est conservée dans la Séquence pour indiquer sa présence passée, mais les données de l'élément sont supprimées. Dans le cadre d'une Séquence WOOT de caractères, *rmv* a donc pour effet de masquer l'élément.

— Finalement, WOOT définit $<_{id}$, un ordre strict total sur les identifiants associés aux éléments. En effet, il convient de noter que la relation $\prec$ ne spécifie qu'un ordre partiel entre les éléments. Ainsi, $\prec$ ne permet pas d'ordonner les éléments insérés en concurrence et possédant les mêmes prédecesseur et successeur, e.g. $ins(a \prec 1 \prec b)$ et $ins(a \prec 2 \prec b)$. Pour que tous les noeuds convergent, ils doivent choisir comment ordonner ces éléments de manière déterministe et indépendante de l'ordre de réception des modifications. Ils utilisent pour cela $<_{id}$.

— Ainsi WOOT offre une spécification de la Séquence dont les opérations sont commutatives, c.-à-d. ne génèrent pas de conflits. Nous illustrons son fonctionnement à l'aide de la Figure 1.13.

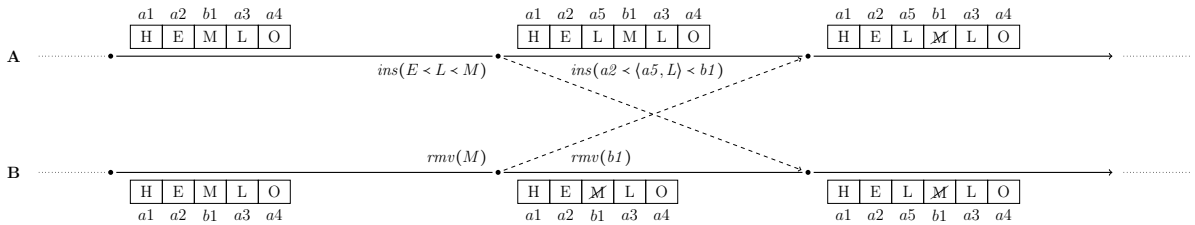

FIGURE 1.13 – Modifications concurrentes d'une séquence répliquée WOOT

Dans cet exemple, deux noeuds A et B partagent et éditent collaborativement une séquence répliquée WOOT. Initialement, ils possèdent le même état : la séquence

contient les éléments "HEMLO", et à chaque élément est associé un identifiant, e.g. $a1$, $b1$, $a2$...

— Le noeud A insère l'élément "L" entre les éléments "E" et "M", c.-à-d. $ins(E \prec L \prec M)$. WOOT convertit cette modification en opération $ins(a2 \prec \langle a5, L\rangle \prec b1)$. L'opération est intégrée à la copie locale, ce qui produit l'état "HELMLO", puis diffusée sur le réseau.

— En concurrence, le noeud B supprime l'élément "M" de la séquence, c.-à-d. $rmv(M)$. De la même manière, WOOT génère l'opération correspondante $rmv(b1)$. Comme expliqué précédemment, l'intégration de cette opération ne supprime pas l'élément "M" de l'état mais se contente de le masquer. L'état produit est donc "HE$\not{M}$LO". L'opération est ensuite diffusée.

— A (resp. B) reçoit ensuite l'opération de B, $rmv(b1)$ (resp. A, $ins(a2 \prec \langle a5, L\rangle \prec b1)$), et l'intègre à sa copie. Les opérations de WOOT étant commutatives, les noeuds obtiennent le même état final : "HEL$\not{M}$LO".

— Grâce à la commutativité de ses opérations, WOOT s'affranchit du modèle de livraison causale nécessitant l'utilisation coûteuse de vecteurs d'horloges. WOOT met en place un modèle de livraison sur-mesure basé sur les pré-conditions des opérations :

(i) L'opération $ins(predId \prec \langle id, elt\rangle \prec succId)$ ne peut être délivrée qu'après la livraison des opérations d'insertion des éléments associés à $predId$ et $succId$.

(ii) L'opération $rmv(id)$ ne peut être délivrée qu'après la livraison de l'opération d'insertion de l'élément associé à id.

Ce modèle de livraison ne requiert qu'une quantité fixe de métadonnées associées à chaque opération pour être respecté. WOOT est donc adapté aux systèmes P2P dynamiques.

— *Matthieu: TODO : Mentionner que WOOT ne propose pas de mécanisme pour purger les tombstones, par choix : souhaite mettre en place des fonctionnalités d'undo.*

— Des améliorations de WOOT furent par la suite proposées : WOOTO [65] et WOOTH [66]. *Matthieu: TODO : Ajouter limite de WOOT : complexité en temps de l'algorithme d'intégration des insertions distantes trop importante* Weiss, Urso et Molli [65] remanient la structure des identifiants associés aux éléments. Cette modification permet un algorithme d'intégration des opérations *ins* plus efficace. Ahmed-Nacer et al. [66] se basent sur WOOTO et proposent l'utilisation de structures de données améliorant la complexité des algorithmes d'intégration des opérations, au détriment des métadonnées stockées localement par chaque noeud.

— Néanmoins, l'évaluation expérimentale des différentes approches pour l'édition collaborative P2P en temps réel menée dans [66] a montré que les CRDTs de la famille WOOT n'étaient pas assez efficaces. Dans le cadre de cette expérience, des utilisateur-rices effectuaient des tâches d'édition collaborative données. Les traces de ces sessions d'édition collaboratives furent ensuite rejouées en utilisant divers mécanismes de résolution de conflits, dont WOOT, WOOTO et WOOTH. Le but était de mesurer les performances de ces mécanismes, notamment leurs temps d'intégration des modifications et opérations. Dans le cas de la famille WOOT, Ahmed-Nacer

et al. ont constaté que ces temps dépassaient parfois 50ms. Il s'agit là de la limite des délais acceptables par les utilisateur-rices d'après [67, 68]. Ces performances disqualifient donc les CRDTs de la famille WOOT comme approches viables pour l'édition collaborative P2P temps réel.

Replicated Growable Array

- Replicated Growable Array (RGA) [62] est le second CRDT appartenant à l'approche à pierres tombales. Spécifié dans le cadre d'un effort pour établir les principes nécessaires à la conception de Replicated Abstract Data Types (RADTs).
- Dans cet article, définissent et se basent sur 2 principes pour concevoir RADTs. Le premier d'entre eux est Operation Commutativity (OC). Celui-ci indique que toutes paires possibles d'opérations concurrentes doit être commutative. Permet de garantir que l'intégration par des noeuds différents d'une même séquence d'opérations concurrentes, mais dans des ordres différents, produira un état équivalent.
- Le second principe sur lequel repose les RADTs est la Precedence Transitivity (PT). Définit une relation de précédence, $\dashrightarrow$, qui permet d'établir un ordre de priorité sur les intentions des opérations, pour déterminer laquelle conserver en cas de conflits. Precedence Transitivity définit que pour tout trio d'opérations a, b et c, si on a $a \dashrightarrow b$ et $b \dashrightarrow c$, on a $a \dashrightarrow c$.
- À partir de ces principes, proposent plusieurs RADTs : Replicated Fixed-Size Array (RFA), Replicated Hash Table (RFT) et Replicated Growable Array (RGA), qui nous intéresse ici.
- Dans RGA, l'intention de l'insertion est définit comme l'insertion d'un nouvel élément directement après un élément existant. Ainsi, RGA se base sur le prédecesseur d'un élément pour déterminer où l'insérer. De fait, tout comme WOOT, RGA repose sur un système d'identifiants qu'il associe aux éléments pour pouvoir s'y référer par la suite. Nommé S4Vector, ces identifiants sont de la forme suivante $\langle ssid, sum, ssn, seq \rangle$ avec :
  - *ssid*, identifiant de la session de collaboration
  - *sum*, somme du vecteur d'horloges courant du noeud auteur de l'élément à son insertion
  - *ssn*, identifiant du noeud auteur de l'élément
  - *seq*, numéro de séquence de l'auteur de l'élément à son insertion

  L'insertion est donc définie de la manière suivante : $ins(predId \prec \langle id, elt \rangle)$. Dans présentations suivantes de RGA, utilisent horloge de Lamport [15] plutôt. Nous abstrayons donc ici la structure des identifiants avec le symbole t.
- Puisque plusieurs éléments peuvent être insérés en concurrence à la même position, c.-à-d. avec le même prédecesseur, il est nécessaire de définir une relation d'ordre strict total pour ordonner les éléments de manière déterministe et indépendante de l'ordre de réception des modifications. Pour cela, utilise l'ordre lexicographique sur les composants des identifiants des éléments. Par exemple, dans le cadre des

S4Vector, on a $\langle 1, 10, A, 2\rangle < \langle 2, 5, B, 1\rangle$ ou $\langle 1, 10, A, 2\rangle < \langle 1, 15, B, 1\rangle$. L'utilisation de cet ordre comme stratégie de résolution de conflits permet de rendre commutative les modifications *ins* concurrentes.

— Concernant les suppressions, RGA se comporte de manière similaire à WOOT : la séquence conserve une tombstone pour chaque élément supprimé, de façon à pouvoir insérer à la bonne position un élément dont le prédecesseur a été supprimé en concurrence. Cette stratégie rend commutative les modifications *ins* et *rmv*.

— Nous récapitulons le fonctionnement de RGA à l'aide de la Figure 1.14. *Matthieu:*

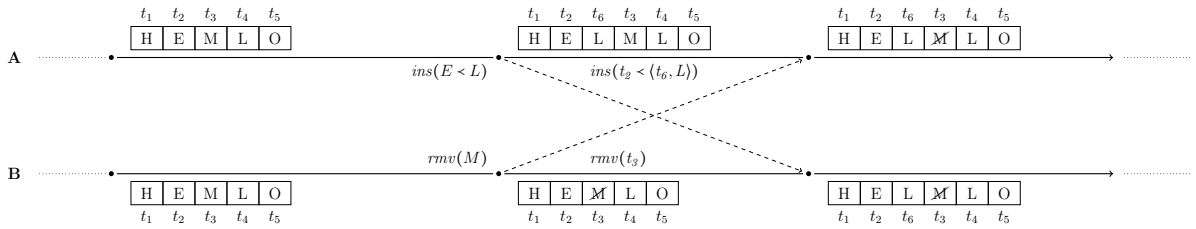

FIGURE 1.14 – Modifications concurrentes d'une séquence répliquée RGA

TODO : Trouver exemple et le présenter.

— À la différence des auteurs de WOOT, ROH et al. [62] jugent le coût des tombstones trop élévé. Ils proposent alors un mécanisme de Garbage Collection (GC) des tombstones. Ce mécanisme repose sur deux conditions :

(i) La stabilité causale de l'opération *rmv*, c.-à-d. l'ensemble des noeuds a observé la suppression de l'élément et ne peut émettre d'opérations utilisant l'élément supprimé comme prédecesseur.

(ii) L'impossibilité pour l'ensemble des noeuds de générer un *S4Vector* plus petit que l'élément suivant la tombstone [15].

— Concernant le modèle de livraison adopté, RGA repose sur une livraison causale des opérations. [62] indique que ce modèle de livraison pourrait être relaxé, de façon à ne plus dépendre de vecteurs d'horloges. Ce point est néanmoins laissé comme piste de recherche future. À notre connaissance, celle-ci n'a pas été explorée dans la littérature. Néanmoins ELVINGER [5] indique que RGA pourrait adopter le même modèle de livraison que WOOT. Pour rappel, celui-ci consiste en :

(i) L'opération $ins(predId \prec \langle id, elt\rangle)$ ne peut être délivrée qu'après la livraison de l'opération d'insertion de l'élément associé à $predId$.

(ii) L'opération $rmv(id)$ ne peut être délivrée qu'après la livraison de l'opération d'insertion de l'élément associé à id.

Nous secondons cette observation.

— *Matthieu: TODO : Ajouter quelques lignes sur efficacité de RGA. Meilleure complexité en temps pour l'intégration des opérations. [69, 70] montrent que RGA est optimal du point de vue complexité spatiale comme CRDT pour Séquence par élément basée sur l'approche tombstone. Implémenté dans Yjs et Automerge.*

15. On constate cependant que la condition (i) implique la condition (ii), puisque tout noeud ayant observé l'insertion de l'élément suivant possédera un vecteur d'horloges supérieur.

— Plusieurs extensions de RGA furent proposées par la suite. Briot, Urso et Shapiro [71] indiquent que les pauvres performances des modifications locales[16] des CRDTs pour Séquence constituent une de leurs limites. Il s'agit en effet des performances impactant le plus l'expérience utilisateur, celleux-ci s'attendant à un feedback immédiat de la part de l'application. Les auteurs souhaitent donc réduire la complexité en temps des modifications locales à une complexité logarithmique. Pour cela, ils proposent l'*identifier structure*, une structure de données auxiliaire utilisable par les CRDTs pour Séquence. Cette structure permet de retrouver plus efficacement l'identifiant d'un élément à partir de son index, au pris d'un surcoût en métadonnées. Briot, Urso et Shapiro [71] combinent cette structure de données à un mécanisme d'aggrégation des élements en blocs[17] tels que proposés par [72, 4], qui permet de réduire la quantité de métadonnées stockées par la séquence répliquée. Cette combinaison aboutit à la définition d'un nouveau CRDT pour Séquence, *RGATreeSplit*, qui offre une meilleure complexité en temps et en espace.

— Kleppmann et al. [73] mettent en lumière un problème des CRDTs pour Séquence. Lorsque des séquences de modifications sont effectuées en concurrence par des noeuds, les CRDTs assurent la convergence des répliques mais pas la correction du résultat. Notamment, il est possible que les éléments insérés en concurrence se retrouvent entrelacés. *Matthieu: TODO : Insérer et présenter exemple* Pour remédier

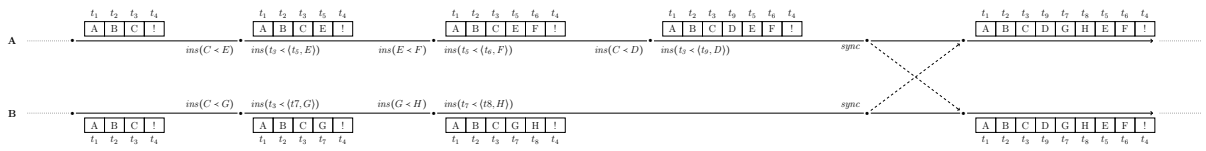

Figure 1.15 – Entrelacement d'éléments insérés de manière concurrente

à ce problème, les auteurs définissent une nouvelle spécification que doivent respecter les approches pour la mise en place de séquences répliquées : *la spécification forte sans entrelacement des séquences répliquées*. Basée sur la spécification forte des séquences répliquées spécifiée dans [69, 70], cette nouvelle spécification ajoute comme propriété que les éléments insérés en concurrence ne doivent pas s'entrelacer dans l'état final. Kleppmann et al. [73] proposent ensuite une évolution de RGA respectant cette spécification. Pour cela, les auteurs ajoutent à l'opération *ins* un paramètre, *samePredId*, un ensemble correspondant à l'ensemble des identifiants connus utilisant le même *predId* que l'élément inséré. En maintenant en plus un exemplaire de cet ensemble pour chaque élément de la séquence, il est possible de déterminer si deux opérations *ins* sont concurrentes ou causalement liées et ainsi déterminer comment ordonner leurs éléments. Cependant, les auteurs ne prouvent pas dans [73] que cette extension empêche tout entrelacement[18].

16. Relativement par rapport aux algorithmes de l'approche OT.
17. Nous détaillerons ce mécanisme par la suite.
18. Un travail en cours [74] indique en effet qu'une séquence répliquée empêchant tout entrelacement est impossible.

1.3.2 Approche à identifiants densément ordonnés

Treedoc

— [64, 63] proposent une nouvelle approche pour CRDTs pour Séquence. Se base sur des identifiants de position, respectant les propriétés suivantes :

(i) Chaque élément se voit attribuer un identifiant.

(ii) Aucune paire d'éléments ne partage le même identifiant.

(iii) L'identifiant d'un élément est immuable.

(iv) Il existe un ordre total strict sur les identifiants, $<_{id}$, cohérent avec l'ordre des éléments dans la séquence.

(v) Les identifiants sont tirés d'un ensemble dense, noté $\mathbb{I}$.

— La propriété (v) signifie que :

$$\forall predId, succId \in \mathbb{I}, \exists id \in \mathbb{I} \cdot predId <_{id} id <_{id} succId$$

Par exemple, les nombres réels forment un ensemble dense. Ceux-ci sont néanmoins inutilisables en informatique puisqu'ils nécessiteraient une précision infinie. [63] utilise donc un type dédié pour les émuler.

— L'utilisation d'identifiants de position permet de redéfinir les modifications :

(i) $ins(pred \prec elt \prec succ)$ devient alors $ins(id, elt)$, avec $predId <_{id} id <_{id} succId$.

(ii) $rmv(elt)$ devient $rmv(id)$.

Ces redéfinitions permettent d'obtenir une spécification de la séquence avec des modifications commutatives.

— À partir de cette spécification, Preguica et al. propose un CRDT pour Séquence : *Treedoc*. Ce dernier se base sur un arbre binaire pour générer les identifiants de position.

— La racine de l'arbre binaire, notée ϵ, correspond à l'identifiant du premier élément inséré dans la séquence répliquée. Puis, pour générer l'identifiant d'un nouvel élément inséré à gauche (resp. à droite) d'un noeud de l'arbre binaire, Treedoc concatène un 0 (resp. un 1) à l'identifiant de ce dernier.

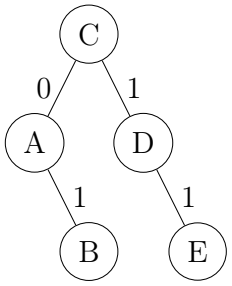

Figure 1.16 – Identifiants de positions

— Ce mécanisme souffre néanmoins d'un écueil : en l'état, plusieurs noeuds du systèmes peuvent générer un même identifiant pour des éléments insérés en concurrence, contravenant alors à la propriété (ii). Pour corriger cela, Treedoc ajoute aux identifiants des désambiguateurs : un *Dot*. Un désambiguateur est ajouté à chaque partie d'un identifiant lorsque nécessaire, c.-à-d.

(i) La partie courante de l'identifiant est la fin de l'identifiant.

(ii) La partie courante de l'identifiant nécessite désambiguation, c.-à-d. plusieurs éléments utilisent ce même identifiant.

Il convient de noter que les noeuds de l'arbre binaire des identifiants peuvent ainsi contenir une liste d'identifiants en cas d'insertions concurrentes.

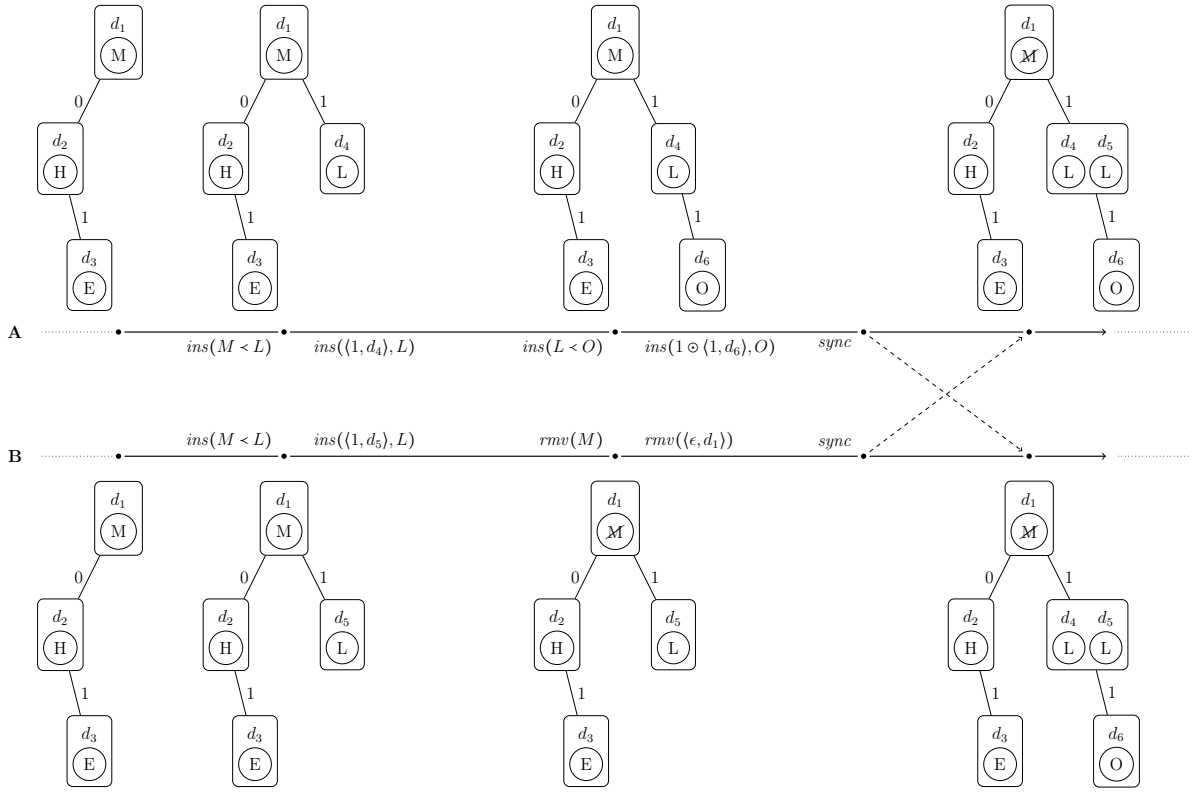

FIGURE 1.17 – Modifications concurrentes d'une séquence répliquée Treedoc

— Concernant le modèle de livraison utilisé, [63] indique reposer sur le modèle de livraison causal. En pratique, nous pouvons néanmoins relaxer le modèle de livraison comme expliqué dans [5] :

(i) Les opérations *ins* peuvent être livrées dans n'importe quel ordre.

(ii) L'opération $rmv(id)$ ne peut délivrée qu'après la livraison de l'opération d'insertion de l'élément associé à id.

— Treedoc souffre néanmoins de plusieurs limites. Tout d'abord, le mécanisme d'identifiants de positions proposé est couplé à la structure d'arbre binaire. Cependant, les utilisateur-rices ont tendance à écrire de manière séquentielle, c.-à-d. dans le sens

d'écriture de la langue utilisée. Les nouveaux identifiants forment donc généralement une liste chaînée, qui déséquilibre l'arbre.

— Ensuite, Treedoc doit conserver un noeud de l'arbre des identifiants malgré sa suppression lorsque ce dernier possède des enfants. Ce noeud de l'arbre devient alors une pierre tombale. Le mécanisme de GC des pierres tombales de Treedoc ne repose cependant pas sur la stabilité causale d'opérations, mais sur l'état du noeud de l'arbre, c.-à-d. si la pierre tombale devient une feuille. Néanmoins, l'évaluation de [63] a montré que les pierres tombales pouvait représenter jusqu'à 95% des noeuds de l'arbre.

— Finalement, Treedoc souffre du problème de l'entrelacement d'éléments insérés de manière concurrente, contrairement à ce qui est conjecturé dans [73].

Logoot

— En parallèle à Treedoc [63], Weiss, Urso et Molli [59] proposent Logoot. Ce nouvel CRDT pour Séquence repose sur idée similaire à celle de Treedoc : il associe un identifiant de position, provenant d'un espace dense, à chaque élément de la séquence.

— Dans [59], les identifiants de positions sont de la forme $\langle tuples, seq \rangle$ avec
  — *seq*, le numéro de séquence du noeud à l'insertion de l'élément
  — *tuples*, une liste de paires $\langle pos, nodeId \rangle$ avec
  — *pos*, un entier
  — *nodeId*, l'identifiant du noeud auteur de l'élément.

— Par la suite, [75] re-spécifie les identifiants de positions de la manière suivante :
  — liste de tuples avec chaque tuple un triplet <pos, site, seq> avec
    — pos un entier
    — site l'identifiant de site
    — seq le numéro de séquence du site à l'ins de l'élément

— Dans le cadre de cette section, nous nous basons sur cette dernière spécification. Nous utiliserons la notation suivante $pos^{nodeId\ \ seq}$ pour représenter chacun des tuples composant un identifiant de position Logoot. Sans perdre en généralité, nous utiliserons des lettres minuscules comme valeurs pour *pos*, des lettres majuscules pour *nodeId* et des entiers pour *seq*. Par exemple, l'identifiant $\langle\langle i, A, 1\rangle\langle f, B, 1\rangle\rangle$ est représenté par $i^{A1} f^{B1}$.

— Logoot définit un ordre strict total $<_{id}$ sur les identifiants de position. Cet ordre lui permet de les ordonner relativement les uns aux autres, et ainsi ordonner les éléments associés. Pour définir $<_{id}$, Logoot se base sur l'ordre lexicographique.

Définition 9 (Relation $<_{id}$) *Étant donné deux identifiants* $id = t_1 \oplus t_2 \oplus ... \oplus t_n$ *et* $id' = t'_1 \oplus t'_2 \oplus ... \oplus t'_m$*, on a :*

$$id <_{id} id' \quad \textit{iff} \quad \begin{cases} n < m \wedge \forall i \in [1, n] \cdot t_i = t'_j & \textit{or} \\ \exists j \leq m \cdot \forall i < j \cdot t_i = t'_i \wedge t_j <_t t'_j \end{cases}$$

avec :

Définition 10 (Relation $<_t$) *Étant donné deux tuples $t = \langle pos, nodeId, seq \rangle$ et $t' = \langle pos', nodeId', seq' \rangle$, on a :*

$$t <_t t' \quad iff \quad \begin{cases} pos < pos' & or \\ pos = pos' \wedge nodeId < nodeId' & or \\ pos = pos' \wedge nodeId = nodeId' \wedge seq < seq' \end{cases}$$

— Logoot spécifie une fonction `generateId`. Cette fonction permet de générer un nouvel identifiant de position, *id*, entre deux identifiants donnés, *predId* et *succId*, tel que $predId \prec id \prec succId$. Plusieurs algorithmes peuvent être utilisés pour cela. Notamment, [59] présente un algorithme permettant de générer N identifiants de manière aléatoire entre les identifiants *predId* et *succId*. Nous présentons ci-dessous un algoritme naïf. *Matthieu: TODO : Insérer algo*

— Pour illustrer cet algorithme, considérons `generateId(predId, nextId, nodeId, seq)` avec $nodeId = C$ et $seq = 1$ et

(i) $predId = e^{A1}$, $nextId = m^{B1}$. Ces deux identifiants n'ayant aucun préfixe commun, `generateId` renvoie un identifiant composé d'un tuple avec une position aléatoire choisie entre leur positions respectives, c.-à-d. dans $]e, m[$, et avec les valeurs *nodeId* et *seq*, e.g. l^{C1}.

(ii) $predId = i^{A1} f^{A2}$, $succId = i^{A1} g^{B1}$. Ici, les deux identifiants ont un préfixe commun, i^{A1}, que `generateId` récupère. `generateId` compare ensuite les tuples suivants de *predId* et *succId* pour déterminer si l'intervalle entre les deux est suffisant pour insérer un nouvel identifiant. Ces tuples étant respectivement f^{A2} et g^{B1}, ce n'est pas le cas ici. `generateId` copie alors le tuple courant de *predId*, récupère le prochain tuple de *predId* et utilise $MAXTUPLE$ à la place du prochain tuple *succId*. Il génère finalement un nouveau tuple avec une valeur aléatoire de position choisie dans $]MINPOS, MAXPOS]$ [19], et avec les valeurs *nodeId* et *seq*, e.g. $i^{A1} f^{A2} m^{C1}$.

— L'utilisation d'identifiants de position permet de redéfinir les modifications :

(i) $ins(pred \prec elt \prec succ)$ devient alors $ins(id, elt)$, avec $predId <_{id} id <_{id} succId$.

(ii) $rmv(elt)$ devient $rmv(id)$.

— Nous illustrons à présent son fonctionnement à l'aide de la Figure 1.18.

— Concernant le modèle de livraison de Logoot, [59] indique se reposer sur le modèle de livraison causal. Nous constatons cependant que :

— Les opérations *ins* sont commutatives, elles peuvent donc être délivrées dans le désordre.

19. Il est important d'exclure *MINPOS* des valeurs possibles pour *pos* du dernier tuple d'un identifiant *id* afin de garantir que l'espace reste dense, notamment pour garantir qu'un noeud sera toujours en mesure de générer un nouvel identifiant id' tel que $id' <_{id} id$.

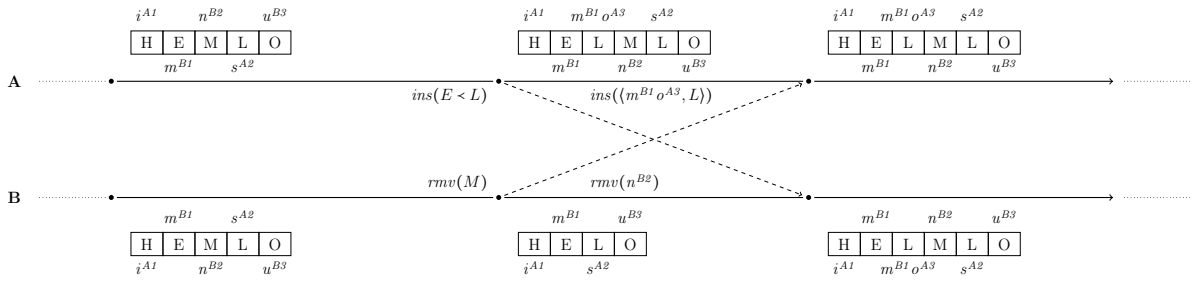

FIGURE 1.18 – Modifications concurrentes d'une séquence répliquée Logoot

— Les opérations *ins* et *rmv* d'un même élément ne sont pas commutatives, les opérations *rmv* doivent être délivrées après les opérations *ins* correspondantes.

Logoot peut donc adopter le modèle de livraison *Exactly-once + Causal Remove* comme indiqué dans [5].

— Limites
  — Croissance non-bornée des identifiants
  — Même si expérience montre que taille reste limitée

— Plusieurs extensions furent proposées pour Logoot. WEISS, URSO et MOLLI [75] proposent une nouvelle stratégie d'allocation des identifiants pour `generateId`. Cette stratégie consiste à limiter la distance entre deux identifiants insérés au cours de la même modification *ins*, au lieu des les répartir de manière aléatoire entre *predId* et *succId*. Ceci permet de regrouper les identifiants des éléments insérés par une même modification et de laisser plus d'espace pour les insertions suivantes. En réduisant la vitesse à laquelle l'espace d'identifiants se sature pour une taille donnée d'identifiants, et donc la vitesse de croissance des identifiants, ce nouvel algorithme permet de réduire le surcoût en métadonnées, calculs et bande-passant du CRDT.

— Toujours dans [75], les auteurs introduisent *Logoot-Undo*, une version de Logoot dotée d'un mécanisme d'undo. Ce mécanisme prend la forme d'une nouvelle modification, *undo*, qui permet d'annuler l'effet d'une ou plusieurs modifications passées. Cette modification, et l'opération en résultant, est spécifiée de manière à être commutative avec toutes autres opérations concurrentes, c.-à-d. *ins*, *rmv* et *undo* elle-même.

— Pour définir *undo*, une notion de *degrée de visibilité* d'un élément est introduite. Elle permet à Logoot-Undo de déterminer si l'élément doit être affiché ou non. Pour cela, Logoot-Undo maintient une structure auxiliaire, le *Cimetière*, qui référence les identifiants des éléments dont le degrée est inférieur à 0 [20]. Ainsi, Logoot-Undo ne référence qu'un nombre réduit de pierres tombales. Qui plus est, ces pierres tombales sont stockées en dehors de la structure représentant la séquence et n'impactent donc pas les performances des modifications ultérieures.

— Nous illustrons le fonctionnement de ce mécanisme à l'aide de la Figure 1.19.

20. Nous pouvons dès lors inférer le degrée des identifiants restants en fonction de s'ils se trouvent dans la séquence (1) ou s'ils sont absents à la fois de la séquence et du cimetière (0).

TODO

FIGURE 1.19 – Modifications concurrentes d'une séquence répliquée Logoot-Undo

— Il convient de noter que l'ajout du degrée de visibilité des éléments permet de rendre commutatives l'opération *ins* avec l'opération *rmv* d'un même élément. Ainsi, Logoot-Undo ne nécessite comme modèle de livraison que le modèle *Eventual Delivery*.

— Finalement, ANDRÉ et al. [4] introduit *LogootSplit*. Reprenant les idées introduites par [72], ce travail présente un mécanisme d'aggrégation dynamiques des élements en blocs. Ceci permet de réduire la granularité des éléments stockés dans la séquence, et ainsi de réduire le surcoût en métadonnées, calculs et bande-passante du CRDT. Nous utilisons ce CRDT pour séquence comme base pour les travaux présentés dans ce manuscrit. Nous dédions donc la section 1.4 à sa présentation en détails.

Matthieu: TODO : Autres Sequence CRDTs à considérer : String-wise CRDT [72], Chronofold [76]

1.3.3 Synthèse

— Deux approches différentes pour la résolution de conflits ont été proposées pour CRDTs pour Séquence. Chacune de ces approches visent à minimiser surcoût du type répliqué, que ce soit d'un point de vue mémoire, computations et réseau.

— Au fil des années, ces approches ont été raffinées avec de nouveaux CRDTs de plus en plus en efficaces.

— Néanmoins, malgré les évaluations et comparaisons, la littérature n'a pas établi une supériorité d'une approche sur l'autre. Les approches proposent seulement des compromis différents sur la nature du surcoût, que nous récapitulons dans Tableau 1.2. L'approche basée sur pierres tombales offre une consommation réseau constante grâce à ses identifiants de taille fixe, mais souffre d'une consommation mémoire ne pouvant qu'augmenter. L'approche basée sur identifiants densément ordonnés bénéficie d'un meilleur délai de diffusion des modifications, les modifications pouvant être livrées dans le désordre, mais souffre d'une empreinte réseau augmentant avec la taille de ses identifiants. L'approche basée sur pierres tombales et l'approche basée sur identifiants densément ordonnés souffrent toutes les deux d'une augmentation théorique de leur surcoût en mémoire et en computations, respectivement dûe au nombre forcément croissant d'éléments stockées dans la Séquence et à la taille croissante [21] des identifiants associés aux éléments de la Séquence.

— Pour la suite de ce manuscrit, nous prenons LogootSplit comme base de travail. Nous détaillons donc son fonctionnement dans la section suivante.

21. [66] montre expérimentalement que les performances de l'approche basée sur identifiants densément ordonnés restent stables tout au long des tâches d'édition collaborative proposées.

TABLE 1.2 – Récapitulatif comparatif des différents approches pour CRDTs pour Séquence

	Pierres tombales	Identifiants denséments ordo
Performances stables en fct. de la taille de la séq.	✗	✗
Identifiants de taille fixe	✓	✗
Eléments réellement supprimés	✗	✓
Taille des messages fixe	✓	✗
Peut s'affranchir de la cohérence causale	✓	✓

1.4 LogootSplit

LogootSplit [4] est l'état de l'art des séquences répliquées à identifiants densément ordonnés. Comme expliqué précédemment, LogootSplit utilise des identifiants provenant d'un ordre total dense pour positionner les éléments dans la séquence répliquée.

1.4.1 Identifiants

Pour ce faire, LogootSplit assigne des identifiants composés d'une liste de tuples aux éléments. Les tuples sont définis de la manière suivante :

Définition 11 (Tuple) *Un* Tuple *est un quadruplet* ⟨*position, nodeId, nodeSeq, offset*⟩ *où*

- *position incarne la position souhaitée de l'élément.*
- *nodeId est l'identifiant unique du noeud qui a généré le tuple.*
- *nodeSeq est le numéro de séquence courant du noeud à la génération du tuple.*
- *offset indique la position de l'élément au sein d'un bloc. Nous reviendrons plus en détails sur ce composant dans la sous-section 1.4.2.*

Matthieu: TODO : Ajouter une relation d'ordre sur les tuples

Dans ce manuscrit, nous représentons les tuples par le biais de la notation suivante : $position_{offset}^{nodeId\ nodeSeq}$ où *position* est une lettre minuscule, *nodeId* une lettre majuscule et *nodeSeq* et *offset* des entiers, e.g. i_0^{B0}.

À partir de là, les identifiants LogootSplit sont définis de la manière suivante :

Définition 12 (Identifiant) *Un* Identifiant *est une liste de* Tuples.

Matthieu: TODO : Définir la notion de base (et autres fonctions utiles sur les identifiants ? genre isPrefix, concat, getTail...)

Nous représentons les identifiants en listant les tuples qui les composent. Par exemple, l'identifiant composé des tuples ⟨⟨i, B, 0, 0⟩⟨f, A, 0, 0⟩⟩ est présenté de la manière suivante : $i_0^{B0} f_0^{A0}$.

Les identifiants ont pour rôle d'ordonner les éléments relativement les uns par rapport aux autres. Pour ce faire, une relation d'ordre total aux identifiants est associée à l'ensemble des identifiants :

Bibliographie

[1] Mihai Letia, Nuno Preguiça et Marc Shapiro. « Consistency without concurrency control in large, dynamic systems ». In : *LADIS 2009 - 3rd ACM SIGOPS International Workshop on Large Scale Distributed Systems and Middleware*. T. 44. Operating Systems Review 2. Big Sky, MT, United States : Assoc. for Computing Machinery, oct. 2009, p. 29–34. DOI : 10.1145/1773912.1773921. URL : https://hal.inria.fr/hal-01248270.

[2] Marek Zawirski, Marc Shapiro et Nuno Preguiça. « Asynchronous rebalancing of a replicated tree ». In : *Conférence Française en Systèmes d'Exploitation (CFSE)*. Saint-Malo, France, mai 2011, p. 12. URL : https://hal.inria.fr/hal-01248197.

[3] Matthieu Nicolas et al. « MUTE : A Peer-to-Peer Web-based Real-time Collaborative Editor ». In : *ECSCW 2017 - 15th European Conference on Computer-Supported Cooperative Work*. T. 1. Proceedings of 15th European Conference on Computer-Supported Cooperative Work - Panels, Posters and Demos 3. Sheffield, United Kingdom : EUSSET, août 2017, p. 1–4. DOI : 10.18420/ecscw2017_p5. URL : https://hal.inria.fr/hal-01655438.

[4] Luc André et al. « Supporting Adaptable Granularity of Changes for Massive-Scale Collaborative Editing ». In : *International Conference on Collaborative Computing : Networking, Applications and Worksharing - CollaborateCom 2013*. Austin, TX, USA : IEEE Computer Society, oct. 2013, p. 50–59. DOI : 10.4108/icst.collaboratecom.2013.254123.

[5] Victorien Elvinger. « Réplication sécurisée dans les infrastructures pair-à-pair de collaboration ». Theses. Université de Lorraine, juin 2021. URL : https://hal.univ-lorraine.fr/tel-03284806.

[6] Hoang-Long Nguyen, Claudia-Lavinia Ignat et Olivier Perrin. « Trusternity : Auditing Transparent Log Server with Blockchain ». In : *Companion of the The Web Conference 2018*. Lyon, France, avr. 2018. DOI : 10.1145/3184558.3186938. URL : https://hal.inria.fr/hal-01883589.

[7] Hoang-Long Nguyen et al. « Blockchain-Based Auditing of Transparent Log Servers ». In : *32th IFIP Annual Conference on Data and Applications Security and Privacy (DBSec)*. Sous la dir. de Florian Kerschbaum et Stefano Paraboschi. T. LNCS-10980. Data and Applications Security and Privacy XXXII. Part 1 : Administration. Bergamo, Italy : Springer International Publishing, juil. 2018, p. 21–37. DOI : 10.1007/978-3-319-95729-6_2. URL : https://hal.archives-ouvertes.fr/hal-01917636.

[8] A. Das, I. Gupta et A. Motivala. « SWIM : scalable weakly-consistent infection-style process group membership protocol ». In : *Proceedings International Conference on Dependable Systems and Networks*. 2002, p. 303–312. DOI : 10.1109/DSN.2002.1028914.

[9] Armon Dadgar, James Phillips et Jon Currey. « Lifeguard : Local health awareness for more accurate failure detection ». In : *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. IEEE. 2018, p. 22–25.

[10] Brice Nédelec et al. « An adaptive peer-sampling protocol for building networks of browsers ». In : *World Wide Web* 21.3 (2018), p. 629–661.

[11] Mike Burmester et Yvo Desmedt. « A secure and efficient conference key distribution system ». In : *Advances in Cryptology — EUROCRYPT'94*. Sous la dir. d'Alfredo De Santis. Berlin, Heidelberg : Springer Berlin Heidelberg, 1995, p. 275–286. ISBN : 978-3-540-44717-7.

[12] Yasushi Saito et Marc Shapiro. « Optimistic Replication ». In : *ACM Comput. Surv.* 37.1 (mar. 2005), p. 42–81. ISSN : 0360-0300. DOI : 10.1145/1057977.1057980. URL : https://doi.org/10.1145/1057977.1057980.

[13] D. Abadi. « Consistency Tradeoffs in Modern Distributed Database System Design : CAP is Only Part of the Story ». In : *Computer* 45.2 (2012), p. 37–42. DOI : 10.1109/MC.2012.33.

[14] Rachid Guerraoui, Matej Pavlovic et Dragos-Adrian Seredinschi. « Trade-offs in replicated systems ». In : *IEEE Data Engineering Bulletin* 39.ARTICLE (2016), p. 14–26.

[15] Leslie Lamport. « Time, Clocks, and the Ordering of Events in a Distributed System ». In : *Commun. ACM* 21.7 (juil. 1978), p. 558–565. ISSN : 0001-0782. DOI : 10.1145/359545.359563. URL : https://doi.org/10.1145/359545.359563.

[16] Marc Shapiro et al. « Conflict-Free Replicated Data Types ». In : *Proceedings of the 13th International Symposium on Stabilization, Safety, and Security of Distributed Systems*. SSS 2011. 2011, p. 386–400. DOI : 10.1007/978-3-642-24550-3_29.

[17] Nuno M. Preguiça, Carlos Baquero et Marc Shapiro. « Conflict-free Replicated Data Types (CRDTs) ». In : *CoRR* abs/1805.06358 (2018). arXiv : 1805.06358. URL : http://arxiv.org/abs/1805.06358.

[18] Nuno M. Preguiça. « Conflict-free Replicated Data Types : An Overview ». In : *CoRR* abs/1806.10254 (2018). arXiv : 1806.10254. URL : http://arxiv.org/abs/1806.10254.

[19] B. A. Davey et H. A. Priestley. *Introduction to Lattices and Order*. 2e éd. Cambridge University Press, 2002. DOI : 10.1017/CBO9780511809088.

[20] Sebastian Burckhardt et al. « Replicated Data Types : Specification, Verification, Optimality ». In : *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. POPL '14. San Diego, California, USA : Association for Computing Machinery, 2014, p. 271–284. isbn : 9781450325448. doi : 10.1145/2535838.2535848. url : https://doi.org/10.1145/2535838.2535848.

[21] Gowtham Kaki et al. « Mergeable Replicated Data Types ». In : *Proc. ACM Program. Lang.* 3.OOPSLA (oct. 2019). doi : 10.1145/3360580. url : https://doi.org/10.1145/3360580.

[22] Paul R Johnson et Robert Thomas. *RFC0677 : Maintenance of duplicate databases*. RFC Editor, 1975.

[23] Weihai Yu et Sigbjørn Rostad. « A Low-Cost Set CRDT Based on Causal Lengths ». In : *Proceedings of the 7th Workshop on Principles and Practice of Consistency for Distributed Data*. New York, NY, USA : Association for Computing Machinery, 2020. isbn : 9781450375245. url : https://doi.org/10.1145/3380787.3393678.

[24] Marc Shapiro et al. *A comprehensive study of Convergent and Commutative Replicated Data Types*. Research Report RR-7506. Inria – Centre Paris-Rocquencourt ; INRIA, jan. 2011, p. 50. url : https://hal.inria.fr/inria-00555588.

[25] Carlos Baquero, Paulo Sérgio Almeida et Ali Shoker. « Making Operation-Based CRDTs Operation-Based ». In : *Proceedings of the First Workshop on Principles and Practice of Eventual Consistency*. PaPEC '14. Amsterdam, The Netherlands : Association for Computing Machinery, 2014. isbn : 9781450327169. doi : 10.1145/2596631.2596632. url : https://doi.org/10.1145/2596631.2596632.

[26] Carlos Baquero, Paulo Sergio Almeida et Ali Shoker. *Pure Operation-Based Replicated Data Types*. 2017. arXiv : 1710.04469 [cs.DC].

[27] Paulo Sérgio Almeida, Ali Shoker et Carlos Baquero. « Efficient State-Based CRDTs by Delta-Mutation ». In : *Networked Systems*. Sous la dir. d'Ahmed Bouajjani et Hugues Fauconnier. Cham : Springer International Publishing, 2015, p. 62–76. isbn : 978-3-319-26850-7.

[28] Paulo Sérgio Almeida, Ali Shoker et Carlos Baquero. « Delta state replicated data types ». In : *Journal of Parallel and Distributed Computing* 111 (jan. 2018), p. 162–173. issn : 0743-7315. doi : 10.1016/j.jpdc.2017.08.003. url : http://dx.doi.org/10.1016/j.jpdc.2017.08.003.

[29] Prince Mahajan, Lorenzo Alvisi, Mike Dahlin et al. « Consistency, availability, and convergence ». In : ().

[30] Ravi Prakash, Michel Raynal et Mukesh Singhal. « An Adaptive Causal Ordering Algorithm Suited to Mobile Computing Environments ». In : *Journal of Parallel and Distributed Computing* 41.2 (1997), p. 190–204. issn : 0743-7315. doi : https://doi.org/10.1006/jpdc.1996.1300. url : https://www.sciencedirect.com/science/article/pii/S0743731596913003.

[31] D. S. Parker et al. « Detection of Mutual Inconsistency in Distributed Systems ». In : *IEEE Trans. Softw. Eng.* 9.3 (mai 1983), p. 240–247. ISSN : 0098-5589. DOI : 10.1109/TSE.1983.236733. URL : https://doi.org/10.1109/TSE.1983.236733.

[32] Giuseppe DeCandia et al. « Dynamo : Amazon's highly available key-value store ». In : *ACM SIGOPS operating systems review* 41.6 (2007), p. 205–220.

[33] Nico Kruber, Maik Lange et Florian Schintke. « Approximate Hash-Based Set Reconciliation for Distributed Replica Repair ». In : *2015 IEEE 34th Symposium on Reliable Distributed Systems (SRDS)*. 2015, p. 166–175. DOI : 10.1109/SRDS.2015.30.

[34] Ricardo Jorge Tomé Gonçalves et al. « DottedDB : Anti-Entropy without Merkle Trees, Deletes without Tombstones ». In : *2017 IEEE 36th Symposium on Reliable Distributed Systems (SRDS)*. 2017, p. 194–203. DOI : 10.1109/SRDS.2017.28.

[35] Jim Bauwens et Elisa Gonzalez Boix. « Improving the Reactivity of Pure Operation-Based CRDTs ». In : *Proceedings of the 8th Workshop on Principles and Practice of Consistency for Distributed Data*. PaPoC '21. Online, United Kingdom : Association for Computing Machinery, 2021. ISBN : 9781450383387. DOI : 10.1145/3447865.3457968. URL : https://doi.org/10.1145/3447865.3457968.

[36] V. Enes et al. « Efficient Synchronization of State-Based CRDTs ». In : *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. 2019, p. 148–159. DOI : 10.1109/ICDE.2019.00022.

[37] Petru Nicolaescu et al. « Yjs : A Framework for Near Real-Time P2P Shared Editing on Arbitrary Data Types ». In : *15th International Conference on Web Engineering*. ICWE 2015. Springer LNCS volume 9114, juin 2015, p. 675–678. DOI : 10.1007/978-3-319-19890-3_55. URL : http://dbis.rwth-aachen.de/~derntl/papers/preprints/icwe2015-preprint.pdf.

[38] Petru Nicolaescu et al. « Near Real-Time Peer-to-Peer Shared Editing on Extensible Data Types ». In : *19th International Conference on Supporting Group Work*. GROUP 2016. ACM, nov. 2016, p. 39–49. DOI : 10.1145/2957276.2957310.

[39] Yjs. *Yjs : A CRDT framework with a powerful abstraction of shared data*. URL : https://github.com/yjs/yjs.

[40] Martin Kleppmann et Alastair R. Beresford. « A Conflict-Free Replicated JSON Datatype ». In : *IEEE Transactions on Parallel and Distributed Systems* 28.10 (oct. 2017), p. 2733–2746. ISSN : 1045-9219. DOI : 10.1109/tpds.2017.2697382. URL : http://dx.doi.org/10.1109/TPDS.2017.2697382.

[41] Automerge. *Automerge : data structures for building collaborative applications in Javascript*. URL : https://github.com/automerge/automerge.

[42] Christopher Meiklejohn et Peter Van Roy. « Lasp : A Language for Distributed, Coordination-free Programming ». In : *17th International Symposium on Principles and Practice of Declarative Programming*. PPDP 2015. ACM, juil. 2015, p. 184–195. DOI : 10.1145/2790449.2790525.

[43] Kevin De Porre et al. « CScript : A distributed programming language for building mixed-consistency applications ». In : *Journal of Parallel and Distributed Computing volume 144* (oct. 2020), p. 109–123. ISSN : 0743-7315. DOI : 10.1016/j.jpdc.2020.05.010.

[44] Riak. *Riak KV*. URL : http://riak.com/.

[45] The SyncFree Consortium. *AntidoteDB : A planet scale, highly available, transactional database*. URL : http://antidoteDB.eu/.

[46] C. Wu et al. « Anna : A KVS for Any Scale ». In : *IEEE Transactions on Knowledge and Data Engineering* 33.2 (2021), p. 344–358. DOI : 10.1109/TKDE.2019.2898401.

[47] Concordant. *Concordant*. URL : http://www.concordant.io/.

[48] Weihai Yu et Claudia-Lavinia Ignat. « Conflict-Free Replicated Relations for Multi-Synchronous Database Management at Edge ». In : *IEEE International Conference on Smart Data Services, 2020 IEEE World Congress on Services*. Beijing, China, oct. 2020. URL : https://hal.inria.fr/hal-02983557.

[49] Martin Kleppmann et al. « Local-First Software : You Own Your Data, in Spite of the Cloud ». In : *Proceedings of the 2019 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*. Onward ! 2019. Athens, Greece : Association for Computing Machinery, 2019, p. 154–178. ISBN : 9781450369954. DOI : 10.1145/3359591.3359737. URL : https://doi.org/10.1145/3359591.3359737.

[50] Peter van Hardenberg et Martin Kleppmann. « PushPin : Towards Production-Quality Peer-to-Peer Collaboration ». In : *7th Workshop on Principles and Practice of Consistency for Distributed Data*. PaPoC 2020. ACM, avr. 2020. DOI : 10.1145/3380787.3393683.

[51] Brice Nédelec, Pascal Molli et Achour Mostefaoui. « CRATE : Writing Stories Together with our Browsers ». In : *25th International World Wide Web Conference*. WWW 2016. ACM, avr. 2016, p. 231–234. DOI : 10.1145/2872518.2890539.

[52] C. A. Ellis et S. J. Gibbs. « Concurrency Control in Groupware Systems ». In : *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data*. SIGMOD '89. Portland, Oregon, USA : Association for Computing Machinery, 1989, p. 399–407. ISBN : 0897913175. DOI : 10.1145/67544.66963. URL : https://doi.org/10.1145/67544.66963.

[53] Chengzheng Sun et Clarence Ellis. « Operational transformation in real-time group editors : issues, algorithms, and achievements ». In : *Proceedings of the 1998 ACM conference on Computer supported cooperative work*. 1998, p. 59–68.

[54] Matthias Ressel, Doris Nitsche-Ruhland et Rul Gunzenhäuser. « An integrating, transformation-oriented approach to concurrency control and undo in group editors ». In : *Proceedings of the 1996 ACM conference on Computer supported cooperative work*. 1996, p. 288–297.

[55] Chengzheng Sun et al. « A consistency model and supporting schemes for real-time cooperative editing systems ». In : *Australian Computer Science Communications* 18 (1996), p. 582–591.

[56] David Sun et Chengzheng Sun. « Context-Based Operational Transformation in Distributed Collaborative Editing Systems ». In : *Parallel and Distributed Systems, IEEE Transactions on* 20 (nov. 2009), p. 1454–1470. DOI : 10.1109/TPDS.2008.240.

[57] Gérald Oster et al. « Tombstone Transformation Functions for Ensuring Consistency in Collaborative Editing Systems ». In : *2006 International Conference on Collaborative Computing : Networking, Applications and Worksharing*. 2006, p. 1–10. DOI : 10.1109/COLCOM.2006.361867.

[58] Chengzheng Sun et al. « Achieving Convergence, Causality Preservation, and Intention Preservation in Real-Time Cooperative Editing Systems ». In : *ACM Trans. Comput.-Hum. Interact.* 5.1 (mar. 1998), p. 63–108. ISSN : 1073-0516. DOI : 10.1145/274444.274447. URL : https://doi.org/10.1145/274444.274447.

[59] Stéphane Weiss, Pascal Urso et Pascal Molli. « Logoot : A Scalable Optimistic Replication Algorithm for Collaborative Editing on P2P Networks ». In : *Proceedings of the 29th International Conference on Distributed Computing Systems - ICDCS 2009*. Montreal, QC, Canada : IEEE Computer Society, juin 2009, p. 404–412. DOI : 10.1109/ICDCS.2009.75. URL : http://doi.ieeecomputersociety.org/10.1109/ICDCS.2009.75.

[60] Bernadette Charron-Bost. « Concerning the size of logical clocks in distributed systems ». In : *Information Processing Letters* 39.1 (1991), p. 11–16.

[61] Gérald Oster et al. « Data Consistency for P2P Collaborative Editing ». In : *ACM Conference on Computer-Supported Cooperative Work - CSCW 2006*. Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work. Banff, Alberta, Canada : ACM Press, nov. 2006, p. 259–268. URL : https://hal.inria.fr/inria-00108523.

[62] Hyun-Gul Roh et al. « Replicated abstract data types : Building blocks for collaborative applications ». In : *Journal of Parallel and Distributed Computing* 71.3 (2011), p. 354–368. ISSN : 0743-7315. DOI : https://doi.org/10.1016/j.jpdc.2010.12.006. URL : http://www.sciencedirect.com/science/article/pii/S0743731510002716.

[63] Nuno Preguica et al. « A Commutative Replicated Data Type for Cooperative Editing ». In : *2009 29th IEEE International Conference on Distributed Computing Systems*. Juin 2009, p. 395–403. DOI : 10.1109/ICDCS.2009.20.

[64] Marc Shapiro et Nuno Preguiça. *Designing a commutative replicated data type*. Research Report RR-6320. INRIA, 2007. URL : https://hal.inria.fr/inria-00177693.

[65] Stéphane Weiss, Pascal Urso et Pascal Molli. « Wooki : a P2P Wiki-based Collaborative Writing Tool ». In : t. 4831. Déc. 2007. ISBN : 978-3-540-76992-7. DOI : 10.1007/978-3-540-76993-4_42.

[66] Mehdi Ahmed-Nacer et al. « Evaluating CRDTs for Real-time Document Editing ». In : *11th ACM Symposium on Document Engineering*. Sous la dir. d'ACM. Mountain View, California, United States, sept. 2011, p. 103–112. DOI : 10.1145/2034691.2034717. URL : https://hal.inria.fr/inria-00629503.

[67] Ben Shneiderman. « Response Time and Display Rate in Human Performance with Computers ». In : *ACM Comput. Surv.* 16.3 (sept. 1984), p. 265–285. ISSN : 0360-0300. DOI : 10.1145/2514.2517. URL : https://doi.org/10.1145/2514.2517.

[68] Caroline Jay, Mashhuda Glencross et Roger Hubbold. « Modeling the Effects of Delayed Haptic and Visual Feedback in a Collaborative Virtual Environment ». In : *ACM Trans. Comput.-Hum. Interact.* 14.2 (août 2007), 8–es. ISSN : 1073-0516. DOI : 10.1145/1275511.1275514. URL : https://doi.org/10.1145/1275511.1275514.

[69] Hagit Attiya et al. « Specification and Complexity of Collaborative Text Editing ». In : *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing*. PODC '16. Chicago, Illinois, USA : Association for Computing Machinery, 2016, p. 259–268. ISBN : 9781450339643. DOI : 10.1145/2933057.2933090. URL : https://doi.org/10.1145/2933057.2933090.

[70] Hagit Attiya et al. « Specification and space complexity of collaborative text editing ». In : *Theoretical Computer Science* 855 (2021), p. 141–160. ISSN : 0304-3975. DOI : https://doi.org/10.1016/j.tcs.2020.11.046. URL : http://www.sciencedirect.com/science/article/pii/S0304397520306952.

[71] Loïck Briot, Pascal Urso et Marc Shapiro. « High Responsiveness for Group Editing CRDTs ». In : *ACM International Conference on Supporting Group Work*. Sanibel Island, FL, United States, nov. 2016. DOI : 10.1145/2957276.2957300. URL : https://hal.inria.fr/hal-01343941.

[72] Weihai Yu. « A String-Wise CRDT for Group Editing ». In : *Proceedings of the 17th ACM International Conference on Supporting Group Work*. GROUP '12. Sanibel Island, Florida, USA : Association for Computing Machinery, 2012, p. 141–144. ISBN : 9781450314862. DOI : 10.1145/2389176.2389198. URL : https://doi.org/10.1145/2389176.2389198.

[73] Martin Kleppmann et al. « Interleaving Anomalies in Collaborative Text Editors ». In : *Proceedings of the 6th Workshop on Principles and Practice of Consistency for Distributed Data*. PaPoC '19. Dresden, Germany : Association for Computing Machinery, 2019. ISBN : 9781450362764. DOI : 10.1145/3301419.3323972. URL : https://doi.org/10.1145/3301419.3323972.

[74] Matthew Weidner. *There Are No Doubly Non-Interleaving List CRDTs*. URL : https://mattweidner.com/assets/pdf/List_CRDT_Non_Interleaving.pdf.

[75] Stéphane Weiss, Pascal Urso et Pascal Molli. « Logoot-Undo : Distributed Collaborative Editing System on P2P Networks ». In : *IEEE Transactions on Parallel and Distributed Systems* 21.8 (août 2010), p. 1162–1174. DOI : 10.1109/TPDS.2009.173. URL : https://hal.archives-ouvertes.fr/hal-00450416.

[76] Victor GRISHCHENKO et Mikhail PATRAKEEV. « Chronofold : A Data Structure for Versioned Text ». In : *Proceedings of the 7th Workshop on Principles and Practice of Consistency for Distributed Data*. PaPoC '20. Heraklion, Greece : Association for Computing Machinery, 2020. ISBN : 9781450375245. DOI : 10.1145/3380787.3393680. URL : https://doi.org/10.1145/3380787.3393680.

[77] Elena YANAKIEVA et al. « Access Control Conflict Resolution in Distributed File Systems Using CRDTs ». In : *Proceedings of the 8th Workshop on Principles and Practice of Consistency for Distributed Data*. PaPoC '21. Online, United Kingdom : Association for Computing Machinery, 2021. ISBN : 9781450383387. DOI : 10.1145/3447865.3457970. URL : https://doi.org/10.1145/3447865.3457970.

[78] Pierre-Antoine RAULT, Claudia-Lavinia IGNAT et Olivier PERRIN. « Distributed Access Control for Collaborative Applications Using CRDTs ». In : *Proceedings of the 9th Workshop on Principles and Practice of Consistency for Distributed Data*. PaPoC '22. Rennes, France : Association for Computing Machinery, 2022, p. 33–38. ISBN : 9781450392563. DOI : 10.1145/3517209.3524826. URL : https://doi.org/10.1145/3517209.3524826.