1.3 Types de données répliquées sans conflits

Pour limiter la coordination entre les noeuds, les systèmes distribués adoptent le paradigme de la réplication optimiste. Ce paradigme consiste à ce que chaque noeud possède une copie de la donnée. Chaque noeud possède le droit de la modifier sans se coordonner avec les autres noeuds. Noeuds peuvent alors temporairement diverger, c.-à-d. posséder des états différents. Un mécanisme de synchronisation leur permet ensuite de partager leurs modifications respectives et de nouveau converger. Ce paradigme offre ainsi aux noeuds une haute disponibilité *Matthieu: TODO : insérer ref sticky model* ainsi qu'une faible latence.

Afin d'ordonner les modifications effectués dans un système, la littérature repose généralement sur la relation *happens-before*[41]. Nous l'adaptons ci-dessous à notre contexte :

Définition 1 (Relation *happens-before*) *La relation* happens-before*indique qu'une modification m_1 a eu lieu avant une modification m_2, notée $m_1 \rightarrow m_2$, si et seulement si une des conditions suivantes est respectée :*

(i) m_1 *a eu lieu avant* m_2 *sur le même noeud.*

(ii) m_1 *a été délivrée au noeud auteur de* m_2 *avant la génération de* m_2*.*

(iii) Il existe une modification m *telle que* $m_1 \rightarrow m \wedge m \rightarrow m_2$*.*

Dans le cadre d'un système distribué, on note que la relation *happens-before*ne permet pas d'établir un ordre total entre les modifications [1]. En effet, deux modifications m_1 et m_2 peuvent être effectuées en parallèle par deux noeuds différents, sans avoir connaissance de la modification de leur pair respectif. De telles modifications sont alors dites *concurrentes* :

Définition 2 (Concurrence) *Deux modifications m_1 et m_2 sont concurrentes, noté $m_1 \parallel m_2$, si et seulement si $m_1 \nrightarrow m_2 \wedge m_1 \nrightarrow m_2$.*

Lorsque les modifications possibles sur un type de données sont commutatives, l'intégration des modifications effectuées par les autres noeuds, même concurrentes, ne nécessite aucun mécanisme particulier. Cependant, généralement les modifications permises par un type de données ne sont pas commutatives car de sémantiques contraires. Ainsi, une exécution dans un système distribué suivant le paradigme de réplication optimiste peut mener à la génération de modifications concurrentes non commutatives. Nous parlons alors de conflits. La figure Figure 1.1 présente un scénario où des modifications de sémantiques opposées sont générées en concurrence.

Dans cet exemple, deux noeuds A et B répliquent et partagent un même Ensemble. Les deux noeuds possèdent le même état initial : $\{a\}$. Le noeud A retire l'élément a de l'ensemble, $rmv(a)$. Puis, le noeud A ré-ajoute l'élément a dans l'ensemble via l'opération de modification $add(a)$. En concurrence, le noeud B retire lui aussi l'élément a de l'ensemble. Les deux noeuds se synchronisent ensuite.

À l'issue de ce scénario, l'état à produire n'est pas trivial : le noeud A a exprimé son intention d'ajouter l'élément a à l'ensemble, tandis que le noeud B a exprimé son intention

1. Nous utilisons le terme *modifications* pour désigner les *opérations de modifications* des types abstraits de données afin d'éviter une confusion avec le terme *opération* introduit ultérieurement.

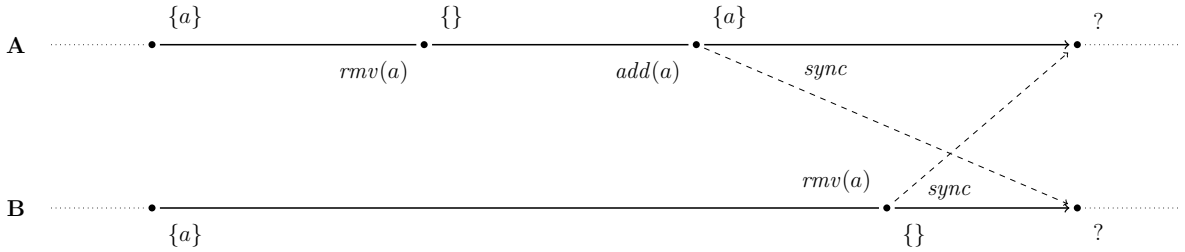


FIGURE 1.1 – Modifications concurrentes d'un Ensemble répliqué provoquant un conflit provoqué suite à l'ajout et la suppression d'un même élément

contraire de retirer l'élément a de ce même ensemble. Ainsi, les états $\{a\}$ et $\{\}$ semblent tous les deux corrects et légitimes dans cette situation. Il est néanmoins primordial que les noeuds choisissent et convergent vers un même état pour leur permettre de poursuivre leur collaboration. Pour ce faire, il est nécessaire de mettre en place un mécanisme de résolution de conflits, potentiellement automatique.

Les CRDTs répondent à ce besoin. Les CRDTs sont de nouvelles spécifications des types de données existants, e.g. l'Ensemble ou la Séquence. Ces nouvelles spécifications sont conçues pour être utilisées dans des systèmes distribués adoptant la réplication optimiste. Ainsi, elles offrent les deux propriétés suivantes :

(i) Les CRDTs peuvent être modifiés sans coordination avec les autres noeuds.

(ii) Tous les noeuds ayant observés le même ensemble de modifications obtiendront des états équivalents. Cette propriété est connue sous le nom de *convergence forte* *Matthieu: TODO : Ajouter ref.*

Pour offrir la propriété de *convergence forte*, la spécification des CRDTs reposent sur la théorie des treillis :

(i) Les différents états possibles d'un CRDT forment un sup-demi-treillis, possédant une relation d'ordre partiel $\leq$.

(ii) Les modifications génèrent par inflation un nouvel état supérieur ou égal à l'état original d'après $\leq$.

(iii) Il existe une fonction de fusion qui, pour toute paire d'états, génère l'état minimal supérieur d'après $\leq$ aux deux états fusionnés. On parle aussi de borne supérieure ou de Least Upper Bound (LUB) pour catégoriser l'état résultant de cette fusion.

Malgré leur spécification différente, les CRDTs partagent la même sémantique et interface que les types non-répliqués correspondant du point de vue des utilisateurs. Ainsi, les CRDTs partagent le comportement des types non-répliqués dans le cadre d'exécutions séquentielles. Cependant, ils définissent aussi une nouvelle sémantique pour chaque nouveau scénario ne pouvant se produire que dans le cadre d'une exécution distribuée.

Plusieurs sémantiques valides peuvent être proposées pour ces nouveaux scénarios. Un CRDT se doit donc de préciser quelle sémantique il choisit.

L'autre aspect définissant un CRDT donné est le modèle qu'il adopte pour propager les modifications. Au fil des années, la littérature a établi et défini plusieurs modèles dit de synchronisation, chacun ayant ses propres besoins et avantages. De fait, plusieurs CRDTs

peuvent être proposés pour un même type donné en fonction du modèle de synchronisation choisi.

Ainsi, ce qui définit un CRDT est sa sémantique en cas de concurrence et son modèle de synchronisation. Dans les prochaines sections, nous présenterons les différentes sémantiques possibles pour un type donné, l'Ensemble, en guise d'exemple. Puis nous présenterons les différents modèles de synchronisation proposés dans la littérature, et détaillerons leurs contraintes et impact sur les CRDT les adoptant, toujours en utilisant le même exemple.

Matthieu: TODO : Faire le lien avec les travaux de Burckhardt [13] et les MRDTs [37]

1.3.1 Sémantique en cas de concurrence

Plusieurs sémantiques peuvent être proposées pour résoudre les conflits. Certaines de ces sémantiques ont comme avantage d'être générique, c.-à-d. applicable à l'ensemble des types de données. En contrepartie, elles souffrent de cette même généricité, en ne permettant que des comportements simples en cas de conflits.

À l'inverse, la majorité des sémantiques proposées dans la littérature sont spécifiques à un type de données. Elles visent ainsi à prendre plus finement en compte l'intention des modifications pour proposer des comportements plus précis.

Dans la suite de cette section, nous présentons ces sémantiques génériques ainsi que celles spécifiques à l'Ensemble et, à titre d'exemple, les illustrons à l'aide du scénario présenté dans la Figure 1.1.

Sémantique *Last-Writer-Wins*

Une manière simple pour résoudre un conflit consiste à trancher de manière arbitraire et de sélectionner une modification parmi l'ensemble des modifications en conflit. Pour faire cela de manière déterministe, une approche est de reproduire et d'utiliser l'ordre total sur les modifications qui serait instauré par une horloge globale pour choisir la modification à prioritiser.

Cette approche, présentée dans [36], correspond à la sémantique nommée *Last-Writer-Wins* (LWW). De par son fonctionnement, cette sémantique est générique et est donc utilisée par une variété de CRDTs pour des types différents. La Figure 1.2 illustre son application à l'Ensemble pour résoudre le conflit de la Figure 1.1.

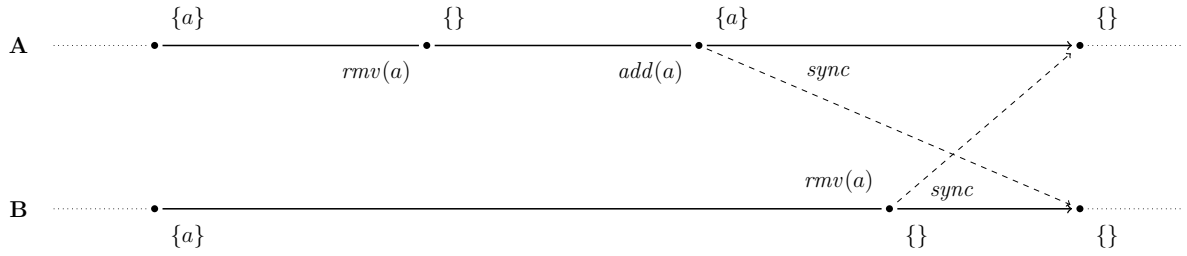


FIGURE 1.2 – Résolution du conflit en utilisant la sémantique LWW

Comme indiqué précédemment, le scénario illustré dans la Figure 1.2 présente un conflit entre les modifications concurrentes $add(a)$ et $rmv(a)$ générées de manière concurrente respectivement par les noeuds A et B. Pour le résoudre, la sémantique LWW associe à chaque modification une estampille. L'ordre créé entre les modifications par ces dernières permet de déterminer quelle modification désigner comme prioritaire. Ici, nous considérons que $add(a)$ a eu lieu plus tôt que $rmv(a)$. La sémantique LWW désigne donc $rmv(a)$ comme prioritaire et ignore $add(a)$. L'état obtenu à l'issue de cet exemple par chaque noeud est donc $\{\}$.

Il est à noter que si la modification $rmv(a)$ du noeud B avait eu lieu plus tôt dans notre exemple, l'état final obtenu aurait été $\{a\}$. Ainsi, des exécutions reproduisant le même ensemble de modifications produiront des résultats différents en fonction de l'ordre créé par les estampilles associées à chaque modification. Ces estampilles étant des métadonnées du mécanisme de résolution de conflits, elles sont dissimulées aux utilisateur-rices. Le comportement de cette sémantique peut donc être perçu comme aléatoire et s'avérer perturbant pour les utilisateur-rices.

La sémantique LWW repose sur l'horloge de chaque noeud pour attribuer une estampille à chacune de leurs modifications. Les horloges physiques étant sujettes à des imprécisions et notamment des décalages, utiliser les estampilles qu'elles fournissent peut provoquer des anomalies vis-à-vis de la relation *happens-before*. Les systèmes distribués préfèrent donc généralement utiliser des horloges logiques [41]. *Matthieu: TODO : Ajouter refs des horloges logiques plus intelligentes (Interval Tree Clock, Hybrid Clock...)*

Sémantique *Multi-Value*

Une seconde sémantique générique[2] est la sémantique *Multi-Value* (MV). Cette approche propose de gérer les conflits de la manière suivante : plutôt que de prioritiser une modification par rapport aux autres modifications concurrentes, la sémantique MV maintient l'ensemble des états résultant possibles. Nous présentons son application à l'Ensemble dans la Figure 1.3.

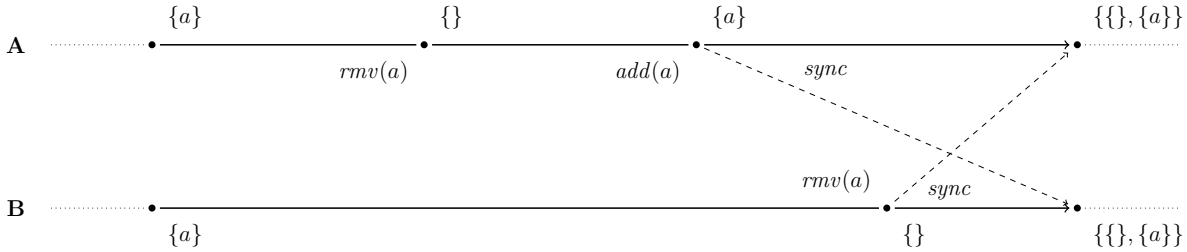


FIGURE 1.3 – Résolution du conflit en utilisant la sémantique MV

La Figure 1.3 présente la gestion du conflit entre les modifications concurrentes $add(a)$ et $rmv(a)$ par la sémantique MV. Devant ces modifications contraires, chaque noeud calcule chaque état possible, c.-à-d. un état sans l'élément a, $\{\}$, et un état avec ce dernier, $\{a\}$. Le CRDT maintient alors l'ensemble de ces états en parallèle. L'état obtenu est donc $\{\{\}, \{a\}\}$.

2. Bien que généralement associée au type *Registre* uniquement.

Ainsi, la sémantique MV expose les conflits aux utilisateur-rices lors de leur prochaine consultation de l'état du CRDT. Les utilisateur-rices peuvent alors prendre connaissance des intentions de chacun-e et résoudre le conflit manuellement. Dans la Figure 1.3, résoudre le conflit revient à re-effectuer une modification $add(a)$ ou $rmv(a)$ selon l'état choisi. Ainsi, si plusieurs personnes résolvent en concurrence le conflit de manière contraire, la sémantique MV exposera de nouveau les différents états proposés sous la forme d'un conflit.

Il est intéressant de noter que cette sémantique mène à un changement du domaine du CRDT considéré : en cas de conflit, la valeur retournée par le CRDT correspond à un Ensemble de valeurs du type initialement considéré. E.g. si nous considérons que le type correspondant au CRDT dans la Figure 1.3 est le type $Set\langle V\rangle$, nous observons que la valeur finale obtenue a pour type $Set\langle Set\langle V\rangle\rangle$. Il s'agit à notre connaissance de la seule sémantique opérant ce changement.

Sémantiques *Add-Wins* et *Remove-Wins*

Comme évoqué précédemment, d'autres sémantiques sont spécifiques au type de données concerné. Ainsi, nous abordons à présent des sémantiques spécifiques au type de l'Ensemble.

Dans le cadre de l'Ensemble, un conflit est provoqué lorsque des modifications *add* et *rmv* d'un même élément sont effectuées en concurrence. Ainsi, deux approches peuvent être proposées pour résoudre le conflit :

(i) Une sémantique où la modification *add* d'une élément prend la précédence sur les modifications concurrentes *rmv* du même élément, nommée *Add-Wins* (AW). L'élément est alors présent dans l'état obtenu à l'issue de la résolution du conflit.

(ii) Une sémantique où la modification *rmv* d'une élément prend la précédence sur les opérations concurrentes *add* du même élément, nommée *Remove-Wins* (RW). L'élément est alors absent de l'état obtenu à l'issue de la résolution du conflit.

La Figure 1.4 illustre l'application de chacune de ces sémantiques sur notre exemple.

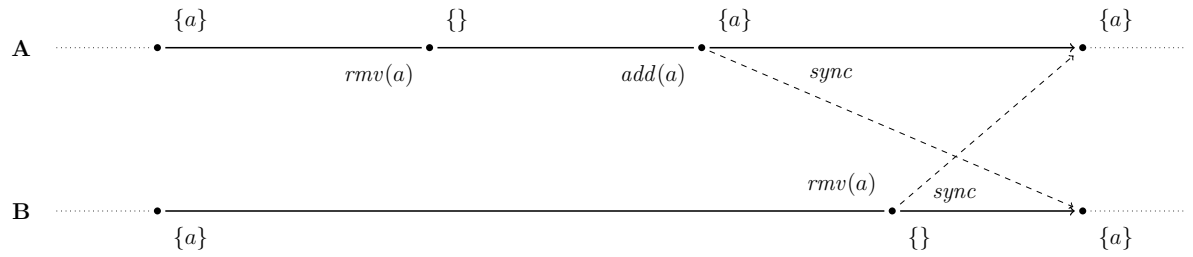
Sémantique *Causal-Length*

Une nouvelle sémantique pour l'Ensemble fut proposée [80] récemment. Cette sémantique se base sur les observations suivantes :

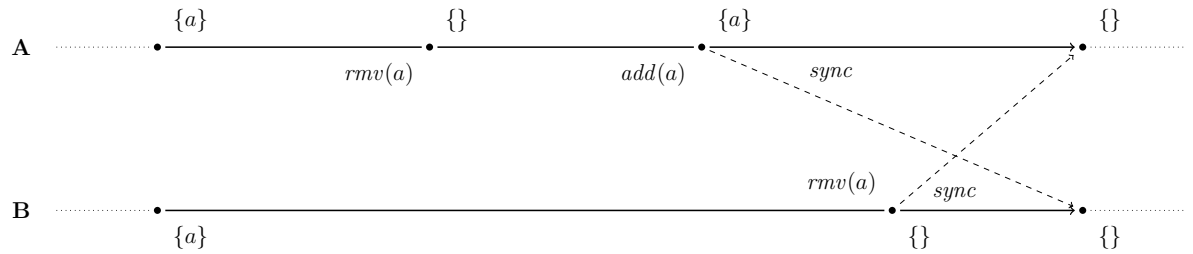
(i) *add* et *rmv* d'un élément prennent place à tour de rôle, chaque modification invalidant la précédente.

(ii) *add* (resp. *rmv*) concurrents d'un même élément représentent la même intention. Prendre en compte une de ces modifications concurrentes revient à prendre en compte leur ensemble.

À partir de ces observations, Yu et Rostad proposent de déterminer pour chaque élément la chaîne d'ajouts et retraits la plus longue. C'est cette chaîne, et précisément son dernier maillon, qui indique si l'élement est présent ou non dans l'ensemble final. La Figure 1.5 illustre son fonctionnement.



(a) Application de la sémantique AW



(b) Application de la sémantique RW

FIGURE 1.4 – Résolution du conflit en utilisant soit la sémantique AW, soit la sémantique RW

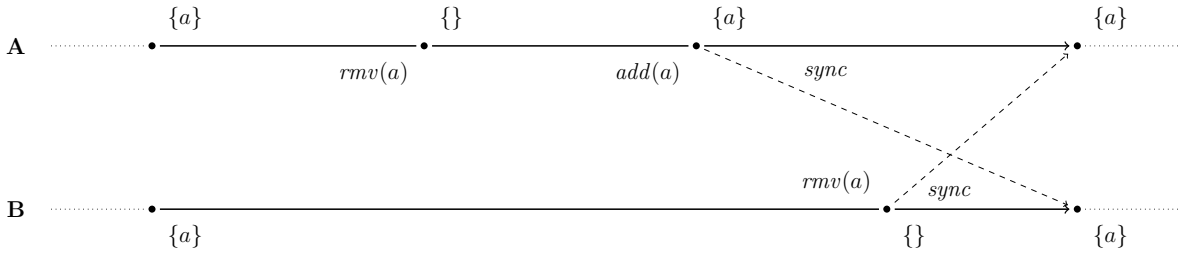


FIGURE 1.5 – Résolution du conflit en utilisant la sémantique CL

Dans notre exemple, la modification $rmv(a)$ effectuée par B est en concurrence avec une modification identique effectuée par A. La sémantique CL définit que ces deux modifications partagent la même intention. Ainsi, A ayant déjà appliqué sa propre modification préalablement, il ne prend pas en compte *de nouveau* cette modification lorsqu'il la reçoit de B. Son état reste donc inchangé.

À l'inverse, la modification $add(a)$ effectuée par A fait suite à sa modification $rmv(a)$. La sémantique CL définit alors qu'elle fait suite à toute autre modification $rmv(a)$ concurrente. Ainsi, B intègre cette modification lorsqu'il la reçoit de A. Son état évolue donc pour devenir $\{a\}$.

Synthèse

Dans cette section, nous avons mis en lumière l'existence de solutions différentes pour résoudre un même conflit. Chacune de ces solutions correspond à une sémantique spécifique de résolution de conflits. Ainsi, pour un même type de données, différents CRDTs

peuvent être spécifiés. Chacun de ces CRDTs est spécifié par la combinaison de sémantiques qu'il adopte, chaque sémantique servant à résoudre un des types de conflits du type de données.

Il est à noter qu'aucune sémantique n'est intrinsèquement meilleure et préférable aux autres. Il revient aux concepteur-rices d'applications de choisir les CRDTs adaptés en fonction des besoins et des comportements attendus en cas de conflits.

Par exemple, pour une application collaborative de listes de courses, l'utilisation d'un MV-Registre pour représenter le contenu de la liste se justifie : cette sémantique permet d'exposer les modifications concurrentes aux utilisateur-rices. Ainsi, les personnes peuvent détecter et résoudre les conflits provoquées par ces éditions concurrentes, e.g. l'ajout de l'élément *lait* à la liste, pour cuisiner des crêpes, tandis que les *oeufs* nécessaires à ces mêmes crêpes sont retirés. En parallèle, cette même application peut utiliser un LWW-Registre pour représenter et indiquer aux utilisateur-rices la date de la dernière modification effectuée.

1.3.2 Modèle de synchronisation

Dans le modèle de réplication optimiste, les noeuds divergent momentanément lorsqu'ils effectuent des modifications locales. Pour ensuite converger vers des états équivalents, les noeuds doivent propager et intégrer l'ensemble des modifications. La Figure 1.6 illustre ce point.

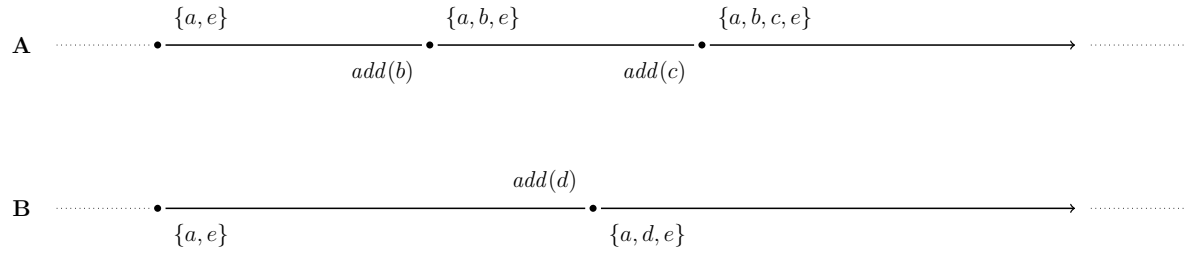


FIGURE 1.6 – Modifications en concurrence d'un Ensemble répliqué par les noeuds A et B

Dans cet exemple, deux noeuds A et B partagent et éditent un même Ensemble à l'aide d'un CRDT. Les deux noeuds possèdent le même état initial : $\{a, e\}$.

Le noeud A effectue les modifications $add(b)$ puis $add(c)$. Il obtient ainsi l'état $\{a, b, c, e\}$. De son côté, le noeud B effectue la modification suivante : $add(d)$. Son état devient donc $\{a, d, e\}$. Ainsi, les noeuds doivent encore s'échanger leur modifications pour converger vers l'état souhaité [3], c.-à-d. $\{a, b, c, d, e\}$.

Dans le cadre des CRDTs, le choix de la méthode pour synchroniser les noeuds n'est pas anodin. En effet, ce choix impacte la spécification même du CRDT et ses prérequis.

Initialement, deux approches ont été proposées : une méthode de synchronisation par états [68, 67] et une méthode de synchronisation par opérations [68, 67, 9, 8]. Une troisième

3. Le scénario ne comportant uniquement des modifications *add*, aucun conflit n'est produit malgré la concurrence des modifications.

approche, nommée synchronisation par différence d'états [4, 3], fut spécifiée par la suite. Le but de cette dernière est d'allier le meilleur des deux approches précédentes.

Dans la suite de cette section, nous présentons ces approches ainsi que leurs caractéristiques respectives. Pour les illustrer, nous complétons l'exemple décrit ici. Cependant, nous nous focalisons uniquement sur les messages envoyés par les noeuds et n'évoquons seulement les métadonnées introduites par chaque modèle de synchronisation, par soucis de clarté et de simplicité.

Synchronisation par états

L'approche de la synchronisation par états propose que les noeuds diffusent leurs modifications en transmettant leur état. Les CRDTs adoptant cette approche doivent définir une fonction `merge`. Cette fonction correspond à la fonction de fusion mentionnée précédemment en (iii) : elle prend en paramètres une paire d'états et génère en retour l'état correspondant à leur LUB. Cette fonction doit être associative, commutative et idempotente.

Ainsi, lorsqu'un noeud reçoit l'état d'un autre noeud, il fusionne ce dernier avec son état courant à l'aide de la fonction `merge`. Il obtient alors un nouvel état intégrant l'ensemble des modifications ayant été effectuées sur les deux états.

La nature croissante des états des CRDTs couplée aux propriétés d'associativité, de commutativité et d'idempotence de la fonction `merge` permettent de reposer sur la couche réseau sans lui imposer de contraintes fortes : les messages peuvent être perdus, réordonnés ou même dupliqués. Les noeuds convergeront tant que la couche réseau garantit que les noeuds seront capables de transmettre leur état aux autres à terme. Il s'agit là de la principale force des CRDTs synchronisés par états.

Néanmoins, la définition de la fonction `merge` offrant ces propriétés peut s'avérer complexe et a des répercussions sur la spécification même du CRDT. Notamment, les états doivent conserver une trace de l'existence des éléments et de leur suppression afin d'éviter qu'une fusion d'états ne les fassent ressurgirent. Ainsi, les CRDTs synchronisés par états utilisent régulièrement des pierres tombales.

En plus de l'utilisation de pierres tombales, la taille de l'état peut croître de manière non-bornée dans le cas de certains types de donnés, e.g. l'Ensemble ou la Séquence. Ainsi, ces structures peuvent atteindre à terme des tailles conséquentes. Dans de tels cas, diffuser l'état complet à chaque modification induirait alors un coût rédhibitoire. L'approche de la synchronisation par états s'avère donc inadaptée aux systèmes temps réel et repose généralement sur une synchronisation périodique.

Nous illustrons le fonctionnement de cette approche avec la Figure 1.7. Dans cet exemple, après que les noeuds aient effectués leurs modifications respectives, le mécanisme de synchronisation périodique de chaque noeud se déclenche. Le noeud A (resp. B) diffuse alors son état $\{a, b, c, e\}$ (resp. $\{a, d, e\}$) à B (resp. A).

À la réception de l'état, chaque noeud utilise la fonction `merge` pour intégrer les modifications de l'état reçu dans son propre état. Dans le cadre de l'Ensemble répliqué, cette fonction consiste généralement à faire l'union des états, en prenant en compte l'estampille et le statut (présent ou non) associé à chaque élément. Ainsi la fusion de leur état respectif, $\{a, b, c, e\} \cup \{a, d, e\}$, permet aux noeuds de converger à l'état souhaité : $\{a, b, c, d, e\}$.

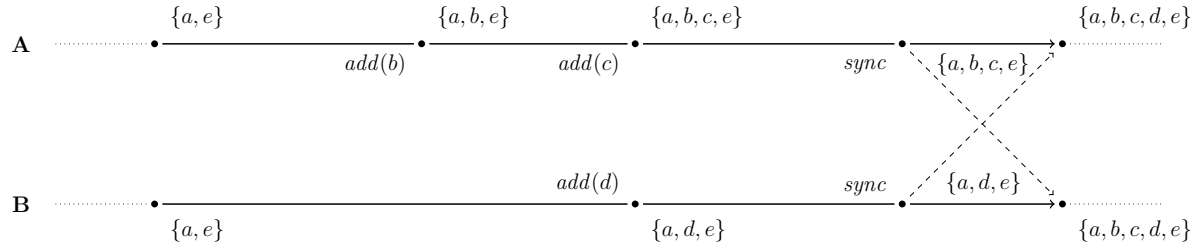


FIGURE 1.7 – Synchronisation des noeuds A et B en adoptant le modèle de synchronisation par états

Avant de conclure, il est intéressant de noter que ce modèle de synchronisation propose par nature une diffusion des modifications suivant le modèle de cohérence causale *Matthieu: TODO : Ajouter ref*. En effet, ce modèle de synchronisation assure une livraison soit de toutes les modifications connues d'un noeud, soit d'aucune. Par exemple, dans la Figure 1.7, le noeud B ne peut pas recevoir et intégrer l'élément c sans l'élement b. Ainsi, ce modèle permet naturellement d'éviter ce qui pourrait être interprétées comme des anomalies par les utilisateur-rices.

Synchronisation par opérations

L'approche de la synchronisation par opérations propose quant à elle que les noeuds diffusent leurs modifications sous la forme d'opérations. Pour chaque modification possible, les CRDTs synchronisés par opérations doivent définir deux fonctions : `prepare` et `effect`.

La fonction `prepare` a pour but de générer une opération correspondant à la modification effectuée, et commutative avec les potentielles opérations concurrentes. Elle prend en paramètres la modification ainsi que ses paramètres, et l'état courant du noeud. Cette fonction n'a pas d'effet de bord, c.-à-d. ne modifie pas l'état courant, et génère en retour l'opération à diffuser à l'ensemble des noeuds.

Une opération est un message. Son rôle est d'encoder la modification sous la forme d'un ou plusieurs éléments irréductibles du sup-demi-treillis.

Définition 3 (Élément irréductible) *Un élément irréductible d'un sup-demi-treillis est un élément atomique de ce dernier. Il ne peut être obtenu par la fusion d'autres états.*

Il est à noter que dans le cas des CRDTs purs synchronisés par opérations [8], les modifications labellisées avec leur information de causalité correspondent à des éléments irréductibles, c.-à-d. à des opérations. La fonction `prepare` peut donc être omise pour cette sous-catégorie de CRDTs synchronisés par opérations.

La fonction `effect` permet quant à elle d'intégrer les effets d'une opération générée ou reçue. Elle prend en paramètre l'état courant et l'opération, et retourne un nouvel état. Ce nouvel état correspond à la LUB entre l'état courant et le ou les éléments irréductibles encodés par l'opération.

La diffusion des modifications par le biais d'opérations présentent plusieurs avantages. Tout d'abord, la taille des opérations est généralement fixe et inférieure à la taille de

l'état complet du CRDT, puisque les opérations servent à encoder un de ses éléments irréductibles. Ensuite, l'expressivité des opérations permet de proposer plus simplement des algorithmes efficaces pour leur intégration par rapport aux modifications équivalentes dans les CRDTs synchronisés par états. Par exemple, la suppression d'un élément dans un Ensemble se traduit en une opération de manière presque littérale, tandis que pour les CRDTs synchronisés par états, c'est l'absence de l'élément dans l'état qui va rendre compte de la suppression effectuée. Ces avantages rendent possible la diffusion et l'intégration une à une des modifications et rendent ainsi utilisables les CRDTs synchronisés par opérations pour construire des systèmes temps réels.

Il est à noter que la seule contrainte imposée aux CRDTs synchronisés par opérations est que leurs opérations concurrentes soient commutatives. Ainsi, il n'existe aucune contrainte sur la commutativité des opérations liées causalement. De la même manière, aucune contrainte n'est définie sur l'idempotence des opérations. Ces libertés impliquent qu'il peut être nécessaire que les opérations soient délivrées au CRDT en respectant un ordre donné et en garantissant leur livraison en exactement une fois pour garantir la convergence. Ainsi, un intergiciel chargé de la diffusion et de la livraison des opérations est usuellement associé aux CRDTs synchronisés par opérations pour respecter ces contraintes.

Généralement, les CRDTs synchronisés par opérations sont présentés dans la littérature comme nécessitant une livraison causale des opérations. Ce modèle de livraison permet de respecter le modèle de cohérence causale et ainsi de simplifier raisonnement sur exécutions.

Ce modèle introduit néanmoins plusieurs effets négatifs. Tout d'abord, ce modèle peut provoquer un délai dans la diffusion des modifications. En effet, la perte d'une opération par le réseau provoque la mise en attente de la livraison des opérations suivantes. Les opérations mises en attente ne sont délivrées qu'une fois l'opération perdue re-diffusée et délivrée.

De plus, il nécessite que des informations de causalité précises soient attachées à chaque opération. Pour cela, les systèmes reposent généralement sur l'utilisation de vecteurs de version *Matthieu: TODO : Insérer ref*. Or, la taille de cette structure de données croît de manière linéaire avec le nombre de noeuds du système. Les métadonnées de causalité peuvent ainsi représenter la majorité des données diffusées sur le réseau [4]. Cependant, nous observons que la livraison dans l'ordre causal de toutes les opérations n'est pas toujours nécessaire. Par exemple, l'ordre d'intégration de deux opérations d'ajout d'éléments différents dans un Ensemble n'a pas d'importance. Nous pouvons alors nous affranchir de la livraison dans l'ordre causal pour accélérer la vitesse d'intégration des modifications et pour réduire les métadonnées envoyées.

Une autre contrainte généralement associée aux CRDTs synchronisés par opérations est la nécessité d'une livraison en exactement un exemplaire de chaque opération. Cette contrainte dérive de :

4. La relation de causalité étant transitive, les opérations et leurs relations de causalité forment un DAG. [61] propose d'ajouter en dépendances causales d'une opération seulement les opérations correspondant aux extremités du DAG au moment de sa génération. Ce mécanisme plus complexe permet de réduire la consommation réseau, mais induit un surcoût en calculs et en mémoire utilisée.

(i) La potentielle non-idempotence des opérations.

(ii) La nécessité de la livraison de chaque opération pour la livraison causale.

Toutefois, nous observons que des opérations peuvent être sémantiquement rendues obsolètes par d'autres opérations, e.g. une opération d'ajout d'un élément dans un Ensemble est rendue obsolète par une opération de suppression ultérieure du même élément. Ainsi, l'intergiciel de livraison peut se contenter d'assurer une livraison en un exemplaire au plus des opérations non-obsolètes. Ce choix permet de réduire la consommation réseau en évitant la diffusion d'opérations désormais non-pertinentes.

Pour compenser la perte d'opérations par le réseau et ainsi garantir la livraison à terme des opérations pertinentes, l'intergiciel de livraison des opérations doit mettre en place un mécanisme d'anti-entropie. Plusieurs mécanismes de ce type ont été proposés dans la littérature [58, 21, 40, 28] *Matthieu: TODO : Ajouter refs Scuttlebutt si applicable à Op-based* et proposent des compromis variés entre complexité en temps, complexité spatiale et consommation réseau.

Nous illustrons le modèle de synchronisation par opérations à l'aide de la Figure 1.8. Dans ce nouvel exemple, les noeuds diffusent les modifications qu'ils effectuent sous la forme d'opérations. Nous considèrons que le CRDT utilisé est un CRDT pur synchronisé par opérations, c.-à-d. que les modifications et opérations sont confondues.

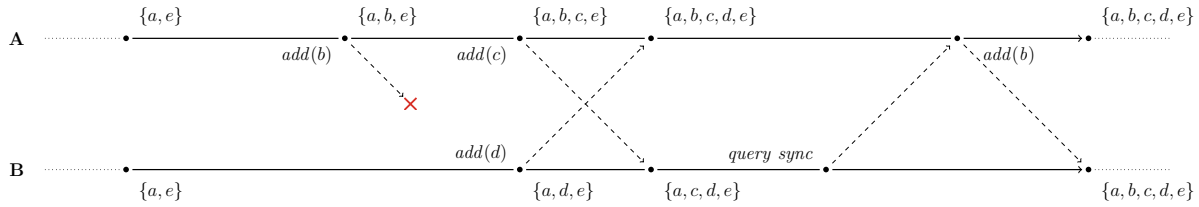


FIGURE 1.8 – Synchronisation des noeuds A et B en adoptant le modèle de synchronisation par opérations

Le noeud A diffuse donc les opérations $add(b)$ et $add(c)$. Il reçoit ensuite l'opération $add(d)$ de B, qu'il intègre à sa copie. Il obtient alors l'état $\{a, b, c, d, e\}$.

De son côté, le noeud B ne reçoit initialement pas l'opération $add(b)$ suite à une défaillance réseau. Il génère et diffuse $add(d)$ puis reçoit l'opération $add(c)$. Nous considérons que la livraisons des opérations *add* n'est pas obligatoire dans cet exemple, cette opération est alors intégrée. Le noeud B obtient alors l'état $\{a, c, d, e\}$.

Ensuite, le mécanisme d'anti-entropie du noeud B se déclenche. Le noeud B envoie alors à A une demande de synchronisation contenant un résumé de son état, e.g. son vecteur de version. À partir de cette donnée, le noeud A détermine que B n'a pas reçu l'opération $add(a)$. Il génère alors une réponse contenant cette opération et lui envoie. À la réception de l'opération, le noeud B l'intègre. Il obtient l'état $\{a, b, c, d, e\}$ et converge ainsi avec A.

Avant de conclure, nous noterons qu'il est nécessaire pour les noeuds de maintenir leur journal d'opérations. En effet, les noeuds l'utilisent pour renvoyer les opérations manquées lors de l'exécution du mécanisme d'anti-entropie évoqué ci-dessus. Ceci se traduit par une augmentation perpétuelle des métadonnées des CRDTs synchronisés par opérations. Pour

y pallier, des travaux [8, 11] proposent de tronquer le journal des opérations pour en supprimer les opérations connues de tous. Les noeuds reposent alors sur la notion de stabilité causale pour déterminer les opérations supprimables de manière sûre.

Définition 4 (Stabilité causale) *Une opération est stable causalement lorsqu'elle a été observée par l'ensemble des noeuds du système. Ainsi, toute opération future dépend causalement des opérations causalement stables, c.-à-d. les noeuds ne peuvent plus générer d'opérations concurrentes aux opérations causalement stables de manière honnête .*

Un mécanisme d'instantané *Matthieu: TODO : Ajouter refs* doit néanmoins être associé au mécanisme de troncage du journal pour générer un état équivalent à celui résultant des opérations supprimées. Ce mécanisme est en effet nécessaire pour permettre un nouveau noeud de rejoindre le système et d'obtenir l'état courant à partir de l'instantané et du journal tronqué.

Pour résumer, cette approche permet de mettre en place simplement un système fonctionnel à l'aide d'un CRDT synchronisé par opérations et d'un intergiciel de diffusion et de livraison RCB. Mais comme illustré ci-dessus, chaque CRDT synchronisé par opérations établit les propriétés de ses différentes opérations et délègue potentiellement des responsabilités à l'intergiciel de diffusion et livraison. La complexité de cette approche réside ainsi dans l'ajustement du couple $\langle CRDT, intergiciel \rangle$ pour régler finement et optimiser leur fonctionnement en tandem. Des approches [8, 11] ont été proposées ces dernières années pour concevoir et structurer plus proprement ces composants et leurs relations, mais reposent encore sur une livraison causale des opérations. *Matthieu: TODO : Vérifier que c'est bien le cas dans [11]*

Synchronisation par différences d'états

Dans [4], Almeida, Shoker et Baquero introduisent un nouveau modèle de synchronisation pour CRDTs. La proposition de ce modèle est nourrie par les observations suivantes :

(i) Les CRDTs synchronisés par opérations sont vulnérables aux défaillances du réseau et nécessitent généralement pour pallier à cette vulnérabilité une livraison des opérations en exactement un exemplaire et respectant l'ordre causal.

(ii) Les CRDTs synchronisés par états pâtissent du surcoût induit par la diffusion de leurs états complets, généralement croissant continuellement.

Pour pallier aux faiblesses de chaque approche et allier le meilleur des deux mondes, les auteurs proposent les CRDTs synchronisés par différences d'états [4, 3, 24]. Il s'agit en fait d'une sous-famille des CRDTs synchronisés par états. Ainsi, comme ces derniers, ils disposent d'une fonction `merge` permettant de produire la LUB entre deux états, et qui est associative, commutative et idempotente.

La spécificité des CRDTs synchronisés par différences d'états est qu'une modification locale produit en retour un delta. Un delta encode la modification effectuée sous la forme d'un état du lattice. Les deltas étant des états, ils peuvent être diffusés puis intégrés par les autres noeuds à l'aide de la fonction `merge`. Ceci permet de bénéficier des propriétés

d'associativité, de commutativité et d'idempotence offertes par cette fonction. Les CRDTs synchronisés par différences d'états offrent ainsi :

(i) Une diffusion des modifications avec un surcoût pour le réseau proche de celui des CRDTs synchronisés par opérations.

(ii) Une résistance aux défaillances réseaux similaire celle des CRDTs synchronisés par états.

Cette définition des CRDTs synchronisés par différences d'états, introduite dans [4, 3], fut ensuite précisée dans [24]. Dans cet article, Enes et al. précisent qu'utiliser des éléments irréductibles (cf. Définition 3) comme deltas est optimal du point de vue de la taille des deltas produits.

Concernant la diffusion des modifications, les CRDTs synchronisés par différences d'états autorisent un large éventail de possibilités. Par exemple, les deltas peuvent être diffusés et intégrés de manière indépendante. Une autre approche possible consiste à tirer avantage du fait que les deltas sont des états : il est possible d'agréger plusieurs deltas à l'aide la fonction `merge`, éliminant leurs éventuelles redondances. Ainsi, la fusion de deltas permet ensuite de diffuser un ensemble de modifications par le biais d'un seul et unique delta, minimal. Et en dernier recours, les CRDTs synchronisés par différences d'états peuvent adopter le même schéma de diffusion que les CRDTs synchronisés par états, c.-à-d. diffuser leur état complet de manière périodique. Plusieurs de ces approches sont évaluées et comparées de manière expérimentale dans [24].

Nous illustrons cette approche avec la Figure 1.9. Dans cet exemple, nous considérons que les noeuds adoptent la seconde approche évoquée, c.-à-d. que périodiquement les noeuds aggrégent les deltas issus de leurs modifications et diffusent le delta résultant.

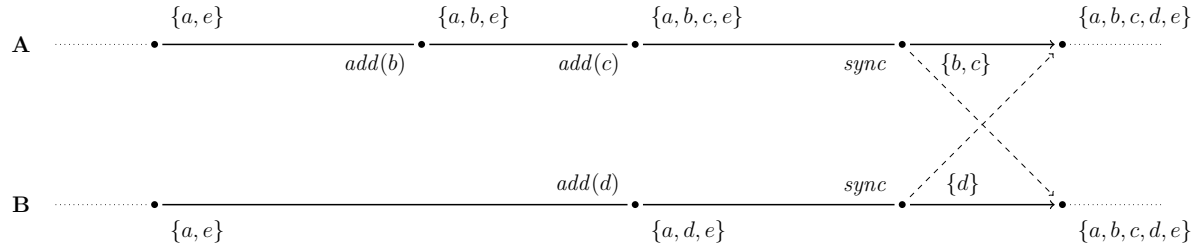


Figure 1.9 – Synchronisation des noeuds A et B en adoptant le modèle de synchronisation par différences d'états

Le noeud A effectue les modifications $add(b)$ et $add(c)$, qui retournent respectivement les deltas $\{b\}$ et $\{c\}$. Le noeud A aggrége ces deltas et diffuse donc le delta suivant b, c. Quant au noeud B, il effectue la modification $add(d)$ qui produit le delta $\{d\}$. S'agissant de son unique modification, il diffuse ce delta inchangé.

Quand A (resp. B) reçoit le delta $\{d\}$ (resp. $\{b, c\}$), il l'intègre à sa copie en utilisant la fonction `merge`. Les deux noeuds convergent alors à l'état $\{a, b, c, d, e\}$.

La synchronisation par différences d'états permet donc de réduire la taille des messages diffusés sur le réseau par rapport à la synchronisation par états. Cependant, il est important de noter que la décomposition en deltas entraîne la perte d'une des propriétés

intrinsèques des CRDTs synchronisés par états : le respect du modèle de cohérence causale. En effet, sans mécanisme supplémentaire, la perte ou le ré-ordonnement de deltas par le réseau peut mener à une livraison dans le désordre des modifications à l'un des noeuds. S'ils souhaitent assurer une diffusion causale des modifications, les CRDTs synchronisés par différences d'états doivent donc définir et intégrer dans leur spécification un mécanisme similaire à l'intergiciel de livraison des CRDTs synchronisés par opérations. En revanche et à l'instar des CRDTs synchronisés par opérations, les CRDTs synchronisés par différences d'états peuvent aussi faire le choix inverse : s'affranchir du modèle de cohérence causale pour accélerer la diffusion des modifications et minimiser le surcoût du type répliqué.

Ainsi, les CRDTs synchronisés par différences d'états sont une évolution prometteuse des CRDTs synchronisés par états. Ce modèle de synchronisation rend ces CRDTs utilisables dans les systèmes temps réels sans introduire de contraintes sur la fiabilité du réseau. Mais pour cela, il ajoute une couche supplémentaire de complexité à la spécification des CRDTs synchronisés par états, c.-à-d. le mécanisme dédié à la livraison des deltas, qui peut s'avérer source d'erreurs et de coûts supplémentaires.

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

	State-based	Op-based	Delta-based
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	✗	✓	✓

1.3.3 Adoption dans la littérature et l'industrie

Matthieu: TODO : Lister les CRDTs ayant été proposés. Registre, Ensemble, Sequence, Graphe, JSON, Filesystem, Access Control

— Conception et développement de librairies mettant à disposition des développeurs d'applications des types de données composés [52, 51, 77, 38, 7]

— Conception de langages de programmation intégrant des CRDTs comme types primitifs, destinés au développement d'applications distribuées [43, 20]

— 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 [64, 16, 75, 15, 79]

Bibliographie

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] Paulo Sérgio Almeida et al. « Scalable and Accurate Causality Tracking for Eventually Consistent Stores ». In : *Distributed Applications and Interoperable Systems*. Sous la dir. de Kostas Magoutis et Peter Pietzuch. Berlin, Heidelberg : Springer Berlin Heidelberg, 2014, p. 67–81. isbn : 978-3-662-43352-2.

[6] 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.

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

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

[9] 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.

[10] Carlos Baquero, Paulo Sérgio Almeida et Ali Shoker. « Making Operation-Based CRDTs Operation-Based ». In : *Distributed Applications and Interoperable Systems*. Sous la dir. de Kostas Magoutis et Peter Pietzuch. Berlin, Heidelberg : Springer Berlin Heidelberg, 2014, p. 126–140.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] Concordant. *Concordant*. url : http://www.concordant.io/.

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

[17] 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.

[18] Quang-Vinh Dang et Claudia-Lavinia Ignat. « Performance of real-time collaborative editors at large scale : User perspective ». In : *Internet of People Workshop, 2016 IFIP Networking Conference*. Proceedings of 2016 IFIP Networking Conference, Networking 2016 and Workshops. Vienna, Austria, mai 2016, p. 548–553. doi : 10.1109/IFIPNetworking.2016.7497258. url : https://hal.inria.fr/hal-01351229.

[19] 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.

[20] 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.

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

[22] 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.

[23] Victorien Elvinger, Gérald Oster et François Charoy. « Prunable Authenticated Log and Authenticable Snapshot in Distributed Collaborative Systems ». In : *2018 IEEE 4th International Conference on Collaboration and Internet Computing (CIC)*. IEEE. 2018, p. 156–165.

[24] 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.

[25] Etherpad. *Etherpad*. URL : https://etherpad.org/.

[26] Barton Gellman et Laura Poitras. *U.S., British intelligence mining data from nine U.S. Internet companies in broad secret program*. URL : https://www.washingtonpost.com/investigations/us-intelligence-mining-data-from-nine-us-internet-companies-in-broad-secret-program/2013/06/06/3a0c0da8-cebf-11e2-8845-d970ccb04497_story.html.

[27] Jim Giles. « Special Report Internet encyclopaedias go head to head ». In : *nature* 438.15 (2005), p. 900–901.

[28] 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.

[29] Google. *Google Docs*. URL : https://docs.google.com/.

[30] Glen Greenwald et Ewen MacAskill. *NSA Prism program taps in to user data of Apple, Google and others*. URL : https://www.theguardian.com/world/2013/jun/06/us-tech-giants-nsa-data.

[31] 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.

[32] 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.

[33] Claudia-Lavinia IGNAT. « Maintaining consistency in collaboration over hierarchical documents ». Thèse de doct. ETH Zurich, 2006.

[34] Claudia-Lavinia IGNAT et al. « How Do User Groups Cope with Delay in Real-Time Collaborative Note Taking ». In : *European Conference on Computer Supported Cooperative Work 2015*. Sous la dir. de Nina BOULUS-RODJE et al. Proceedings of the 14th European Conference on Computer Supported Cooperative Work. Oslo, Norway : Springer International Publishing, sept. 2015, p. 223–242. DOI : 10.1007/978-3-319-20499-4_12. URL : https://hal.inria.fr/hal-01238831.

[35] Claudia-Lavinia IGNAT et al. « Studying the Effect of Delay on Group Performance in Collaborative Editing ». In : *Proceedings of 11th International Conference on Cooperative Design, Visualization, and Engineering, CDVE 2014, Springer 2014 Lecture Notes in Computer Science*. Proceedings of 11th International Conference on Cooperative Design, Visualization, and Engineering, CDVE 2014. Seattle, WA, United States, sept. 2014, p. 191–198. DOI : 10.1007/978-3-319-10831-5_29. URL : https://hal.archives-ouvertes.fr/hal-01088815.

[36] Paul R JOHNSON et Robert THOMAS. *RFC0677 : Maintenance of duplicate databases*. RFC Editor, 1975.

[37] 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.

[38] 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.

[39] 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.

[40] 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.

[41] 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.

[42] 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.

[43] 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.

[44] Madhavan MUKUND, Gautham SHENOY et SP SURESH. « Optimized or-sets without ordering constraints ». In : *International Conference on Distributed Computing and Networking*. Springer. 2014, p. 227–241.

[45] 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.

[46] Brice NÉDELEC, Pascal MOLLI et Achour MOSTÉFAOUI. « A scalable sequence encoding for collaborative editing ». In : *Concurrency and Computation : Practice and Experience* (), e4108. DOI : 10.1002/cpe.4108. eprint : https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.4108. URL : https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.4108.

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

[48] Brice NÉDELEC et al. « LSEQ : an adaptive structure for sequences in distributed collaborative editing ». In : *Proceedings of the 2013 ACM Symposium on Document Engineering*. DocEng 2013. Sept. 2013, p. 37–46. DOI : 10.1145/2494266.2494278.

[49] 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.

[50] 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.

[51] 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.

[52] 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.

[53] Matthieu Nicolas. « Efficient renaming in CRDTs ». In : *Middleware 2018 - 19th ACM/IFIP International Middleware Conference (Doctoral Symposium)*. Rennes, France, déc. 2018. URL : https://hal.inria.fr/hal-01932552.

[54] Matthieu Nicolas, Gérald Oster et Olivier Perrin. « Efficient Renaming in Sequence CRDTs ». In : *7th Workshop on Principles and Practice of Consistency for Distributed Data (PaPoC'20)*. Heraklion, Greece, avr. 2020. URL : https://hal.inria.fr/hal-02526724.

[55] 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.

[56] Sylvie Noël et Jean-Marc Robert. « Empirical study on collaborative writing : What do co-authors do, use, and like ? » In : *Computer Supported Cooperative Work (CSCW)* 13.1 (2004), p. 63–89.

[57] 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.

[58] 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.

[59] Jim Pick. *Graf, Nikolaus*. URL : https://www.serenity.re/en/notes.

[60] Jim Pick. *PeerPad*. URL : https://peerpad.net/.

[61] 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.

[62] 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.

[63] 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.

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

[65] 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.

[66] 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.

[67] 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.

[68] 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.

[69] Haifeng Shen et Chengzheng Sun. « A log compression algorithm for operation-based version control systems ». In : *Proceedings 26th Annual International Computer Software and Applications*. 2002, p. 867–872. DOI : 10.1109/CMPSAC.2002.1045115.

[70] 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.

[71] Douglas B Terry et al. « Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System ». In : *SIGOPS Oper. Syst. Rev.* 29.5 (déc. 1995), p. 172–182. ISSN : 0163-5980. DOI : 10.1145/224057.224070. URL : https://doi.org/10.1145/224057.224070.

[72] 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.

[73] 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.

[74] 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.

[75] 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.

[76] 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.

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

[78] 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.

[79] 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.

[80] 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.

[81] 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.