Rapport de Stage Master 2 FSI

Critères de Cohérence faible byzantine appliquée aux environnements cloud

JOLY Amaury
Encadrants : Godard Emmanuel, Travers Corentin

Aix-Marseille Université, Scille

7 septembre 2023

Résumé

J'ai réalisé ce stage au Laboratoire d'Informatique et Système sous l'encadrement d'Emmanuel Godard et Corentin Travers.

L'objectif à étais de fournir les bases théoriques à la réalisation d'une thèse CIFRE en collaboration entre l'entreprise Scille et le laboratoire. La thèse à pour but d'étudier les critères de cohérence faible et d'appliquer ces recherches au développement d'une application permettant l'édition collaborative sur des documents textes sans intermédiaire de confiance.

Table des matières

1 Présentation du Sujet

Le stage a était réalisé dans le cadre d'une collaboration entre le LIS et l'entreprise Scille SAS. Il a pour objectif la réalisation d'un travail d'état de l'art d'un sujet au vu de son traitement dans une thèse ultérieur.

1.1 Présentation de l'entreprise

Scille[8] est une startup Française d'une dizaine d'employés localisée à Bordeaux et travaillant majoritairement dans une politique de télétravail. Elle développe le produit Parsec, qui est une solution reposant sur une architecture client-serveur hébergée dans le cloud, et visant à fournir un service d'hébergement et de partage de fichier. Parsec se démarque de ses concurrents, en proposant une solution répondant à un haut niveau de sécurité et en adoptant une approche en source ouverte, le rendant ainsi facilement auditable pour leurs clients. Ainsi le projet s'encre dans une conception dite de zero-trust en proposant un chiffrement de bout en bout des données, rendant ainsi le serveur incapable de consulter le contenu des données qu'il stocke. Son rôle se limite donc à celui de "relai" entre les clients ainsi que de support de stockage pour le chiffré des données.

Parsec répond aux critères nécessaires à l'obtention de la certification ANSSI-CSPN-2021/08[9], avec pour objectif de toucher un marché d'entreprise national ayant un besoin d'échanger des fichiers répondant à des prérequis importants en termes de confidentialités tel que pour la santé, le juridique ou encore l'industrie et le militaire.

1.2 Présentation du Sujet

Scille souhaite développer le produit Parsec de manière à rajouter un service de suite bureautique au service principale qui est le partage de fichier. À ce titre leur premier besoin est la réalisation d'un outil de traitement de texte.

L'objectif est de fournir une interface permettant l'édition des fichiers présents dans un espace de travail Parsec dans une approche collaborative et en conservant les propriétés de sécurité de l'application principale. C'est-à-dire en restant dans une approche zero-trust et donc chiffré de bout en bout.

Ce projet fait l'objet d'une proposition de thèse émise par Scille et encadrée par le Laboratoire d'Informatique et Système. C'est dans ce cadre que c'est effectué mon stage, qui à pour objectif principal la réalisation d'un état de l'art sur ces questions. Et comme objectif secondaire mon intégration à l'organisme finançant ma potentielle thèse.

1.3 Bref résumé du stage

Mon stage a débuté en avril 2023. J'ai commencé par étudier les pistes fournit par mes encadrants, dont majoritairement le livre de Mathieu Perrin *Concurrence et Cohérence dans les Systèmes repartis* [7]. Suite à ça j'ai réalisé des recherches sur des sujets connexes, en jonglant ainsi entre lecture de la littérature scientifique et réalisation de présentations orales. Me permettant ainsi d'affiner ma compréhension et de rendre compte de mes avancements à mes encadrants.

Une fois une maitrise satisfaisante du sujet obtenu, j'ai pu commencer à essayer d'affiner le sujet potentiel de la thèse, et de réfléchir à une solution pour la problématique de Scille.

Ainsi j'ai pu étudier les preuves de concepts réalisés par les ingénieurs en internes à Scille, ainsi qu'aborder la faisabilité en identifiants les outils et algorithmes existants pouvant aider à la réalisation d'une telle application.

2 Travail accomplit

2.1 L'étude de la cohérence des données dans les applications distribuée

La première étape de la réalisation de mon stage a était la lecture et compréhension de la littérature mise en lien avec le sujet du stage. Le sujet sous-jacent à ces différentes sources peut être résumé à l'étude de la **cohérence dans des systèmes répartis en milieu malicieux**.

Il semble essentiel dans un premier temps d'expliquer cet énoncé et de faire le lien avec notre projet plus concret.

En tant qu'application collaborative en temps réel, notre problème doit impliquer une application distribuée. C'est-à-dire une suite de procédure sous la forme d'un algorithme qui pourra être exécutés par l'ensemble des membres collaborant et qui aura comme objectif la synchronisation des informations entre les utilisateurs.

Ce genre d'application nécessite de manière intrinsèque une architecture dite distribuée. On formalisera ainsi un système distribué comme étant un ensemble de nœud capable d'échanger de l'information les uns avec les autres au vu de réaliser une tâche commune.

Les applications collaboratives grand public actuelles (Google Doc [2], Office365 [6], OnlyOffice [10]) utilisent une approche de synchronisation entre les utilisateurs qui imite un comportement similaire à une exécution sans collaboration. Elles utilisent ainsi une architecture client-serveur. Chaque opération d'écriture et de lecture du document soumise par le client doit être validé en amont par le serveur qui s'assurera de la préservation de la cohérence du document entre chaque membre à chaque instant. En d'autres termes avec cette approche si on prend une image de l'état des données stocké par l'ensemble du réseau, alors elle sera identique pour chaque nœud.

Cette approche est la plus évidente et est la plus proche d'un comportement attendu par un utilisateur sur ce genre d'usage. Mais elle présente tout de même quelques problèmes. Premièrement elle nécessite que tous les membres possèdent une latence raisonnable, afin de permettre la meilleure interactivité possible. L'ensemble des nœuds du système doivent se baser sur la latence du nœud le plus lent, nuisant à l'interactivité de l'application. Pour éviter cela, les applications exclues les membres possédants des connexions trop lentes limitant donc les utilisateurs potentiels.

Cette approche n'est pas non plus compatible avec une réplication du serveur sans occasionner une baisse de l'interactivité par la même occasion. Posant ainsi le problème d'un point de faille unique où si le serveur se trouve être inaccessible, l'application ne peut plus être fonctionnel. Par extension dans le cas d'une segmentation du réseau, les membres n'étant pas sur le segment commun avec le serveur se retrouve exclues de l'application.

La dernière limitation de cette approche et qu'elle implique que le serveur central ait accès aux données partagé et en soit l'ordonnanceur, ce qui rentre complétement en désaccord avec l'approche zero-trust de Parsec. Cette solution n'est donc pas acceptable, heureusement des compromis sont possibles autorisant des comportements qui serait inacceptable dans un contexte de synchronicité forte entre les nœuds, mais ne nuisant pas nécessairement à notre application finale. Afin d'étudier les différents compromis possibles nous devons donc nous pencher sur l'étude de la cohérence dans les systèmes répartie.

Nous nous intéressons à la notion de milieu malicieux, c'est-à-dire considérer que n'importe quel nœud du réseau peut se révéler être dissident de la procédure attendue à des fin rationnelle ou non, puisque dans notre contexte, l'application sera distribuée dans un environnement de production et doit donc à ce titre prendre en considération ce genre de cas.

2.1.1 Comment étudier la cohérence dans un système répartie

L'étude de la cohérence des données dans un système répartie est à différencié de l'étude algorithmique de l'application distribuée sous-jacente. Ce qui nous intéresse dans l'étude de la cohérence est la manière dont les états locaux des différents nœuds évolues dans le temps et non pas la manière avec laquelle nous avons obtenu ces états locaux. Cette nuance est importante puisqu'elle nous force à regarder les changements d'états comme une histoire résultant d'une certaine exécution d'un algorithme sans être biaisé par les spécifications

de ce dernier. On ne se base donc plus sur ce que l'algorithme est censé produire, mais sur l'histoire que nous observons.

Pour observer et décrire ces histoires, nous avons besoin d'un modèle assez simple pour pouvoir être facilement généralisé tout en étant capable d'expliquer la totalité de ce qui peut être observable. Nous utiliserons ainsi la modélisation utiliser par Perrin [7] dans son ouvrage précédemment cité. PERRIN définit différents types de données mettant plus ou moins en avant certains comportements. Dans le cadre de ce rapport, nous utiliserons essentiellement l'"ensemble" qui se rapproche le plus de notre objectif final.

L'ensemble peut être soumis à 3 opérations différentes :

— L'insertion notée $I(v)$ permet d'insérer l'élément v dans l'ensemble.

— La suppression notée $D(v)$ permet de retirer l'élément v de l'ensemble.

— La lecture notée R retourne l'état de l'ensemble.

Par exemple imaginons un système à un seul nœud où les opérations $I(0) \bullet I(1) \bullet I(2) \bullet R \bullet D(1) \bullet R$ sont réalisées. Nous pouvons modéliser cette exécution comme vue dans la figure 1.

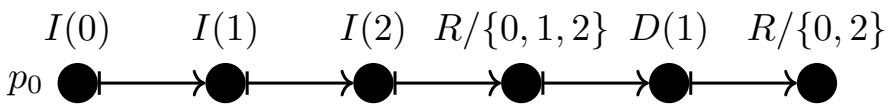

FIGURE 1 – Exemple d'histoire concurrente à un seul nœud sur un ensemble

Il est important de noter que dans un ensemble, il n'y a pas de notion d'ordre. Si un élément est inséré plusieurs fois dans l'ensemble alors il ne sera présent qu'une unique fois au maximum et ne nécessitera qu'une seule opération de suppression pour être retiré.

Nous pouvons, à partir de la figure 1 retrouver la liste des opérations qui nous on servit à la produire, c'est-à-dire $I(0) \bullet I(1) \bullet I(2) \bullet R \bullet D(1) \bullet R$. Cette liste d'opération est ce qu'on appelle une linéarisation de l'histoire décrite par la figure. Dans le contexte de l'étude de la cohérence d'un système on ne prend pas une linéarisation existante pour produire une histoire, mais on observe de manière extérieure une histoire et on essaye par la suite de montrer si une linéarisation est possible.

Reprenons notre exemple en y ajoutant un second nœud.

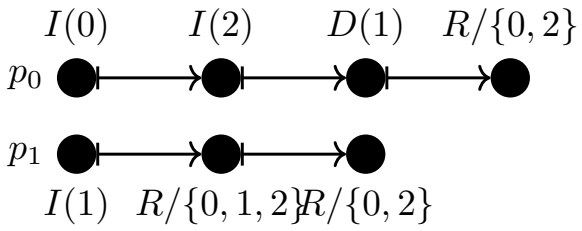

FIGURE 2 – Exemple d'histoire concurrente à 2 nœuds sur un ensemble

Dans la figure 2 nous avons répartie les opérations entre plusieurs nœuds. C'est le comportement qu'on pourrait observer dans une application d'édition collaborative par exemple. Mais ici il semble moins évident d'extraire une linéarisation.

La linéarisation la plus évidente pourrait être montrée sur la figure 3.

Ici nous imaginons que ce qui à pu produire une telle histoire est l'exécution séquentielle des opérations $I(0) \bullet I(1) \bullet I(2) \bullet R \bullet D(1) \bullet R$. Ce qui ne diffère donc en rien d'une exécution à un seul nœud. La cohérence est entièrement préservée et on ne distingue pas cette exécution d'une exécution similaire réalisée par un unique nœud.

On parle alors de cohérence forte, ou de cohérence séquentielle. Puisque la linéarisation est indiscernable de celle d'une exécution séquentielle.

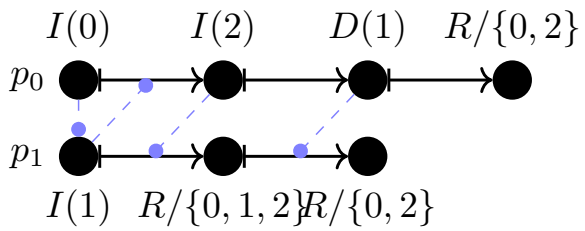

FIGURE 3 – Exemple de linéarisation possible

Néanmoins, il est aussi envisageable que l'exécution se soit déroulée comme ce qui est montrée dans la figure 4.

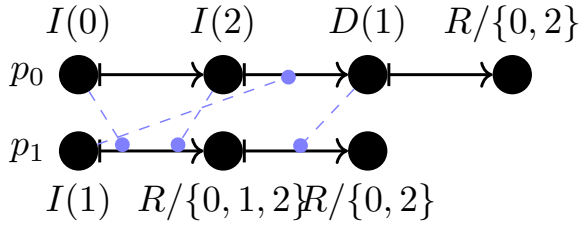

FIGURE 4 – Exemple de linéarisation possible

Ici chaque nœud possède sa propre linéarisation. Du point de vue du nœud 0 l'exécution est linéarisé comme suit $I(0) \bullet I(2) \bullet I(1) \bullet D(1) \bullet R$. Tandis que pour le nœud 1 il s'est plutôt passé l'exécution suivante $I(1) \bullet I(0) \bullet I(2) \bullet R \bullet D(1) \bullet R$. Ici le résultat final ne change pas, mais les évolutions des états de chaque nœud diverge à certains moments. Dans le cas où l'exécution se serait terminé prématurément, les deux nœuds auraient fini leur collaboration dans des états différents.

L'histoire présentée en figure 2 ne nous permet pas de trancher. Par défaut l'approche est optimistes et nous considérons que cette histoire satisfait le critère de cohérence séquentielle puisque étant le critère le plus haut applicable à cette histoire.

2.1.2 Les critères de cohérences faibles

La cohérence séquentielle est le nom donné au critère qui définit qu'une histoire concurrente soit indiscernable de son équivalent non concurrent. C'est le critère de cohérence le plus fort, celui privilégier par les applications d'édition collaborative grand public.

La cohérence séquentielle a était découverte au début de l'informatique distribuée par Lamport [4, 5] lorsqu'il travaillait pour Intel sur les premiers processeurs multicœur. À l'époque les processeurs atteignait une limite en termes de précision de gravure. Et est naturellement venu l'idée de mettre plusieurs cœurs dans une même puce afin de répartir la tâche et de gagner en performance. Il a fallu réfléchir au comportement qu'on attendait d'une telle architecture afin d'éviter au maximum les conflits et la perte de temps lié à la résolution de problèmes de synchronisation des cœurs entre eux.

La cohérence séquentielle peut être découpée en 3 propriétés que sont : la validité, la localité d'état et la convergence. On dit que la cohérence séquentielle satisfait ces trois propriétés, on la considère ainsi comme étant un critère de cohérence fort. Les critères de cohérences ne satisfaisant qu'un seul ou deux de ces propriétés sont donc qualifiés de cohérence faible, proposant des compromis diminuant la cohérence d'une application, mais présentant des avantages sur l'efficacité algorithmique.

La convergence La convergence est la première propriété que nous allons définir. Elle est illustrée par la figure 5.

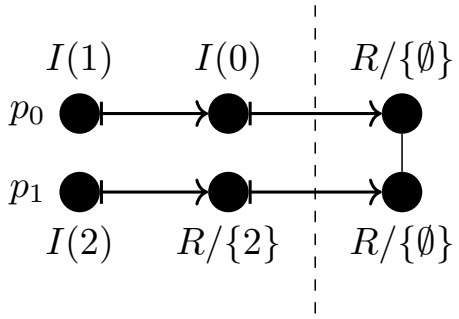

FIGURE 5 – Exemple d'histoire convergente

La convergence se concentre seulement sur l'état final de l'histoire concurrente sans jamais prendre en compte les opérations précédentes. Pour qu'une histoire satisfasse le critère de convergence, elle doit admettre un état unique justifiant les lectures finales d'un système.

En d'autres termes, si tous les membres du réseau arrêtent d'écrire et réalisent en boucle sur une durée infinie des opérations de lecture. Alors toutes les opérations de lectures doivent fournir un résultat identique dépendant d'un seul et même état.

Dans la figure 5 on constate que les lectures finales ne prennent pas en compte les opérations précédentes. Ce qui se traduit du point de vue du nœud 0 par une non prise en compte de ses soumissions, et du point de vue du nœud 1 comme un retour en arrière de son état après la soumission d'un événement. Mais aucune opération de suppression n'a jamais était émise. Ainsi il n'existe aucune linéarisation possible permettant de justifier cet état final.

Si nous voulions le traduire de manière concrète à la problématique d'un éditeur collaboratif, cela reviendrait en un document qui se verrait modifier dans un premier temps, puis sur lequel ces mêmes modifications serait supprimées sans raison apparente pour l'utilisateur. Cela ne permet donc pas de garantir une cohérence du point de vue de l'expérience utilisateur, mais seulement une cohérence sur l'état final des données.

La Localité d'état Là où la convergence assure la cohérence des données à la fin de l'exécution, la localité d'état assure la cohérence du point de vue du nœud.

On le formalise donc comme l'existence pour chaque nœud d'une linéarisation incluant l'ensemble des opérations de lecture de ce nœud ainsi qu'un sous-ensemble des écritures du système.

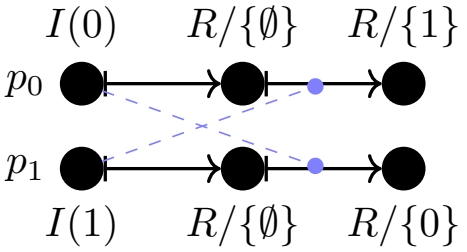

FIGURE 6 – Exemple d'histoire respectant la localité d'état

Dans le cas de la figure 6 on observe bel et bien une histoire qui respecte la localité d'état. Les lectures du nœud 0 peuvent être linéarisées tel que $R/\{\emptyset\} \bullet I(1) \bullet R/\{1\}$. Il en est de même pour le nœud 1 via la linéarisation $R/\{\emptyset\} \bullet I(0) \bullet R/\{0\}$.

Ce qui nous interpelle ici, c'est que nous avons omis les écritures locales à chaque nœud afin de préserver la cohérence des lectures. Dans la localité d'état, il est plus important que les observations de l'utilisateur soient cohérentes, plutôt que de conserver les actions réalisées par ce dernier. Les informations peuvent très bien être divergente in fine puisque ce qui compte, c'est la cohérence de l'expérience et non pas celle du système.

En appliquant le critère à notre problématique d'éditeur collaboratif. Cela reviendrait à un document qui aura toujours du sens du point de vue de l'utilisateur, mais risquant de diverger entre les différents nœuds. Créant ainsi une ramification des versions du document. On peut l'assimiler au fonctionnement d'un dépôt git mettant à jour seulement les fichiers n'ayant jamais était édité par l'utilisateur, s'affranchissant ainsi de tout conflit à régler qui viendrait perturber la cohérence locale.

La Validité La dernière propriété à être présentée est la Validité. Elle assure que les lectures finales résultent d'une linéarisation de l'ensemble des opérations d'écritures.

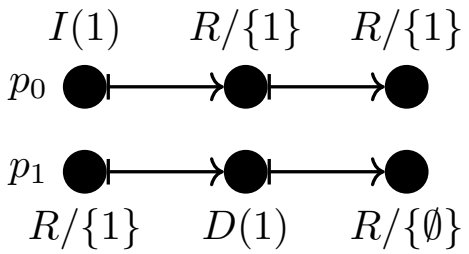

FIGURE 7 – Exemple d'histoire respectant la validité

Dans la figure 7 cela se traduit par la lecture $R/\{1\}$ linéarisable par $D/(1) \bullet I(1) \bullet R/\{1\}$ bien que cet ordre n'est pas cohérent au vu de la première lecture du nœud 1. En effet, elle force à croire que l'insertion est déjà prise en compte avant même que l'opération de suppression ne soit soumise.

La validité ne peut donc pas permettre la préservation d'une cohérence locale ou convergente. Mais elle assure que le résultat est cohérent dépendamment des opérations d'écritures émises dans le système.

Dans notre contexte imaginons que deux utilisateurs modifie la même partie du document en local. Si leurs travaux vise à être fusionnés, alors il risque d'y avoir deux fois la même ligne, ou bien un entrelacement des deux lignes. Mais en respectant la validité on assure qu'en aucun cas l'utilisateur se verra supprimer ou refuser son opération par l'algorithme.

2.2 Quelle direction pour notre sujet ?

Perrin [7] propose une cartographie des différents critères de cohérences connus. Elle prend la forme d'une carte en deux dimensions comportant 3 zones s'entrelaçant comme montré dans la figure 8. Chaque zone correspondant à une propriété définit plus haut.

L'intersection de ces 3 zones correspond donc à la cohérence forte. Et le reste du schéma à des critères de cohérence faibles.

Pour savoir où chercher le critère de cohérence nécessaire à la résolution de notre problématique, nous devons déterminer quelles sont les propriétés nécessaires à la réalisation d'un bon éditeur collaboratif.

Ainsi la convergence semble être une propriété essentielle pour notre produit. En effet, on attend d'un bon éditeur de document qu'il nous permette de réaliser, une fois les sessions d'éditions terminées, un document unique et non pas plusieurs versions divergentes.

Pour rester dans la cohérence faible, nous devons au choix respecter soit la validité, soit la localité d'état.

Dans le premier cas, nous prenons parti de tolérer des remaniements de l'histoire lors des synchronisations provoquant le non-respect de la cohérence du document du point de vue de l'utilisateur. Cela risque de

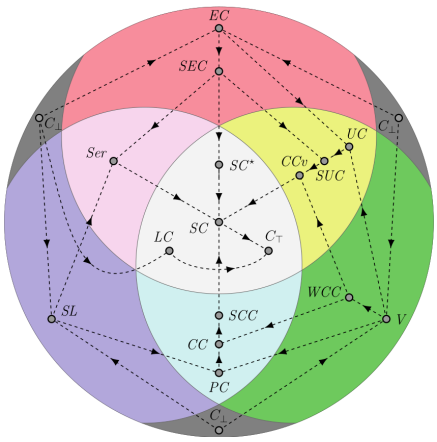

FIGURE 8 – Cartographie des critères de cohérences par Mathieu Perrin

créer des insertions et suppressions arbitraires et des annulations d'opérations pourtant bien retourner à l'utilisateur. L'expérience semblera incohérente et illogique pour ce dernier.

Dans le second on autorise le système à réaliser des opérations n'ayant jamais était émise ou omettre d'autres opérations afin de préserver la cohérence locale. Dégradant par la même occasion l'expérience de l'utilisateur en lui donnant une sensation de "retour en arrière" sur ses opérations.

2.3 Accepter de la non-convergence avec les CRDTs

Une autre solution est néanmoins possible, en utilisant un type de donnée particulier. En effet, il est possible d'accepter une non-convergence des états si le type de données que nous échangeons permet déjà d'assurer une convergence. C'est possible en utilisant des CRDTs que nous allons définir.

Les CRDTs (Conflict-free Replicated Data Types) [11] sont des structures de données conçues pour fonctionner efficacement dans des environnements distribués, où plusieurs répliques des données peuvent être modifiées simultanément. Ils garantissent la convergence des données répliquées sans nécessiter d'ordonnanceur intermédiaire.

Le principe fondamental des CRDTs repose sur la notion d'opérations commutatives et associatives. En d'autres termes, les opérations sur les données répliquées peuvent être effectuées dans n'importe quel ordre sans entraîner de conflits ou de résultats incohérents. Cela permet aux applications utilisant des CRDTs de soumettre des opérations concurrentes sans jamais avoir à ne résoudre de conflit.

Les CRDTs peuvent nous permettre dans notre cas, de s'affranchir du critère de convergence et ainsi privilégier la validité et la localité d'état qui sont les deux impactants l'expérience de l'utilisateur d'un point de vue local.

2.3.1 Quel est donc le critère le plus adapté ?

En étudiant la cartographie de Perrin (figure 8), il semble que les critères les plus pertinents sont la cohérence pipeline (PC), la cohérence causale (CC) et la cohérence causale forte (SCC). Au moment où j'écris ce rapport je n'ai pas encore déterminé lequel de ces 3 critères est le plus pertinent pour notre problématique. Néanmoins, nous pouvons présenter la cohérence pipeline qui est la base des deux autres critères, ce qui donne une idée d'à quoi un système respectant à la fois la Validité et la Localité d'état peut ressembler.

Cohérence Pipeline La cohérence pipeline fonctionne sur une logique du "premier arrivé premier servit" comme illustré dans la figure 9.

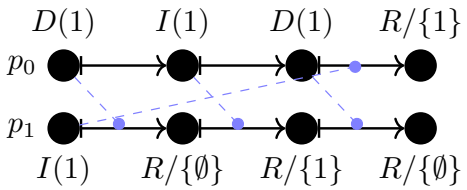

FIGURE 9 – Exemple d'histoire respectant la cohérence pipeline

Ce critère ne garantit en effet pas la convergence, puisque les opérations peuvent arriver dans des ordres différents d'un nœud à l'autre. C'est problématique dans le cas d'un ensemble pouvant être soumis à des insertions et des suppressions comme ici. Mais dans une optique d'utilisation avec des CRDTs il n'y a pas nécessité à retirer des éléments. Notre ensemble stockera simplement un ensemble de CRDTs qui, peu importe leurs ordres d'arrivés reproduira systématiquement la même histoire. Ainsi nous aurions une exécution plus proche de celle de la figure 10.

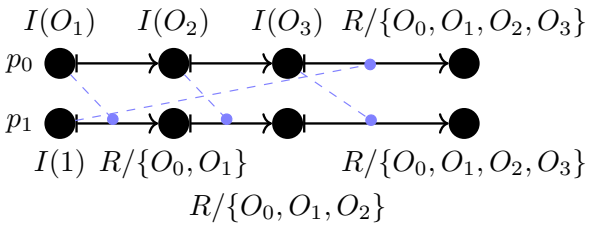

FIGURE 10 – Exemple d'histoire respectant la cohérence pipeline utilisant des CRDTs

2.4 Appliquer un système reparti à un environnement malicieux

Une partie de mon travail à étais d'étudier le lien entre l'acceptation de critère de cohérences faibles et les risques de comportements malicieux.

J'ai donc formulé à ce moment-là deux hypothèses possibles :

Dans la première le fait de baisser le critère de cohérence ouvre la porte à plus de risques de comportement malicieux. Cette hypothèse semble être celle la plus étudiée. [12, 3] Mais les études sur le sujet semble concentrer les comportements à des implémentations satisfaisant les critères de cohérences et non pas au critère lui-même. Rendant ainsi difficile l'attribution de comportements malicieux à un critère de cohérence particulier. Il est compliqué de discerner si ces comportements ne relèvent pas simplement d'une erreur protocolaire dans l'implémentation.

La deuxième hypothèse consiste à dire que puisque le critère de cohérence baisse, on accepte plus de situations qui serait considérés problématiques et malicieuse en temps normal. Ce qui rend par cette occasion un système à faible cohérence plus robuste. Je n'ai pas trouvé de littérature appuyant particulièrement cette seconde hypothèse.

2.5 Les outils et pistes pour le développement d'un produit final

Le développement d'un éditeur collaboratif sans autorité de contrôle implique nécessairement la mise en place d'un réseau en pair à pair. Je me suis donc posé la question de la faisabilité réelle d'un réseau P2P à travers Internet dans le contexte d'une entreprise.

Le frein principal est la présence de NAT du côté du fournisseur d'accès puis de l'entreprise. Dans des systèmes d'information complexe il peut même y avoir plusieurs profondeurs de NAT au sein d'une seule organisation. Le NAT (Network Address Translation) à pour but de faire une translation d'adresse

entre l'adresse publique de l'entreprise, routable via Internet, avec l'adresse privé de la machine, routable seulement sur le réseau local de l'entreprise. Cette couche rend impossible pour une personne extérieure au réseau de l'entreprise de contacter un client cacher derrière un NAT sans que celui-ci soit configurer à cet usage.

Ce fonctionnement est particulièrement problématique dans le contexte d'une application en pair à pair. La solution qui fait néanmoins consensus est d'adopter la technique du hôle-punching. [1] Le hôle-punching consiste en un détournement du fonctionnement du protocole UDP afin de réaliser une connexion en pair à pair.

Pour ce faire nous avons besoin d'un serveur exposé sur Internet et connu des pairs. Lorsqu'un des nœuds source veut entrer en contact avec un nœud cible, il demande au serveur la mise en relation avec un autre membre du réseau. Le serveur envoie donc l'adresse IP et le port du NAT derrière lequel se cache le nœud cible.

Dans le même temps, le serveur prévient le nœud cible de l'arrivée du nœud source. Une fois que les deux nœuds possèdent mutuellement l'adresse de l'un et de l'autre. Ils initient chacun auprès de leurs NAT une connexion l'un vers l'autre. Le NAT du nœud source s'attend donc bien à recevoir un message du NAT du nœud cible et vice-verse.

Cette technique nous permet donc d'envisager le fonctionnement d'un tel système en pair à pair dans un contexte réel d'entreprise.

La seule limitation qui pourrait se retrouver en désaccord avec la politique zero-trust de parsec est la mise en place d'un serveur dit de "rendez-vous" qui devrait gérer un annuaire des adresses IP de chaque nœud du réseau. Mais ce sont les seules informations auquel le serveur à accès puisque le reste de l'échange se réalise directement en pair à pair. Le serveur Parsec possédant déjà l'information de l'adresse IP de par les logs d'accès au stockage de fichier, cette solution ne nécessite aucune information supplémentaire de la part du serveur Parsec. Elle reste donc compatible avec les primitives de sécurités et de confidentialités liées à Parsec.

3 Acquis

3.1 Sécurité des Systèmes d'Information

Ce stage à étais pour moi l'opportunité de m'intéresser à la science de l'informatique appliquée aux systèmes distribuée. J'ai pu ainsi découvrir comment mesurer la fiabilité et la cohérence dans ces systèmes, tout en en appréciant les enjeux. J'ai ainsi pu développer une sensibilité dans ma manière d'aborder les applications distribuée, me permettant d'avoir un regard plus critique sur les choix de conceptions de ces derniers.

Dans le même temps, mes contacts avec l'équipe de Scille m'ont permis de découvrir les enjeux du développement à un stade précoce d'une application orientée sécurité. Mon intégration à ce projet m'a permis de développer un réflexe critique dans la manière dont j'envisage la conception de l'éditeur collaboratif. Devant à tout pris le rendre cohérent avec le produit sur lequel il vise à être greffé.

Cela à étais aussi l'occasion pour moi de découvrir des types d'attaques particuliers et propres aux systèmes distribués, ainsi que des techniques permettant de contourner la sécurité de réseaux dans le but de réaliser des communications en pair à pair.

3.2 Compétences transverses

Durant ce stage, j'ai pu perfectionner ma maitrise du LaTeX et notamment la modélisation de schémas. J'ai eu à de nombreuses occasions l'opportunité de m'exprimer devant un auditoire ce qui m'a énormément apporté en aisance à l'oral et en maitrise de mon sujet.

Ayant travaillé seul en autonomie la plupart de mon temps, j'ai aussi du mettre en place une certaine rigueur et organisation autour de mon travail. Et trouver un équilibre dans l'organisation de mes tâches et de mes pauses.

4 Bilan

Mon sentiment vis-à-vis de ce stage est globalement positif. J'ai pu comprendre de manière plus profonde le fonctionnement et l'organisation de la recherche en Informatique ce qui était pour moi l'objectif principal.

Je me suis pris de passion pour le sujet que j'ai eu à traiter et j'ai hâte de poursuivre mon travail dans le cadre de la thèse.

Mon intégration avec Scille est aussi très positive et je les remercie énormément pour la confiance qu'ils me portent et l'importance du rôle qu'ils me donnent. J'ai réellement le sentiment de faire partie de l'équipe et du projet et c'est quelque chose de très gratifiant.

Mes regrets vis-à-vis de ce stage porte plutôt sur le début durant lequel j'ai dû me construire un rythme qui n'a pas était évident à trouver. Les premières semaines on était éprouvante au point de remettre en question mon engagement dans cette voie. Il m'a fallu m'approprier le sujet et gérer d'une meilleure manière mes sessions de travail avant d'arriver à m'épanouir dans ce projet.

J'aurais aussi aimé apporter une contribution plus importante à Parsec. Je trouve ne pas avoir eu le temps d'acquérir une maitrise du produit suffisante pour être pleinement capable d'y projeter mon travail.

Je suis très reconnaissant envers mes encadrants pour leurs présences et leurs bienveillances à mon égard. Ils m'on était d'une aide capitale autant vis-à-vis de mon travail que d'un point de vue personnel et c'est avec joie et confiance que je m'engage dans une thèse avec eux.

Références

[1] Bryan Ford, Pyda Srisuresh et Dan Kegel. "Peer-to-Peer Communication Across Network Address Translators." In : *USENIX Annual Technical Conference, General Track*. 2005, p. 179-192.

[2] Google. `https://docs.google.com`.

[3] Saptaparni Kumar. "Fault-Tolerant Distributed Services in Message-Passing Systems". Thèse de doct. Texas A&M University, 2019.

[4] Lamport. "How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs". In : *IEEE Transactions on Computers* C-28.9 (sept. 1979). Conference Name : IEEE Transactions on Computers, p. 690-691. ISSN : 1557-9956. DOI : `10.1109/TC.1979.1675439`.

[5] Leslie Lamport. "On interprocess communication". In : *Distributed Computing* 1.2 (1er juin 1986), p. 86-101. ISSN : 1432-0452. DOI : `10.1007/BF01786228`. URL : `https://doi.org/10.1007/BF01786228` (visité le 08/06/2023).

[6] Microsoft. `https://office.com`.

[7] Matthieu Perrin. *Concurrence et cohérence dans les systèmes répartis*. Google-Books-ID : 6DRlDwAAQBAJ. ISTE Group, 1er sept. 2017. 194 p. ISBN : 978-1-78405-295-9.

[8] Scille. `https://parsec.cloud`.

[9] Scille. `https://parsec.cloud/wp-content/uploads/2022/11/20210421-Certification-ANSSI-CSPN-2021-08-PARSEC-2.0.0.pdf`.

[10] Scille. `https://onlyoffice.com`.

[11] Marc Shapiro et al. "Conflict-free replicated data types". In : *Stabilization, Safety, and Security of Distributed Systems : 13th International Symposium, SSS 2011, Grenoble, France, October 10-12, 2011. Proceedings 13*. Springer. 2011, p. 386-400.

[12] Albert Van Der Linde, João Leitão et Nuno Preguiça. "Practical client-side replication : weak consistency semantics for insecure settings". In : *Proceedings of the VLDB Endowment* 13.12 (août 2020), p. 2590-2605. ISSN : 2150-8097. DOI : `10.14778/3407790.3407847`. URL : `https://dl.acm.org/doi/10.14778/3407790.3407847` (visité le 06/06/2023).

Table des figures