Université de Montpellier
Master 1 Informatique Génie Logiciel

Un algorithme d'exclusion mutuelle :

Algorithme de Maekawa

Rapport du projet

Etudiants :
JULIAT Thomas
GARCIA Léa

Table des matières

1 Notre approche de l'algorithme

1.1 Introduction

Pour ce projet d'algorithmes distribués nous avons choisi d'implémenter l'algorithme de Maekawa. C'est un algorithme d'exclusion mutuelle basé sur l'utilisation de quorums. Un quorum étant un sous ensemble de sites parmi tous les sites présents. Dans cet algorithme, chaque site souhaitant entrer en section critique doit faire parvenir sa demande à tous les sites de son quorum et ne peut y accéder qu'une fois qu'il a obtenu toutes les autorisations de la part de ces sites. Afin de casser la symétrie on utilise le principe de l'estampille que l'on ajoute sur les messages, de cette manière on peut gérer la priorité des messages tout en ayant des sites de même importance.

1.2 Lancement de notre projet

Notre projet est composé d'un makefile qui crée deux exécutables : "proc" et "site", et supprime les fichiers objets une fois que tout est compilé. En ligne de commande il suffit donc de faire "make" pour compiler le projet sur votre machine.

Ensuite, nous avons écrit un script bash pour faciliter le lancement du projet et l'observation. Le script prend en argument le nombre de clients attendus par le processus de départ, le port sur lequel le processus de départ va se lancer ainsi que son IP (pour les fournir aux sites clients). Après avoir fait "make" on lance le script avec par exemple : "bash script.bash 9 127.0.0.1 1258".

D'abord le script s'occupe de lancer le processus mère avec l'exécutable "proc" et en précisant le nombre de clients attendus et le port sur lequel il va se lancer. Ensuite avec l'exécutable "site", le script crée 9 terminaux avec l'émulateur de terminaux, leur donne l'IP ainsi que le port du processus mère auxquels ils doivent se connecter. Le script place les 9 xterm sur l'écran de manière à ce que les terminaux soient placés comme ils le sont dans la matrice au moment de la création des quorums comme nous le verrons plus tard. Ainsi, les messages s'échangent et s'observent à la verticale et à l'horizontale, nous expliquerons plus en détails tout cela dans la partie structures de données.

L'utilisateur peut entrer des demandes d'entrées en section critique sur les sites qu'il souhaite en ligne de commande. Il peut également faire la simulation d'un retard en lançant d'abord une demande puis un retard sur un autre site, ce retard va être desservi avant, sauf si le premier site a obtenu tous ses accords (et récupéré son propre accord) et peut donc rentrer en section critique

1.3 Complexité de l'algorithme

Lorsqu'un site fait une demande d'entrée en section critique il envoie autant de messages "DEMANDE" que la taille de son quorum -1. Lorsque chaque site lui envoie son "ACCORD", il y a autant de messages que la taille du quorum -1 qui circulent de nouveau. Enfin, lors de la libération il y a toujours autant de messages pour permettre de libérer tous les sites avec le message "LIBERATION". Pour une utilisation classique de la section critique il y a donc '3 x (tailleQuorum(site) - 1)' messages échangés. Si par contre il y a une demande plus prioritaire alors il y aura autant de messages "SONDAGE" et "RESTITUTION" que la taille du quorum -1, qui seront envoyés afin de récupérer le jeton en faveur d'un autre site et donc là il y aurait '5 x (tailleQuorum(site) - 1)' messages échangés.

2 Les structures de données choisies

Pour la construction du projet, de nombreux choix sur les différentes structures de données ont dû être faits : le choix de la structure du message contenant le quorum de chaque site, de la file d'attente, des messages d'échanges entre les sites et de l'estampillage. Dans ce chapitre, nous allons voir la raison de ces choix et ce que cela implique. Une petite précision : nous parlons de "jeton d'accord" mais c'est "l'accord" d'un site qui est donné à un autre site et le fait qu'il se "verrouille" pour cet autre site ; il n'y a pas réellement de jeton.

2.1 Message contenant le Quorum

Le message envoyé par le serveur mère à tous les sites contenant leur quorum respectif commence par un identifiant de tel sorte que le site receveur connaisse sont identifiant dans le réseau pour qu'il puisse signer certains messages qu'il veut envoyer à d'autres sites. Le message est suivi d'une longue chaîne de caractères contenant les éléments d'un tableau de structure, chaque élément du tableau est séparé par le caractère " :" et chaque attribut d'un élément du tableau est séparé par le caractère ",". Ainsi, lors de la réception du message, le site peut décortiquer la longue chaîne de caractères pour reconstruire un tableau identique à celui qui lui a été envoyé. On procède à une sérialisation du côté serveur mère et à une désérialisation du côté du site.

2.2 File d'attente

Pour le choix de la structure de la file d'attente nécessaire à chaque site, nous avons opté pour le choix d'une liste chaînée, ainsi il est possible d'appliquer 4 opérations simples sur cette liste chaînée qui sont nécessaires au fonctionnement de la liste d'attente. Récupérer le premier élément de la file d'attente pour lui donner son jeton lors de la libération d'un site. Supprimer le premier élément de la file d'attente lorsque cet élément a reçu le jeton pour rentrer en section critique. L'ajout d'un élément en fin de liste pour ajouter un élément qui a fait une demande d'entrer en section critique mais que le jeton du site receveur de cette demande n'est toujours pas disponible. L'ajout d'un élément en début de liste pour mettre l'élément qui vient de restituer le jeton en cas de sondage en premier dans la liste d'attente ou encore le site qui a une demande prioritaire ou encore un site qui se met lui même en attente pour s'accorder son jeton lorsqu'il le récupérera...

2.3 Message d'échange entre les sites

Pour la communication entre sites, nous avons opté pour l'envoi d'une chaîne de caractères contenant plusieurs informations séparées par le caractère " :". La chaîne de caractère est composée du nom de la requête (DEMANDE, ACCORD, etc...) et optionnellement de l'estampille et/ou de l'identifiant du site envoyeur. Ces deux informations sont optionnelles car elles ne sont utiles que pour certains messages, par exemple l'estampille n'est utile que pour la requête DEMANDE.

2.4 Estampillage

Pour pouvoir utiliser l'estampillage de façon simple en C, nous avons choisi de concaténer dans cet ordre, l'année, le mois, le jour, les heures, les minutes, les secondes et les millisecondes. Grâce à cette

concaténation, nous obtenons un très grand nombre qui peut être comparé avec d'autres estampilles créées de la même façon. Avec cette méthode la temporalité des messages est respectée.

2.5 Matrice pour la construction des quorum

Pour pouvoir construire "facilement" nos quorums tout en respectant les règles, nous avons choisi de travailler avec une matrice carrée de taille n. Le processus de départ attend un nombre carré de sites donc 4, 9, 16, 25, 36 sites... Pour connaître le quorum d'un site c'est très simple, on prend tous les sites qui sont sur sa ligne et tous les sites qui sont sur sa colonne. De cette manière on respecte les règles de l'algorithme : Le site i est dans le quorum i (si on leur donne un numéro), la taille du quorum d'un site est égale à K et est identique pour chaque site, un site est contenu dans D quorums différents et ici K = D.

3 Nos choix d'architecture et de modélisation

Notre algorithme de Maekawa s'exécute sur chaque site. Cependant, avant de dérouler le vrai algorithme, chaque site doit connaître son quorum et pour cela, fait appel au processus de départ aussi appelé processus mère.

3.1 Le processus de départ (processusDepart.c)

Afin d'obtenir son quorum, un site doit se connecter à notre processus de départ, que l'on exécute avec l'exécutable "proc" sur un site dédié uniquement à l'initialisation des quorums.

Le processus de départ lance la fonction 'fonctionDepart' à laquelle on fournit le nombre de sites qu'il faut attendre pour construire les quorums. Notre fonction de départ va attendre avec une boucle 'while', que tous les sites soient arrivés. Quand un site se connecte, notre fonction l'accepte, elle se met en réception et attend un message contenant le port du site qui vient de se connecter, elle le stocke puis récupère également son adresse IP. La fonction construit une structure 'accueilClient' et y ajoute toutes ces valeurs ainsi que le numéro de la socket de communication qui a été ouverte spécialement pour ce site, pour ensuite pouvoir lui envoyer son quorum. Ensuite, la fonction ajoute la structure 'accueilClient' de ce nouveau site à la matrice à deux dimensions et attend un nouveau site.

Lorsque tous les sites attendus sont arrivés et que leurs structures associées sont remplies et rangées dans la matrice, la fonction s'occupe, pour chaque site tour à tour, de construire un tableau 'tabQuorum' contenant les structures 'accueilClient' des sites de son quorum. Pour cela elle prend un site (deux boucles 'for' imbriquées qui parcourent la matrice), et se met à parcourir sa ligne (une boucle 'for') puis sa colonne (une boucle 'for') et récupère chaque structure en l'ajoutant à un tableau 'tabQuorum'.

Ensuite une fonction 'seralizationOfArray' s'occupe de construire un message à partir du tableau 'tabQuorum' qui contient toutes les informations nécessaires. La fonction se connecte au site dont la sérialization du tableau est le quorum ; lui transmet d'abord la taille de son quorum, puis lui transmet le message construit. Le site en question est évidemment prêt à recevoir les deux messages, dans le bon ordre et de la bonne longueur.

3.2 Le main (site.c)

Afin d'obtenir son quorum, un site doit se connecter à notre processus de départ, que l'on exécute avec l'exécutable "proc" sur un site dédié uniquement à l'initialisation des quorums. Dans le fichier site.c, nous avons choisi de mettre une fonction main qui s'occupe de faire s'exécuter l'algorithme de Maekawa. Lorsque le script lance l'exécutable "site" sur chaque site, il lui fournit le numéro de port du processus de départ ainsi que son adresse IP.

La fonction main commence par construire tout ce dont les threads qu'elle va lancer auront besoin, c'est à dire qu'elle ouvre deux sockets, une socket 'sockServeur' et une socket 'sockClient'. Ensuite elle ouvre une troisième socket qu'elle va dédier à la connexion au serveur de départ que nous avons vu juste avant. Elle se connecte à ce site, lui envoie son port qu'elle a récupéré car celui-ci a été attribué automatiquement par le système. Puis elle se met en réception et attend qu'on lui envoie la taille de son quorum et son quorum, ces opérations sont bloquantes et permettent d'attendre car il faut que tous les sites attendus se soient aussi connectés au serveur mère.

Lorsque le message contenant toutes les informations nécessaires pour connaître son quorum arrive sur le site, le main s'occupe de découper le message. Il récupère la taille du quorum, l'identifiant du site courant, construit un tableau contenant le quorum du site avec chaque case du tableau contenant une structure 'site', contenant l'identifiant, l'adresse IP et le port du site. Le main s'occupe ensuite de remplir les structures de variables partagées par les deux futurs threads, avec toutes les informations dont on dispose et celles que l'on a récupérées. Le main initialise le mutex et la variable conditionnelle qui seront utilisés plus tard. Enfin, il lance deux threads, un premier thread qui exécute la fonction 'fonctionReception' et un deuxième thread qui exécute la fonction 'fonctionDemande'. Il attend bien évidemment les deux threads avec la fonction 'pthread_join' avant de terminer le programme.

3.3 La fonction d'écoute

Le premier thread lancé est un thread d'écoute, il s'occupe d'exécuter la fonction 'fonctionReception'. Cette fonction est définie dans le fichier 'fonctionsServeur.c' et permet de traiter les réceptions de notre site.

La fonction 'fonctionReception' permet de recevoir via la socket d'écoute nommée 'sockServeur' stockée dans la variable partagée qui a été donnée en paramètre au thread, les différents messages envoyés par les différents sites du quorum. Cette fonction commence par mettre en écoute la socket serveur avec la fonction 'listen' en permettant à 'n' clients de se connecter, ce 'n' étant égal à la taille du quorum définie. Ensuite cette fonction lance une une boucle while(1) qui accepte un client avec la fonction 'accept', reçoit un message de sa part avec le 'recv', décrypte le message avec 'strtok', récupère le texte du message, récupère l'identifiant du site s'il est fournit, récupère l'estampille si elle est fournie puis en fonction du texte grâce à la fonction 'strcmp', effectue un certain traitement :

Les différents texte de messages possibles :

DEMANDE Ce message signifie que le site qui l'a envoyé, souhaite entrer en section critique. Trois cas sont possibles :

Si j'ai mon jeton d'accord (cas où je ne l'ai donné à aucun site) et que ma liste d'attente est vide alors je peux me "verrouiller" pour ce site demandeur en mettant la valeur de la variable 'varPart->jetonDaccord' à 0 et en lui envoyant un message de type "ACCORD". Je me dois de remplir le champ 'varPart->idSiteEnSectionCritique' avec l'identifiant du site demandeur et le champ 'varPart->prioCurrentProcessus' avec l'estampille reçue dans le message, afin que je puisse les utiliser si besoin.

Si je n'ai pas mon jeton d'accord (cas où je l'ai donné à un site) et que l'estampille du message qui vient d'être reçu est plus grande que celle du site pour lequel je me suis verrouillé précédemment, alors ce site demandeur n'est pas prioritaire, je le mets dans ma file d'attente et je lui envoie un message "ATTENTE" pour lui signifier que je le bloque pour l'instant.

Si je n'ai pas mon jeton et que l'estampille du message qui vient d'être reçu est plus petite que celle du site pour lequel je me suis verrouillé précédemment alors ce site demandeur est prioritaire ! Dans ce cas, il y a deux possibilités : Soit je ne suis pas le site qui a fait la demande moins prioritaire et alors j'envoie un message "SONDAGE" au site pour lequel je me suis verrouillé et je mets le site qui est le plus prioritaire en tête de ma file d'attente afin de le faire passer devant toutes les autres demandes qui n'étaient pas prioritaires au moment où j'ai comparé. Nous verrons ce qu'implique ce message de sondage par la suite, dans notre cas, on l'envoie dans le but de récupérer le jeton d'accord que l'on a donné.

Deuxième possibilité, je suis le site qui a fait la demande la moins prioritaire et je n'ai pas reçu l'accord du site qui a fait la demande plus prioritaire que la mienne. Dans ce cas là, je me déverrouille en mettant ma variable 'varPart->jetonDaccord' jeton à 0 car je donne mon jeton, je m'insère à la tête de ma file d'attente (les autres étaient en file d'attente car moins prioritaire sur moi), je remplis les champs nécessaires et surtout, j'envoie un message "ACCORD" au site plus prioritaire que moi. Ce cas n'est possible que si le site plus prioritaire n'avait pas donné son accord car il n'aurait jamais donné son accord s'il avait déjà lui même envoyé une demande.

ACCORD Ce message signifie qu'un site de mon quorum m'accorde son accord pour entrer en section critique et pour l'instant, je suis le plus prioritaire pour lui. Je viens de recevoir un accord, je vais donc verrouiller mon mutex, augmenter la variable 'varPart->v.nbAccords' de 1 et tester si j'ai reçu le bon nombre d'accords et si je possède mon jeton, afin de pouvoir entrer en section. Ce nombre d'accords doit être égal à la taille de mon quorum. Si j'ai le nombre suffisant d'accords alors je mets ma variable 'varPart->jetonDaccord' jeton à 0 pour ne plus donner mon jeton, j'augmente le nombre d'accords à 'tailleQuorum' + 1 pour signifier que je suis actuellement en section critique. Puis je fais appel à la fonction 'pthread_cond_signal' qui réveille le thread endormi sur la condition 'varPart->v.entreeSC' et je déverrouille le mutex associé. C'est tout pour le traitement côté écoute, le reste est fait par la fonction de demande contenue dans le fichier 'fonctionsClient.c'.

ATTENTE Ce message signifie qu'un site de mon quorum, auquel j'ai fait une demande d'accord, a déjà donné son accord à un autre site et ce site a fait une demande plus prioritaire que la mienne. Je suis donc mis en attente jusqu'à ce que l'autre site sorte de la section critique, envoie des messages de type "LIBERATION" et m'envoie son jeton (pas forcément à moi en premier).

SONDAGE Ce message signifie qu'un site de mon quorum qui m'a accordé son accord, a reçu une demande plus prioritaire que la mienne. Plus prioritaire signifiant qu'elle a été effectuée avant la mienne mais n'a pas pu être traitée, peut-être à cause de lenteur de messages. Deux cas sont possibles : Soit je n'ai pas reçu le bon nombre d'accords et je n'ai pas encore pu rentrer en section critique au moment du sondage et dans ce cas je restitue le jeton d'accord au site qui me le demande. Je lui envoie un message "RESTITUTION" et je décrémente le nombre d'accords reçus avec la variable 'varPart->v.nbAccords'. Soit j'ai déjà reçu le bon nombre d'accords et je suis déjà en section critique et je ne la quitte pas car même si ma demande n'était pas prioritaire, la section critique est occupée ce n'est donc pas du gâchis de temps, la demande plus prioritaire sera traitée juste après. Le test de 'être en section critique' est fait avec le nombre d'accords qui doit être égal à 'tailleQuorum' + 1 comme on vient de le voir (cela ne signifie rien c'est un choix arbitraire de notre part).

RESTITUTION Ce message signifie que le site auquel j'ai envoyé une demande de sondage afin de récupérer mon accord, n'était pas entré en section critique et a pu me rendre mon accord. Je vais donc pouvoir l'envoyer au site plus prioritaire. J'envoie un message de type "ACCORD" au premier site de ma file d'attente et je le supprime de la file d'attente pour mettre ses valeurs dans les variables 'varPart->prioCurrentProcessus' et 'varPart->idSiteEnSectionCritique' car je me verrouille pour lui et je veux pouvoir récupérer ces valeurs si besoin. Enfin, j'insère le site qui m'a envoyé le message de restitution en tête de ma file d'attente

LIBÉRATION Ce message signifie qu'un site de mon quorum à qui j'avais donné mon accord, est sorti de la section critique et me redonne mon accord. Trois cas sont possibles :

Soit la file d'attente n'est pas vide et l'identifiant en tête c'est mon identifiant. Dans ce cas c'est que je suis le suivant, je récupère donc mon jeton pour ma propre demande.

Soit la file d'attente n'est pas vide et le premier élément est un autre identifiant que le mien, je dois donc traiter la demande de ce site. Je lui envoie un message de type "ACCORD", je me verrouille pour lui de la même manière que vu jusqu'à présent, je le supprime de ma file d'attente car je l'ai traité et j'indique que c'est ce site qui est en section critique (pour moi) en remplissant la variable 'varPart->idSiteEnSectionCritique' avec son id.

Soit la file d'attente est vide, dans ce cas je récupère mon jeton, en mettant la variable 'varPart->jetonDaccord' à 1.

3.4 La fonction de demande

Le deuxième thread lancé est un thread d'envoi, il s'occupe d'exécuter la fonction 'fonctionDemande'. Cette fonction est définie dans le fichier 'fonctionsClient.c' et permet de traiter les demandes qui sont faites par l'utilisateur pour ce site.

La fonction 'fonctionDemande' est composée d'une boucle 'while' qui attend des entrées de la part de l'utilisateur. La boucle permet deux entrées différentes : 'demande' ou 'retard'. L'option 'demande' permet de faire une simple demande d'entrée en section critique pour le site. L'option 'retard' est une simulation d'une demande ancienne qui n'a pas été desservie et qui va être prioritaire au moment où elle sera reçue par les sites de son quorum. Cependant, afin d'être réaliste, cette demande en retard ne peut être envoyée que si notre site n'a donné son accord à personne car il n'aurait jamais donné son accord si une demande était déjà présente sur le site. La boucle 'while' s'assure donc que l'option 'retard' ne fonctionne que si le site ne s'est pas verrouillé au profit d'un autre.

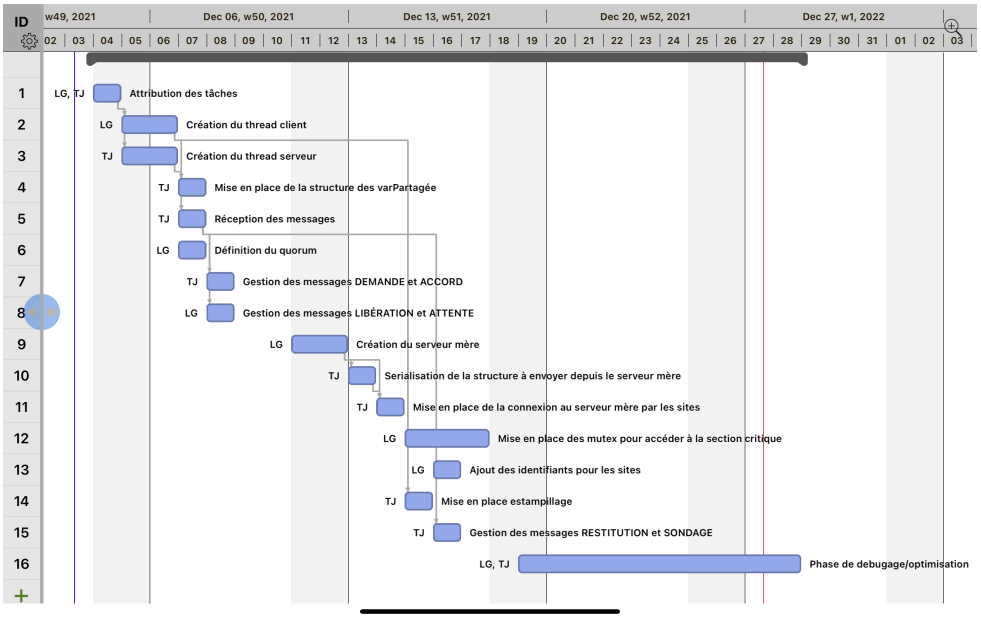
Dans les deux cas, 'demande' ou 'retard', le code est assez similaire. Si il s'agit d'une demande, la fonction insère la demande dans la liste, si c'est un retard, elle l'insère en tête car cette demande va être prioritaire on va le voir au moment de l'estampille. La fonction crée donc l'estampille comme nous l'avons vu dans le chapitre 2 et c'est ici que se trouve la différence entre les deux options. Si l'utilisateur choisi l'option 'demande' alors l'estampille est récupérée en fonction du temps courant, si c'est 'retard' alors l'estampille est créée avec une date fictive égale à "20201214111217". On aurait pu mettre une date presque égale mais tout juste antérieure cependant cela ne change rien à notre exemple, ce n'est pas réaliste mais permet de bien observer les comparaisons d'estampilles au moment de la réception d'un message de demande d'entrée en section critique sur le thread de réception (message de type "DEMANDE").

Par la suite, la fonction construit un message de type "DEMANDE" avec l'identifiant et l'estampille associés, elle se connecte grâce à la socket dans sa variable partagée (qui a été donnée en paramètre au thread), à chaque site de son quorum, dont les informations nécessaires sont contenues dans le tableau 'varPart->quorum[]' que nous avons vu précédemment. Une fois tous ces envois effectuées, la fonction verrouille le verrou 'varPart->v.lock', le même verrou qui est verrouillé lors d'une réception d'un message "ACCORD". Ensuite la fonction effectue un 'wait' sur la condition 'varPart->v.entreeSC' et le verrou qu'elle a verrouillé, elle relâche le verrou à ce niveau et met le thread en attente d'être réveillé par un signal venant de l'autre thread qui signifiera que la condition a été remplie.

Lorsque ce thread va être réveillé par un signal, il pourra verrouiller le verrou et continuer l'exécution de son code et entrer en section critique, on symbolise ceci par deux affichages pour l'entrée et la sortie ainsi qu'une fonction qui affiche des points et fait un calcul entre chaque affichage, afin de simuler un travail fait en section critique. Une fois sorti, le thread déverrouille le mutex, met le nombre d'accords reçu à , supprime la tête de sa file si c'est sa demande qui était positionnée, puis s'occupe d'envoyer un message de type "LIBERATION" à tous les sites de son quorum qui lui avaient donné leurs accords. Ensuite le thread consulte la file d'attente du site qui a pu être remplie en parallèle par le thread d'écoute. Si la file d'attente n'est pas vide alors un message de type "ACCORD" est envoyé au site ayant comme id celui qui est rangé en tête de la file. Si la file est vide alors le site récupère son jeton d'accord.

4 L'organisation du projet

4.1 Diagramme de Gantt



Au début du projet nous nous sommes attribués chacun une partie du site (thread client pour Léa et thread serveur pour Thomas) pour pouvoir travailler chacun sur une partie différente du projet. Après quelques jours nous avons réalisé que la plus grosse partie du travail se trouvait dans la partie serveur de l'application, nous nous sommes alors mis finalement à deux sur cette partie. La répartition des tâches s'est donc faite sur le traitement des différents types de messages possibles que la partie serveur peut recevoir. En plein milieu du projet nous avons décidé de stopper le développement du programme principal pour développer un serveur mère qui a pour but de créer dynamiquement des quorums pour tous les sites du réseau. Cet arrêt était essentiel pour ne pas continuer sur de mauvaises bases car avant cet ajout, la création des quorums se faisait en dur dans le code ce qui est une très mauvaise pratique de programmation. Ayant fini notre projet en avance nous avons décidé de nous plonger dans une grande phase de tests afin de déceler et de corriger toutes les erreurs potentielles de notre part.

4.2 Notre GitHub

Pour la gestion de versionnage de notre projet, nous avons décidé d'utiliser GitHub. Cela nous a permis d'avancer indépendamment l'un de l'autre sur le projet. Une politique de commit a été mis en place : un commit doit être réalisé à chaque ajout/modification d'une fonctionnalité pour éviter de lourd commit en fin de session de travail, cela permet des retours sur version stable en cas de détection de bugs problématiques. Tout commit doit contenir du code fini qui n'empêche pas une exécution basique du projet, cette règle a été mis en place pour éviter de bloquer un camarade voulant travailler sur le projet en cours de route.

5 Conclusion

Après la réalisation du projet, nous sommes satisfaits de celui-ci. Pour une mise en fonctionnement réelle sur une ressource partagée, il suffirait de remplacer la fonction "calcul" par un accès en mémoire à une ressource partagée.

Ce projet nous a permis de comprendre la complexité et le comportement de cet algorithme. Les deux problèmes qui subsistent sont : La construction des quorums et la gestion des priorités dans les listes. En effet, avec notre algorithme, il n'est possible de construire que des réseaux de tailles carrées (4, 9, 16, 25, etc...). Il est tout de même possible de créer des réseaux avec moins de sites en utilisant des pseudo-sites qui ne sont là que pour donner des jetons et qui n'accèdent jamais en section critique. Ensuite, concernant la gestion des priorités dans les listes nous aurions pu facilement traiter ce problème mais nous avions peur que cela ne corresponde plus au principe initial de l'algorithme de Maekawa, cela conduit donc parfois à des inter-blocages dans certains cas rares.

Notre longue phase de débug et d'optimisation nous a permis de remarquer de nombreux problèmes sur des situations précises que nous avons résolues et qui nous ont permis de bien comprendre l'algorithme et ses subtilités. La bonne répartition des tâches pour la réalisation de notre projet nous a permis d'être efficaces et de passer du temps sur nos tests et sur l'observation du déroulement de l'algorithme.