

PROJET ARA

Encadrants: **LEJEUNE Jonathan & MAHAMDI Célia**
BENCHABANE Mohand & UNG Richard
M2 SAR

Table des matières

1	Introduction	2
2	Modélisation du problème	3
2.1	Le protocole Paxos	3
2.1.1	Principes du protocole	3
2.1.2	Notion de leader	4
2.2	Adaptation de Paxos à notre modèle	5
2.2.1	Rôle des nœuds	5
2.2.2	Élection du leader	5
2.2.3	Surveiller le leader	5
2.3	Étude expérimentale 1 : itération d'élection	6
2.3.1	Paramétrage de l'expérience	6
2.3.2	Conclusion de l'expérience	13
2.4	Étude expérimentale 2 : Multi-Paxos séquentiel	14
2.4.1	Paramétrage de l'expérience	14
2.4.2	Conclusion de l'expérience	21

1 Introduction

Un service informatique distant peut être interrogé par des clients via des requêtes. On peut distinguer deux types de requêtes :

- les requêtes dites "de lecture" qui ne modifient pas l'état courant du service.
- les requêtes dites "d'écriture" qui peuvent modifier l'état courant du service.

Le déploiement de ce service se fait dans un environnement potentiellement large échelle (beaucoup de clients pouvant amener le service à ne pas pouvoir honorer les requêtes en temps raisonnable suite à une surcharge) et non fiable, aussi bien au niveau des nœuds de calcul (ici le serveur hébergeant le service) qu'au niveau des liens de communication (les messages entre les clients et le serveur). Ces caractéristiques réalistes peuvent amener à une indisponibilité du service (pas de réponse, forte latence, résultats incohérents, ...). Afin de pouvoir garantir un minimum de disponibilité, il est courant de répliquer ce service. En effet, les requêtes de lecture peuvent requêter directement un réplica ce qui permet une bonne répartition de charge et une transparence des pannes. En revanche, les requêtes d'écriture sont problématiques, car elles remettent en cause la cohérence des états des différents réplicas. L'assurance de cohérence entre les réplicas dans un tel environnement non fiable rend la réplication comme un problème fondamental des systèmes distribués. Pour garder un état cohérent sur les réplicas, il est nécessaire que ces derniers reçoivent les requêtes d'écriture dans le même ordre. On utilise alors un mécanisme de réplication de machine à états qui se base sur un protocole de consensus. Un consensus permet à un ensemble de nœuds de se mettre d'accord sur une valeur de manière durable : une fois que la décision a été prise et validée par une majorité de nœuds, la décision est irrévocable et définitive.

Dans le cadre de ce projet, nous allons implanter et étudier expérimentalement un protocole de réplication de machine à états. Nous nous baserons sur le protocole distribué Paxos conçu par L. Lamport¹ dans les années 90 et qui est un des protocoles les plus utilisés dans les systèmes réels qui font de la réplication.

Objectif

L'objectif principal de ce projet est de faire une étude expérimentale via le simulateur PeerSim. Il s'agit d'étudier un mécanisme de réplication de machine à états, basé sur le protocole Paxos permettant d'assurer une cohérence sur un ensemble de réplicas d'une même variable logique.

Langage et librairie utilisée

Pour réaliser ce projet, nous avons utilisé le langage *Java* version 11.0.13 ainsi que la librairie *Peersim* version 1.0.5.

1. Leslie Lamport, chercheur en informatique, spécialiste de l'algorithmique répartie. Il est également le concepteur du logiciel libre de composition de documents LaTeX que nous utilisons actuellement.

2 Modélisation du problème

Comme mentionné ci-avant, nous nous plaçons dans un cadre où :

1. n'importe quel nœud du système peut tomber en panne n'importe quand
2. une panne peut être définitive ou temporaire
3. un message peut être perdu ou dupliqué
4. les canaux de communications ne sont pas FIFO

Dans notre système, nous considérons un ensemble de nœuds hébergeant les réplicas noté $R = r_0, r_1, \dots$. Chaque nœud replica r_i est identifié par un indice i tel que $0 \leq i < |R|$. Chaque replica $r_i \in R$ maintient un historique noté \mathcal{H}_i . Cet historique maintient l'ordre des requêtes d'écriture délivrées sur le replica. On pourra supposer que cet historique sera stocké localement sur un support persistant et fiable ce qui permettra de recouvrir plus rapidement un état cohérent lors d'un redémarrage du nœud. L'indice d'une requête dans un historique est appelé itération qui est entier supérieur ou égal à zéro. L'itération x d'un historique \mathcal{H}_i est notée $\mathcal{H}_i(x)$. Ainsi pour tout historique \mathcal{H} du système, si deux itérations x et y existent dans \mathcal{H} et que $x < y$ alors $\mathcal{H}(x)$ désigne une requête qui a été délivrée avant $\mathcal{H}(y)$. La propriété de sureté que doit respecter notre protocole de réplication de machine à états est $\forall \mathcal{H}_i$ et $\forall x \in \mathcal{H}_i$:

- soit $\mathcal{H}_i(x) = \perp$ signifiant que le replica r_i ne connaît pas encore la valeur de la requête associée à l'itération x
- soit $\mathcal{H}_i(x) = req$ tel que $(\forall \mathcal{H}_j \neq \mathcal{H}_i \wedge \mathcal{H}_j(x) \neq \perp) \Rightarrow \mathcal{H}_j(x) = req$ signifiant que tout autre historique qui maintient une valeur différente de \perp à l'itération x implique que la valeur de cette itération doit être égale à req .

2.1 Le protocole Paxos

Paxos est un protocole de consensus permettant aux réplicas de se concerter sur une seule valeur. Dans notre contexte une valeur décidée par Paxos serait une seule itération. Il sera donc nécessaire de démarrer une instance de Paxos pour chaque itération de l'ensemble des historiques (cette utilisation de Paxos est aussi appelée Multi-Paxos).

2.1.1 Principes du protocole

Nous décrivons ici le protocole Paxos basique pour décider d'une valeur (c.-à-d. d'une itération). Les nœuds externes au système sont appelés clients et ce sont ces nœuds qui émettent les requêtes. Parmi l'ensemble des nœuds internes au système, le protocole Paxos définit 3 rôles :

- **Learner** : les nœuds qui hébergent les réplicas du service
- **Acceptor** : les nœuds qui permettent de prendre une décision sur la valeur. Ils servent notamment à se rappeler si une valeur a déjà été décidée dans le passé ou bien si une nouvelle valeur doit être décidée. Leur nombre dépend du nombre de fautes que l'on souhaite tolérer. Les Acceptors doivent toujours former un quorum pour que le protocole fonctionne. Typiquement un quorum peut être n'importe quel ensemble d'Acceptor qui forme une majorité.
- **Proposer** : les nœuds qui reçoivent les requêtes des clients et qui les soumettent à la décision des Acceptors. Ils sont responsables de récolter la réponse de ces derniers pour valider ou rejeter la requête du client.

Le schéma global du protocole est le suivant (de l'émission du client à l'acceptation définitive de la requête).

Étape 0 Un client choisit un *Proposer* p et lui envoie sa requête.

Étape 1a Le *Proposer* p émet à l'ensemble des *Acceptors* un message *Prepare* contenant un numéro de round (appelé aussi *ballot* ou *numrodesquence*). Ce numéro de round doit être strictement supérieur à n'importe quel numéro de round déjà envoyé par p . Le but de ce message étant de savoir si un quorum d'*Acceptors* peut être atteint pour pouvoir soumettre la requête du client.

Étape 1b À la réception d'un message *Prepare* sur un *Acceptor* a depuis un *Proposer* p pour un round n , il y a deux cas à considérer :

- ◇ si n est supérieur à n'importe quel round auquel a ait participé, un message *Promise* est envoyé à p . Ce message *Promise* permet d'indiquer à p que a ne participera plus à un scrutin de round inférieur n . Le message *Promise* contient éventuellement la précédente valeur v que a a déjà acceptée (lors d'une précédente phase 2b) associé à son numéro de round : n_v
- ◇ sinon, a ignore le message ou éventuellement renvoi un message *Reject* à p lui indiquant que son numéro de round est invalide et obsolète.

Étape 2a Lorsque p reçoit une majorité de *Promise*, il doit décider d'une valeur e .

- ◇ S'il existe des *Promise* contenant des valeurs déjà acceptées par son envoyeur, alors p choisit la valeur v avec le numéro n_v le plus grand.
- ◇ sinon, p est libre de choisir la valeur et dans notre cas il choisira la requête que le client lui a envoyée à l'étape 0.

Le *Proposer* p envoie alors à l'ensemble des *Acceptors* la valeur e qu'il a choisie associée au numéro de round n qu'il a envoyé dans le message *Prepare* (désigné parfois aussi comme *Commit*).

Étape 2b À la réception d'un message *Accept* sur un *Acceptor* a depuis un *Proposer* p pour un round n et une valeur e :

- ◇ si n est plus grand ou égal au numéro de round du dernier *Promise* que a ait envoyé, alors il accepte la valeur e . Il mémorise cette valeur (pour les éventuels futurs *Prepare*) et diffuse à l'ensemble des *Learners* qu'à p un message *Accepted* contenant la valeur e .
- ◇ sinon le message est ignoré.

Étape 3 Lorsqu'un *Learner* l reçoit une majorité de messages *Accepted* pour une même valeur e alors l prend acte de cette décision (la valeur est définitivement acceptée) et exécute la requête décrite par la valeur e . En fonction du protocole applicatif qui caractérise la requête, l peut répondre directement au client.

2.1.2 Notion de leader

On peut remarquer que plusieurs *Proposers* peuvent rentrer en conflit en phase 1 (étapes 1a et 1b) pour obtenir le numéro de round le plus haut et pouvoir imposer son leadership. De même, il est possible qu'un *Acceptor* change sa valeur précédemment acceptée s'il rencontre un *Proposer* possédant un numéro plus en étape 2b. Cette bataille pour le leadership des *Proposers* peut amener Paxos à ne jamais converger vers une valeur. C'est pourquoi il est nécessaire que l'ensemble des *Proposers* désigne parmi eux un leader qui assurera une stabilité sur l'exécution du protocole. Cette phase d'élection de leader doit se faire avant la phase 1. Ainsi à la réception d'une requête d'un client sur un *Proposer* p :

- soit p est le leader et commence la phase 1
- soit p n'est pas le leader et peut rejeter la requête du client en indiquant l'identité du leader ou bien retransmettre la requête au leader.

2.2 Adaptation de Paxos à notre modèle

2.2.1 Rôle des nœuds

Dans le cadre de ce projet, nous allons faire une simplification très courante qui consiste à considérer que tout nœud interne au système est à la fois Proposer, Acceptor et Learner. Ainsi, si on considère un système à N nœuds où l'ensemble des nœuds est désigné par $\Pi = n_0, n_1, n_2, \dots, n_{N-1}$ alors :

- tout nœud $n \in \Pi$ a le rôle de Proposer autrement dit éligible à l'élection de leader
- tout nœud $n \in \Pi$ a le rôle d'Acceptor, ce qui veut dire que la condition pour avoir une majorité est de recevoir $\frac{N}{2} + 1$ message(s) Promise et Accepted contenant la même valeur pour pouvoir passer en phase 2 et 3
- tout nœud $n \in \Pi$ a le rôle de Learner impliquant qu'il y a N réplica(s) du service (donc N historique) dans le système. À la validation d'une nouvelle itération x dans son historique \mathcal{H} , avant d'exécuter la requête et répondre au client, un learner devra s'assurer qu'il n'y ait pas d'itération $y < x$ tel que $\mathcal{H}(y) = \perp$. Ceci peut se produire si le nœud est tombé en panne puis a redémarré ou bien si des messages Accepted ont été perdus. Si c'est le cas, il faudra relancer un round Paxos (via le leader) pour connaître les valeurs manquantes.

2.2.2 Élection du leader

Comme nous considérons une réplication de machine à états, il est possible de choisir un identifiant de nœud dans l'itération 0 qui fera office de leader pour les itérations suivantes. On peut ainsi considérer qu'élire un leader, c'est analogue à ce que chaque nœud (en tant que *Proposer*) propose dans l'itération 0 une requête (findLeader) émise par eux même (en tant que client). Le leader sera le nœud qui aura été le plus rapide pour valider son identifiant auprès d'une majorité d'*Acceptors*.

Si ce principe peut s'appliquer à l'itération 0, il est également possible de l'appliquer à n'importe quelle itération dès que l'on détecte la panne du leader (voir ci-après). Ainsi il est toujours possible à tout instant qu'un nœud connaisse l'identité de son leader en regardant dans son historique \mathcal{H} (en tant que *Learner*) la dernière requête findLeader qui a été validée. Ceci est également applicable lorsqu'un nœud procède au recouvrement de son état à la suite d'une panne.

Afin d'éviter une trop longue convergence (car tous les nœuds en tant que Proposer rentrent en conflit pour désigner le leader), il est possible d'introduire un mécanisme de backoff (utilisé notamment dans les détections de collisions sur l'utilisation d'un bus de communication). Ainsi lorsqu'un nœud reçoit un message Reject par un *Acceptor* (ce qui signifie que son round n'est pas ou plus valide), il attend pendant une certaine période aléatoire avant de réitérer sa proposition. À chaque rejet, il augmente ce temps d'attente. Ceci permet de réduire à terme les probabilités de conflit et faire en sorte qu'un seul *Proposer* aient le temps de passer les 2 étapes de validation avant qu'un autre *Proposer* redémarre un round plus élevé. À terme, tous les *Proposers* recevront un identifiant de nœud (en tant que Learner) qu'ils reconnaîtront comme leader : celui du *Proposer* qui a réussi à passer en premier les deux phases du protocole Paxos.

Il faut bien comprendre que dans une itération findLeader, les nœuds (en tant que Proposer) ne reconnaissent aucun leader puisque c'est l'objet de la requête. Ils se battent donc pour être le premier à se désigner.

2.2.3 Surveiller le leader

Afin de contrôler que le leader n'est pas en panne, il est possible d'introduire un mécanisme de ping. Ainsi, chaque nœud envoie régulièrement un message Ping au nœud qu'il croit être leader. À la réception d'un message Ping sur un nœud n_i depuis un nœud n_j , n_i renvoie un message Pong à n_j avec l'identité du leader de n_i (qui peut être n_i lui-même). À la réception d'un message Pong sur n_j , le leader et éventuellement mis à jour.

2.3 Étude expérimentale 1 : itération d'élection

Notre première étude portera uniquement sur l'itération 0 de notre réplique de machine à états, c'est-à-dire l'élection du leader.

Scénario considéré : Tous les nœuds (en tant que *Proposer*) tentent d'élire un leader (via la requête `findLeader`) qui sera utilisé dans les itérations suivantes. Aucune faute ne sera injectée.

Critère de validation : Tous les nœuds doivent avoir la même valeur de leader à la fin du scénario.

Métriques considérées dans cette étude :

- le **temps de convergence** : le temps nécessaire au protocole pour que tous les nœuds reconnaissent le même leader
- le **nombre de rounds** nécessaires pour atteindre une majorité, c'est à dire pour que le consensus soit atteint
- le **nombre de messages** émis

Objectif : Quantifier l'impact des paramètres suivants sur les métriques décrites précédemment :

- le nombre de nœuds dans le système, c'est à dire N
- activation et désactivation d'un temps de backoff. En cas d'activation, on pourra étudier différentes valeurs d'incrémentations qu'un nœud fait à chaque rejet d'un `Prepare`
- la valeur du round initial :
 - ◇ une version avec la valeur 0 (Paxos original)
 - ◇ une version avec l'identifiant du nœud, ce qui donnera normalement plus de chance aux nœuds ayant un plus grand identifiant et donc de converger plus vite.

2.3.1 Paramétrage de l'expérience

Pour cette étude expérimentale, nous avons pris en compte le paramétrage suivant :

- ◇ le nombre de nœuds N dans le système est compris dans un intervalle allant de 10 à 510 par pas de 10
- ◇ l'activation et la désactivation d'un temps de backoff, lorsque celui-ci est activé, il peut prendre 2 valeurs qui ont été choisies arbitrairement : 50 et 200
- ◇ la valeur du round initial à soi pour valeur 0, soit pour valeur l'identifiant du nœud

Pour toutes les courbes ci-dessous, à chaque pas de nœud, nous avons effectué 51 simulations (chacune avec une seed différente allant de 0 à 50), auquel nous avons pris la moyenne des résultats.

Attention : L'unité de temps des schémas ci-dessous est en milliseconde.

Désactivation du backoff avec pour valeur de round initial 0

Spéculation

Avec l'augmentation du nombre de nœud N , nous pouvons nous attendre à voir une augmentation des 3 métriques considérées dans cette étude.

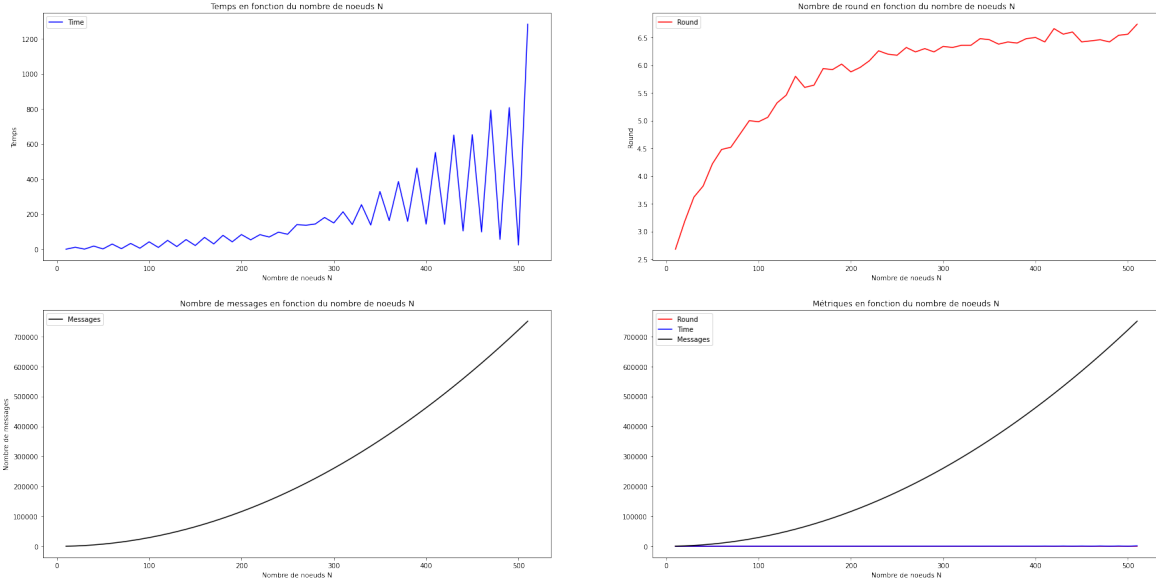


FIGURE 1 – NoBackOffValue0

Interprétation de la figure ci-dessus

Nous pouvons voir que la courbe de temps en fonction du nombre de nœuds (courbe bleue) oscille selon le nombre de nœuds. De manière générale, la courbe croît.

Nous pouvons voir que la courbe du nombre de round en fonction du nombre de nœuds (courbe rouge) croît rapidement au début avant de converger vers 6 à 7 rounds. Cette convergence se remarque aux alentours de 300 nœuds.

Nous pouvons voir que la courbe du nombre de messages en fonction du nombre de nœuds (courbe noire) croît exponentiellement.

Comparaison

Comme nous nous en sommes douté, nous avons pu observer une augmentation des 3 métriques considérées dans cette étude. Étonnamment, nous avons pu voir que la courbe bleue s'est mise à osciller. Malheureusement, nous ne voyons pas de raison pouvant expliquer ce comportement. Quant à la courbe rouge, nous avons pu observer une convergence du nombre de round. Cela signifie dans notre expérience qu'à partir d'un certain nombre de nœuds, la taille du réseau n'influence plus le nombre de round. Et pour finir, nous pouvons voir que la courbe noire croît exponentiellement par rapport aux autres courbes comme nous pouvons le voir sur le quatrième schéma.

Désactivation du backoff avec pour valeur de round initial l'identifiant du nœud

Spéculation

Avec l'augmentation du nombre de nœud N , nous pouvons nous attendre à voir une augmentation des 3 métriques considérées dans cette étude. La valeur de round id devrait permettre une convergence de temps plus rapide.

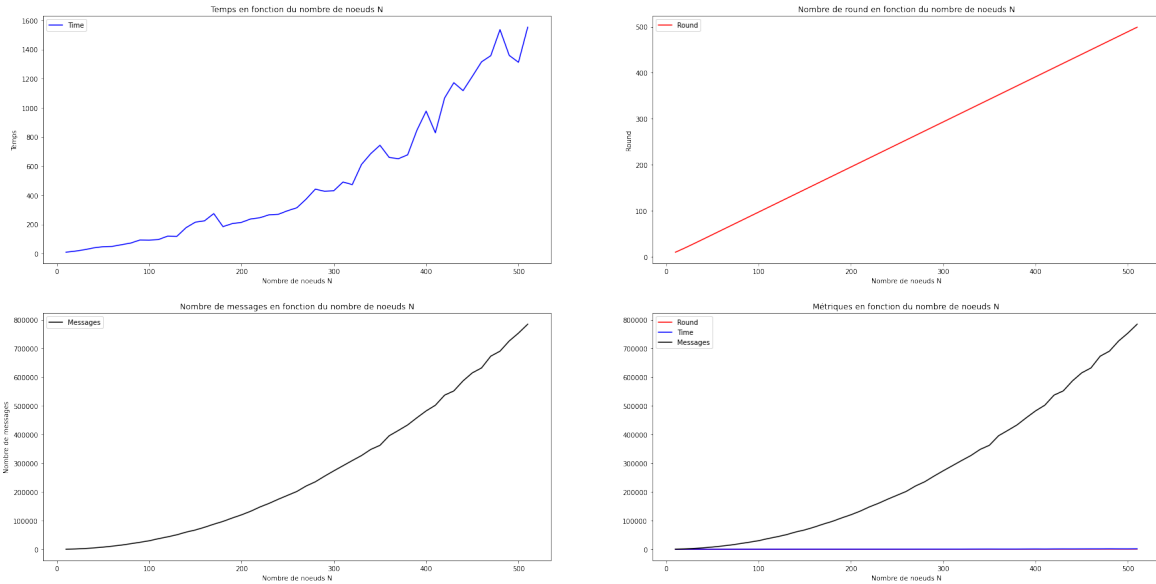


FIGURE 2 – NoBackOffValueID

Interprétation de la figure ci-dessus

Nous pouvons voir que la courbe de temps en fonction du nombre de nœuds (courbe bleue) croît de façon quasi-linéaire et sans oscillations.

Nous pouvons voir que la courbe du nombre de round en fonction du nombre de nœuds (courbe rouge) croît linéairement, ce qui est logique étant donné que la valeur des rounds dépend de l'id des nœuds. On n'observe donc pas de convergence du nombre de rounds.

Nous pouvons voir que la courbe du nombre de messages en fonction du nombre de nœuds (courbe noire) croît exponentiellement.

Comparaison

Comme nous nous en sommes douté, nous avons pu observer une augmentation des 3 métriques considérées dans cette étude. Les courbes correspondent bien à ce à quoi nous nous attendions.

Activation du backoff avec un temps de 50 ms et pour valeur de round initial 0

Spéculation

Avec l'augmentation du nombre de nœud N , nous pouvons nous attendre à voir une augmentation des 3 métriques considérées dans cette étude. Le mécanisme de backoff devrait quant à lui accélérer la convergence de temps mais en contre-partie il va augmenter le nombre de round.

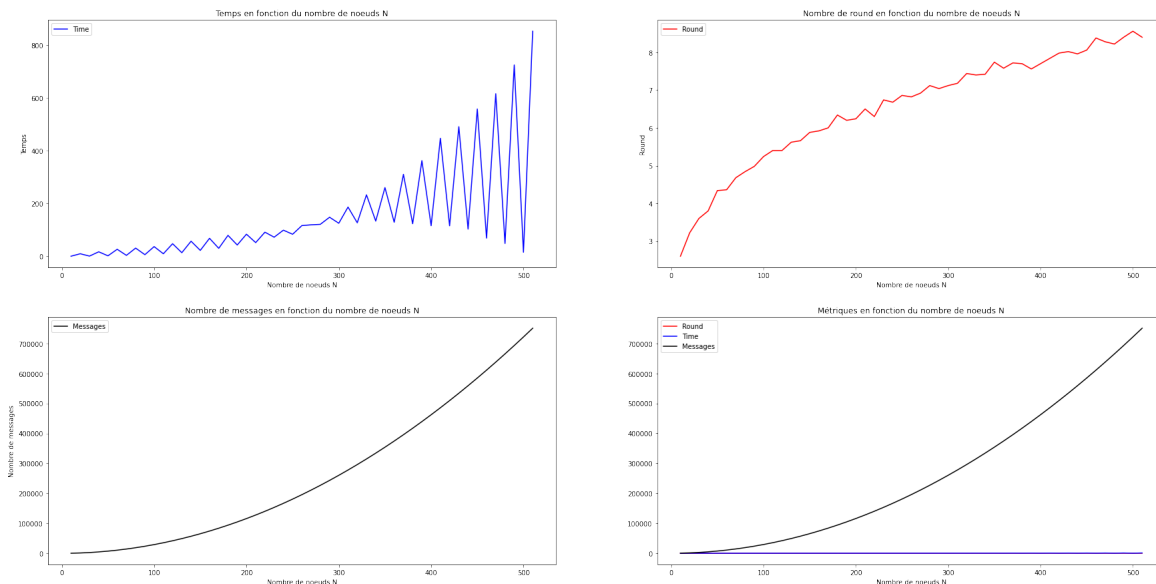


FIGURE 3 – BackOff50Value0

Interprétation de la figure ci-dessus

Nous pouvons voir que la courbe de temps en fonction du nombre de nœuds (courbe bleue) croît avec des oscillations en dents de scie. Mais de manière générale, elle semble croître légèrement de façon quasi-linéaire.

On voit que la courbe du nombre de rounds en fonction du nombre de nœuds (courbe rouge) croît de manière rapide puis connaît un ralentissement et semble converger vers 8 et 9 rounds.

Nous pouvons voir que la courbe du nombre de messages en fonction du nombre de nœuds (courbe noire) croît toujours exponentiellement.

Comparaison

Les courbes correspondent globalement à ce à quoi nous nous attendions, même si nous ne comprenons pas la croissance en dents de scie de la courbe du temps, et surtout l'augmentation de l'intensité des variations après 300 nœuds.

Activation du backoff avec un temps de 50 ms et pour valeur de round initial l'identifiant du nœud

Spéculation

Avec l'augmentation du nombre de nœud N , nous pouvons nous attendre à voir une augmentation des 3 métriques considérées dans cette étude. Le mécanisme de backoff devrait quant à lui accélérer la convergence de temps tout en augmentant le nombre de round.

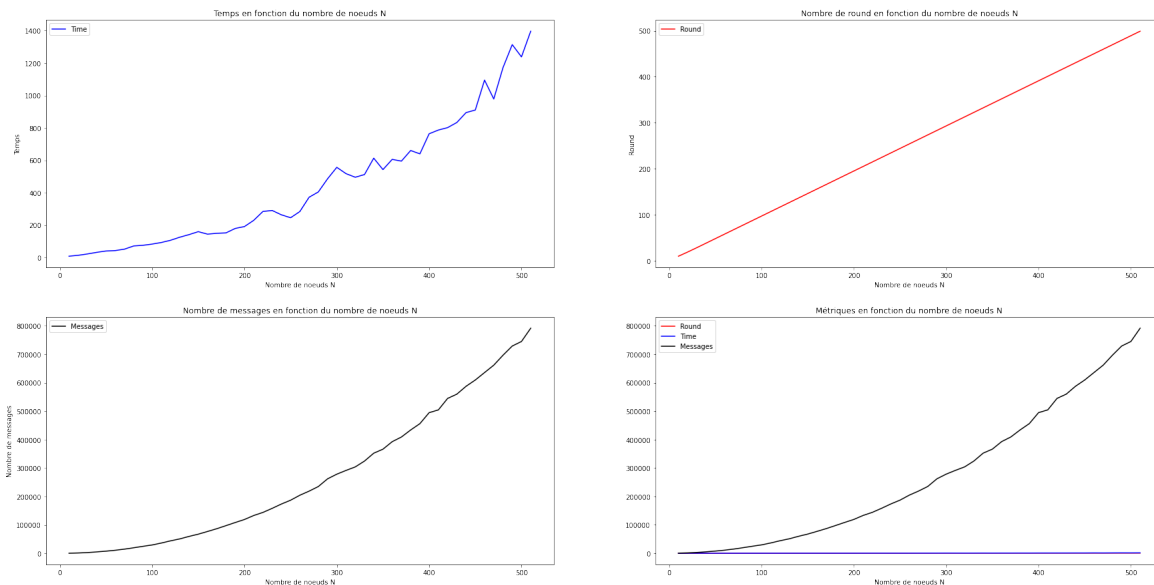


FIGURE 4 – BackOff50ValueID

Interprétation de la figure ci-dessus

Nous pouvons voir que la courbe de temps en fonction du nombre de nœuds (courbe bleue) croît de façon quasi-linéaire, avec une augmentation de la croissance à partir de 300 nœuds.

Nous pouvons voir que la courbe du nombre de round en fonction du nombre de nœuds (courbe rouge) croît linéairement.

Nous pouvons voir que la courbe du nombre de messages en fonction du nombre de nœuds (courbe noire) croît exponentiellement.

Comparaison

Comme nous nous en sommes douté, nous avons pu observer une augmentation des 3 métriques considérées dans cette étude. Les courbes correspondent bien à ce à quoi nous nous attendions.

Activation du backoff avec un temps de 200 ms et pour valeur de round initial 0

Spéculation

Avec l'augmentation du nombre de nœud N , nous pouvons nous attendre à voir une augmentation des 3 métriques considérées dans cette étude. Le mécanisme de backoff devrait accélérer la convergence de temps tout en augmentant le nombre de round.

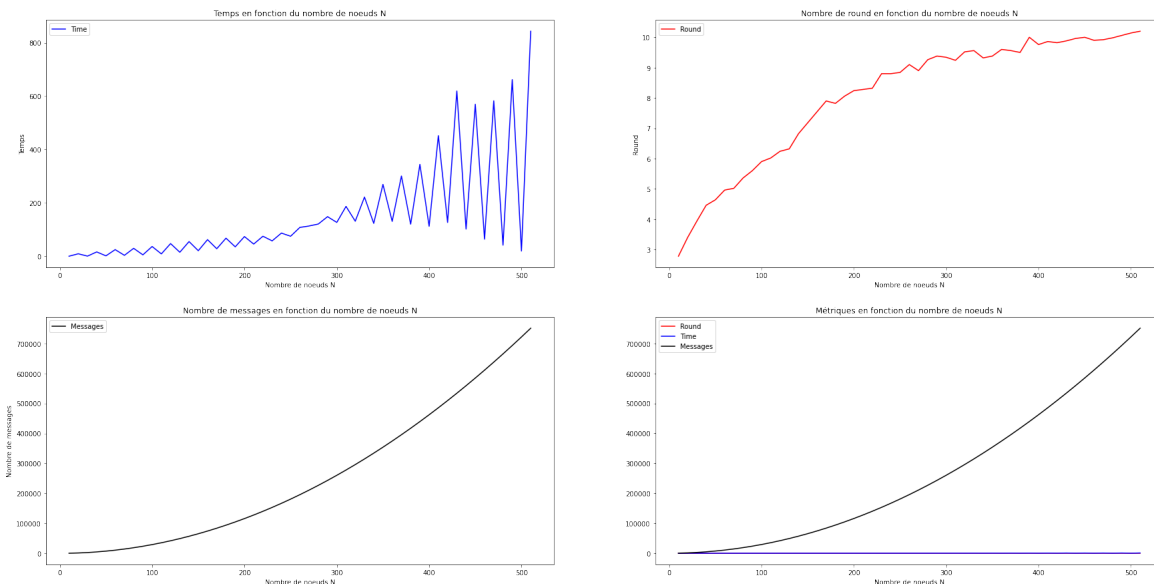


FIGURE 5 – BackOff200Value0

Interprétation de la figure ci-dessus

Nous pouvons voir que la courbe de temps en fonction du nombre de nœuds (courbe bleue) oscille selon le nombre de nœuds. De manière générale, la courbe croît. Le phénomène d'oscillation semble s'intensifier à partir de 300 nœuds.

Nous pouvons voir que la courbe du nombre de round en fonction du nombre de nœuds (courbe rouge) croît rapidement au début avant de converger vers 9 à 10 rounds. Cette convergence se remarque aux alentours de 300 nœuds.

Nous pouvons voir que la courbe du nombre de messages en fonction du nombre de nœuds (courbe noire) croît exponentiellement.

Comparaison

Comme nous nous en sommes douté, nous avons pu observer une augmentation des 3 métriques considérées dans cette étude.

Activation du backoff avec un temps de 200 ms et pour valeur de round initial l'identifiant du nœud

Spéculation

Avec l'augmentation du nombre de nœud N , nous pouvons nous attendre à voir une augmentation des 3 métriques considérées dans cette étude.

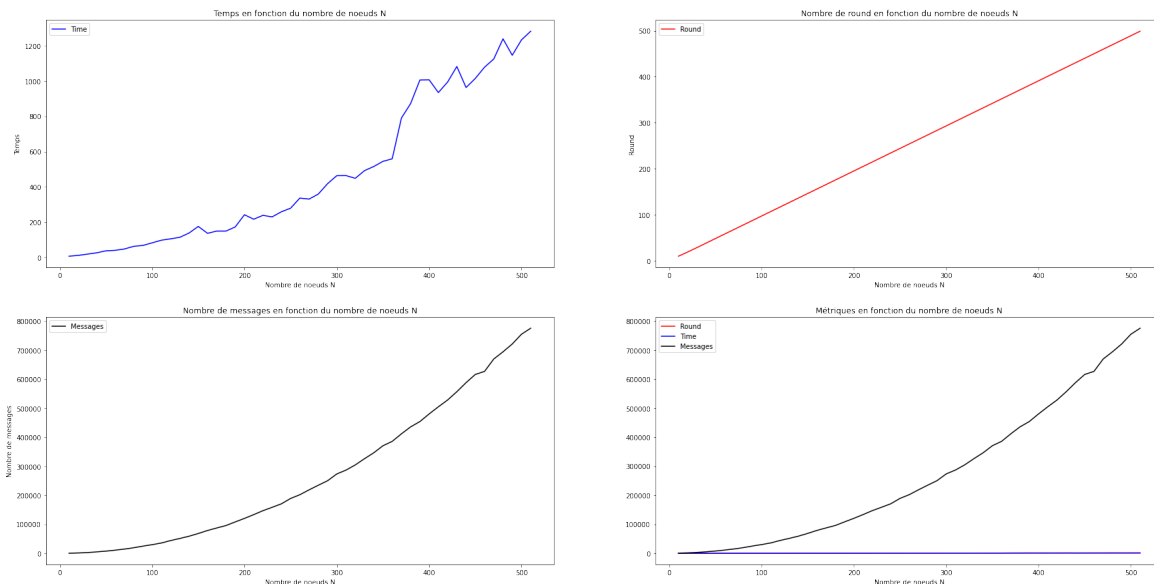


FIGURE 6 – BackOff200ValueID

Interprétation de la figure ci-dessus

Nous pouvons voir que la courbe de temps en fonction du nombre de nœuds (courbe bleue) croît de façon quasi-linéaire, avec une augmentation de la croissance à partir de 400 nœuds.

Nous pouvons voir que la courbe du nombre de round en fonction du nombre de nœuds (courbe rouge) croît linéairement.

Nous pouvons voir que la courbe du nombre de messages en fonction du nombre de nœuds (courbe noire) croît exponentiellement.

Comparaison

Comme nous nous en sommes douté, nous avons pu observer une augmentation des 3 métriques considérées dans cette étude. Les courbes correspondent bien à ce à quoi nous nous attendions.

2.3.2 Conclusion de l'expérience

Tout d'abord nous allons commencer par discuter du mécanisme de backoff. En comparant les figures 1, 3 et 5 (valeur de round initial à 0), nous remarquons que le mécanisme de backoff permet d'accélérer la convergence de temps mais en contre-partie augmente le nombre de round. Dans la figure 1 (sans backoff), le temps maximum de convergence est de 1200 ms pour 6.5 rounds en moyenne, tandis que pour les figures 3 et 5 le temps maximum de convergence est de 800 ms et 8.5 et 10 rounds respectivement. Nous pouvons remarquer aussi que le mécanisme de backoff n'influence pas le nombre de messages. En comparant les figures 2, 4 et 6 (valeur de round initiale égale à l'identifiant du noeud), nous observons le même phénomène que pour le cas avec la valeur de round initial à 0, à l'exception de l'augmentation du nombre de round. Nous pouvons donc en déduire que le mécanisme de backoff influence uniquement la vitesse de convergence de temps.

Passons maintenant à la valeur du round. Commençons par comparer les figures 1 et 2 (sans mécanisme de backoff), nous remarquons plusieurs différences entre le cas où la valeur de round initial est à 0 ou bien égale à l'identifiant du noeud. Tout d'abord nous pouvons voir qu'avec la valeur de round initial à 0, la convergence de temps oscille tandis que le nombre de round converge. Dans le cas où la valeur de round est égale à l'identifiant du noeud, la convergence de temps ainsi que le nombre de round semblent être linéaire. Nous remarquons aussi que le nombre de messages est supérieur avec la valeur égale à l'identifiant du noeud. En comparant les figures 3, 4 (avec un mécanisme de backoff de 50 ms) et 5, 6 (avec un mécanisme de backoff de 200 ms), nous observons exactement le même phénomène. Nous pouvons donc en déduire que la valeur de round initiale influence le nombre de messages, la forme de la courbe de convergence de temps ainsi que du nombre de round. En principe, la valeur de round initial à 0 converge plus rapidement en temps, converge en nombre de rounds et nécessite moins de messages.

Passons finalement au nombre de noeuds. Comme nous pouvons le voir sur tous les schémas, l'augmentation du nombre de noeud augmente à la fois le temps de convergence, le nombre de round ainsi que le nombre de messages.

2.4 Étude expérimentale 2 : Multi-Paxos séquentiel

Notre deuxième étude portera sur une exécution du système dans un temps imparti.

Scénario considéré : En dehors d’une itération d’élection, le leader soumet séquentiellement des requêtes. Le leader attend qu’une itération soit validée (en tant que *Learner*) pour passer à la suivante. En fonction des résultats que vous avez eus dans l’étude précédente, vous prendrez la meilleure configuration en ce qui concerne les paramètres de backoff et de round initial dans les itérations findLeader.

Critère de validation : La propriété de sûreté décrite à la section 2 doit être vérifiée.

Métriques considérées dans cette étude :

- le **débit moyen de requêtes applicatives validées** qui est le rapport entre le nombre total de requêtes hors élection validées et le temps d’expérience
- la **latence moyenne** qui est le temps moyen pour qu’une requête applicative (hors élection) soit validée
- le **nombre de messages** émis (à l’exception des messages Ping et Pong)

Objectif : Quantifier l’impact des paramètres suivants sur les métriques décrites précédemment :

- le nombre de nœuds dans le système, c’est à dire N
- probabilité qu’un nœud tombe en panne
- probabilité pour qu’une panne soit franche ou transitoire

On s’assurera que le nombre de nœuds en panne franche soit toujours en minorité afin de garantir que le protocole fonctionne

2.4.1 Paramétrage de l’expérience

Pour cette étude expérimentale, nous avons pris en compte le paramétrage suivant :

- ◇ le nombre de nœuds N dans le système est compris dans un intervalle allant de 10 à 310 par pas de 10
- ◇ la probabilité qu’un nœud tombe en panne est soit de 10%, 25% ou bien 50%.
- ◇ l’injection ou non de panne franche ou transitoire

Pour toutes les courbes ci-dessous, à chaque pas de nœud, nous avons effectué 51 simulations (chacune avec une seed différente allant de 0 à 50), auquel nous avons pris la moyenne des résultats.

Attention : L’unité de temps des schémas ci-dessous est en milliseconde.

Sans fautes avec une probabilité de panne de noeuds de 0.1

Spéculation

Avec l'augmentation du nombre de nœud N , nous pouvons nous attendre à voir une augmentation des 3 métriques considérées dans cette étude.

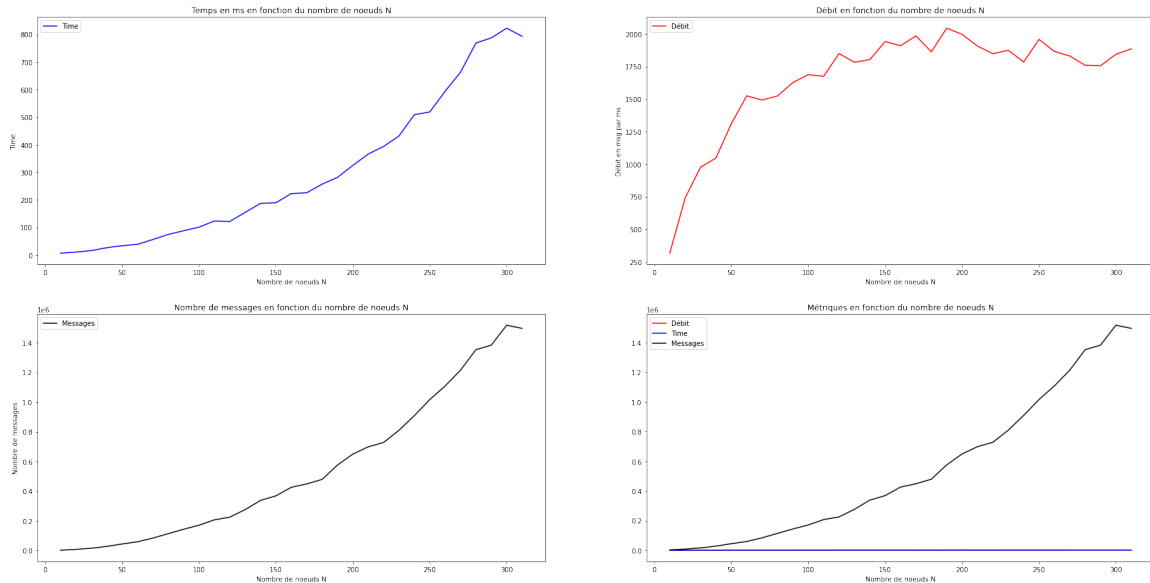


FIGURE 7 – NoFaultWithDrop10%

Interprétation de la figure ci-dessus

Nous pouvons voir que la courbe de latence moyenne en fonction du nombre de nœuds (courbe bleue) semble être croissante.

Nous pouvons voir que la courbe du débit moyen en fonction du nombre de nœuds (courbe rouge) semble être croissante et convergente vers un débit de 1800 messages/ ms à partir de 150 noeuds.

Nous pouvons voir que la courbe du nombre de messages en fonction du nombre de nœuds (courbe noire) croît exponentiellement.

Comparaison

Comme nous nous en sommes douté, nous avons pu observer une augmentation des 3 métriques considérées dans cette étude. Les courbes correspondent bien à ce à quoi nous nous attendions.

Fautes avec une probabilité de panne de noeuds de 0.1

Spéculation

Avec l'augmentation du nombre de nœud N et l'ajout de fautes, nous pouvons nous attendre à voir une augmentation des 3 métriques considérées dans cette étude.

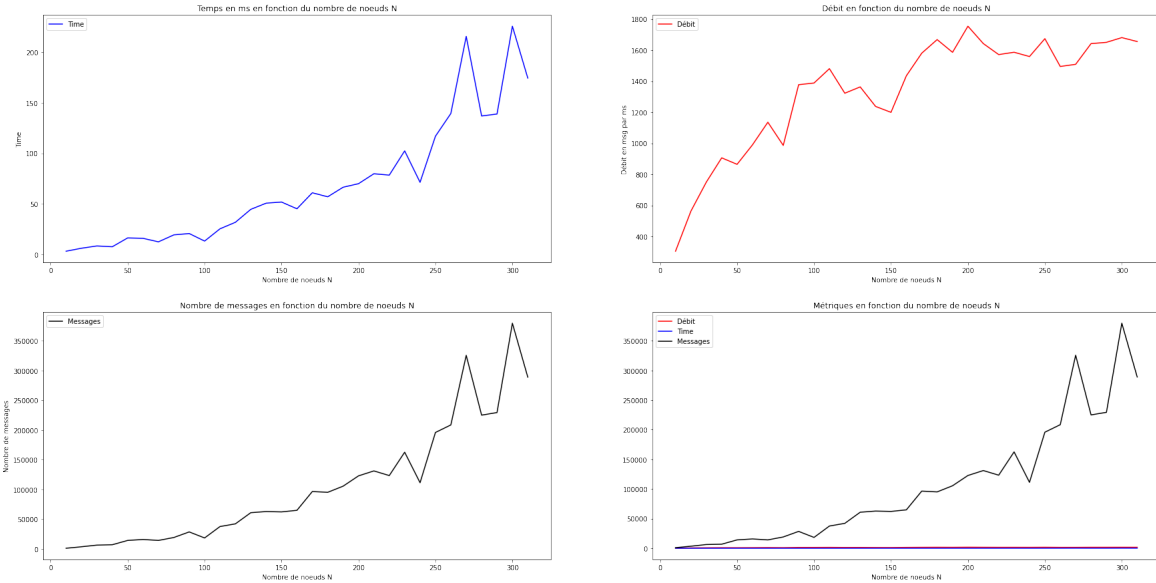


FIGURE 8 – FaultWithDrop10%

Interprétation de la figure ci-dessus

Nous pouvons voir que la courbe de latence moyenne en fonction du nombre de nœuds (courbe bleue) semble être croissante de façon linéaire jusqu'à 250 nœuds puis commence à osciller en dent de scie.

Nous pouvons voir que la courbe du débit moyen en fonction du nombre de nœuds (courbe rouge) semble croître rapidement avant de converger vers 1700 messages par ms à partir de 200 nœuds.

Nous pouvons voir que la courbe du nombre de messages en fonction du nombre de nœuds (courbe noire) semble avoir la même tendance que le temps.

Comparaison

Comme nous nous en sommes douté, nous avons pu observer une augmentation des 3 métriques considérées dans cette étude. Par contre, nous observons que les courbes de temps et de messages possèdent le même schéma.

Sans fautes avec une probabilité de panne de noeuds de 0.25

Spéculation

Avec l'augmentation du nombre de nœud N et de la probabilité de panne de noeuds, nous pouvons nous attendre à voir une augmentation des 3 métriques considérées dans cette étude.

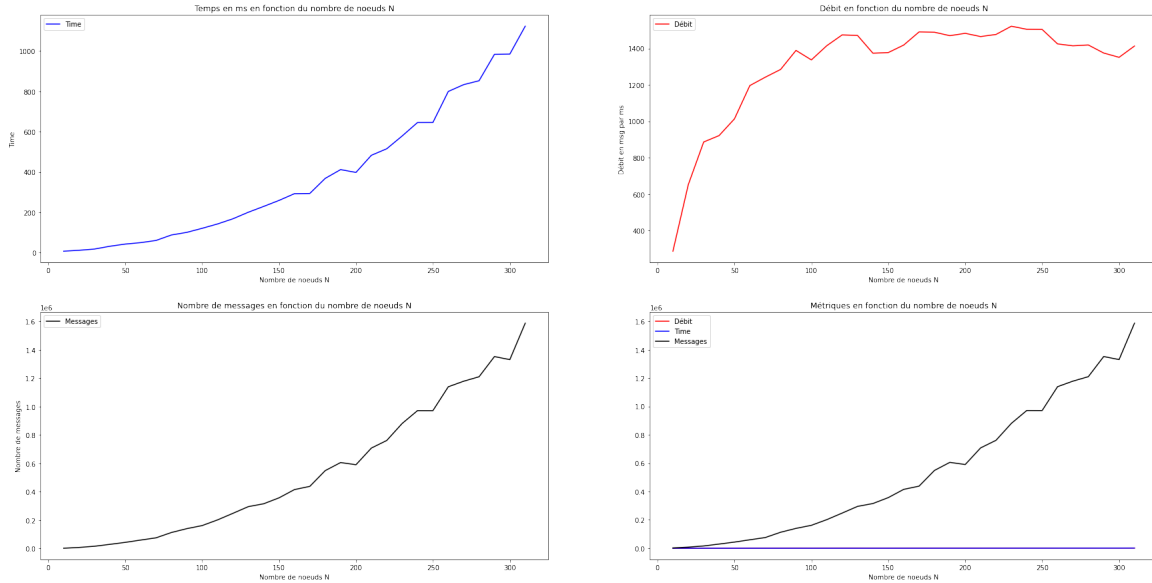


FIGURE 9 – NoFaultWithDrop25%

Interprétation de la figure ci-dessus

Nous pouvons voir que la courbe de latence moyenne en fonction du nombre de nœuds (courbe bleue) semble être croissante linéaire.

Nous pouvons voir que la courbe du débit moyen en fonction du nombre de nœuds (courbe rouge) semble être rapidement croissante avant de converger vers un débit de 1400 messages/ ms à partir de 150 noeuds.

Nous pouvons voir que la courbe du nombre de messages en fonction du nombre de nœuds (courbe noire) croît exponentiellement.

Comparaison

Comme nous nous en sommes douté, nous avons pu observer une augmentation des 3 métriques considérées dans cette étude. Les courbes correspondent bien à ce à quoi nous nous attendions.

Fautes avec une probabilité de panne de noeuds de 0.25

Spéculation

Avec l'augmentation du nombre de nœud N et l'ajout de faute, nous pouvons nous attendre à voir une augmentation des 3 métriques considérées dans cette étude.

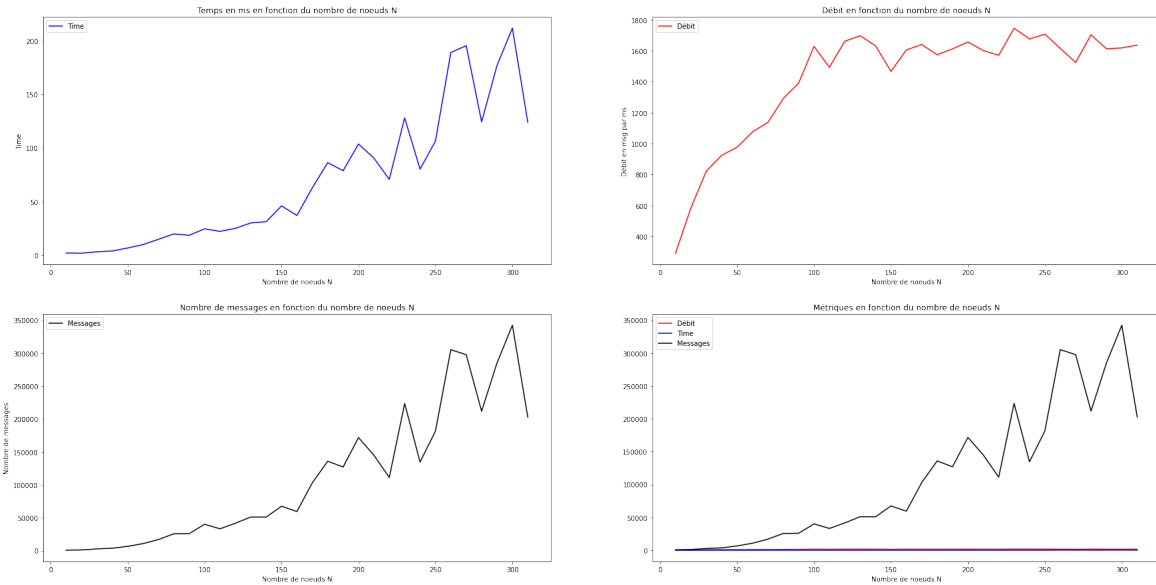


FIGURE 10 – FaultWithDrop25%

Interprétation de la figure ci-dessus

Nous pouvons voir que la courbe de latence moyenne en fonction du nombre de nœuds (courbe bleue) semble être croissante de façon linéaire jusqu'à 250 nœuds puis commence à osciller en dent de scie.

Nous pouvons voir que la courbe du débit moyen en fonction du nombre de nœuds (courbe rouge) semble croître rapidement avant de converger vers 1600 messages par ms à partir de 100 nœuds.

Nous pouvons voir que la courbe du nombre de messages en fonction du nombre de nœuds (courbe noire) semble avoir la même tendance que le temps.

Comparaison

Comme nous nous en sommes douté, nous avons pu observer une augmentation des 3 métriques considérées dans cette étude.

Sans fautes avec une probabilité de panne de noeuds de 0.5

Spéculation

Avec l'augmentation du nombre de nœud N et de la probabilité de panne de noeuds, nous pouvons nous attendre à voir une augmentation des 3 métriques considérées dans cette étude.

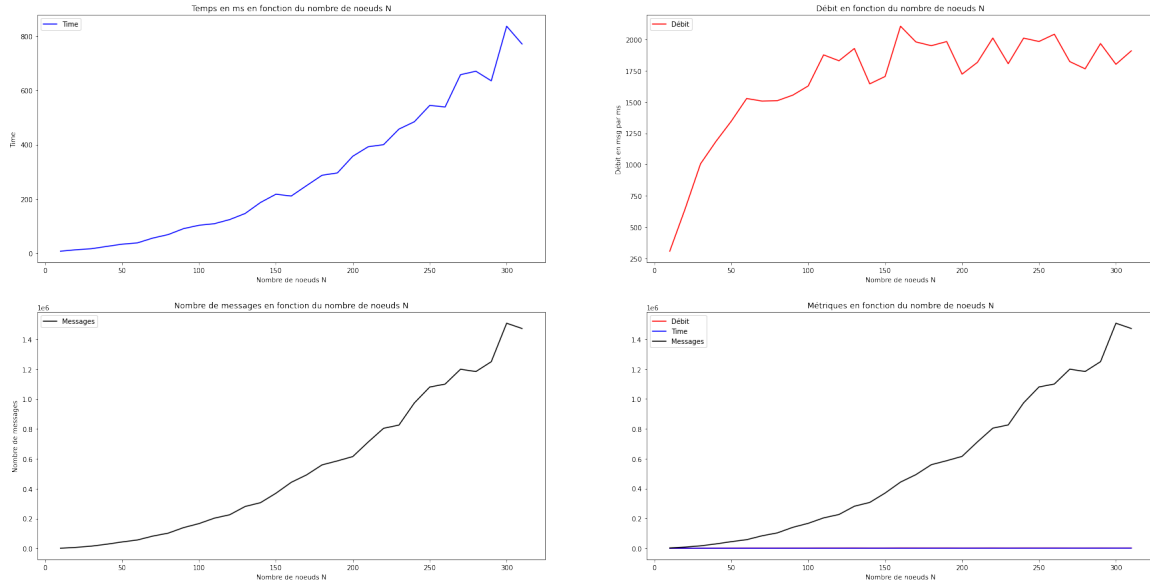


FIGURE 11 – NoFaultWithDrop50%

Interprétation de la figure ci-dessus

Nous pouvons voir que la courbe de latence moyenne en fonction du nombre de nœuds (courbe bleue) semble être croissante linéaire.

Nous pouvons voir que la courbe du débit moyen en fonction du nombre de nœuds (courbe rouge) semble être rapidement croissante avant de converger vers un débit de 1900 messages/ ms à partir de 150 noeuds.

Nous pouvons voir que la courbe du nombre de messages en fonction du nombre de nœuds (courbe noire) croît exponentiellement.

Comparaison

Comme nous nous en sommes douté, nous avons pu observer une augmentation des 3 métriques considérées dans cette étude. Les courbes correspondent bien à ce à quoi nous nous attendions.

Fautes avec une probabilité de panne de noeuds de 0.5

Spéculation

Avec l'augmentation du nombre de nœud N et l'ajout de faute, nous pouvons nous attendre à voir une augmentation des 3 métriques considérées dans cette étude.

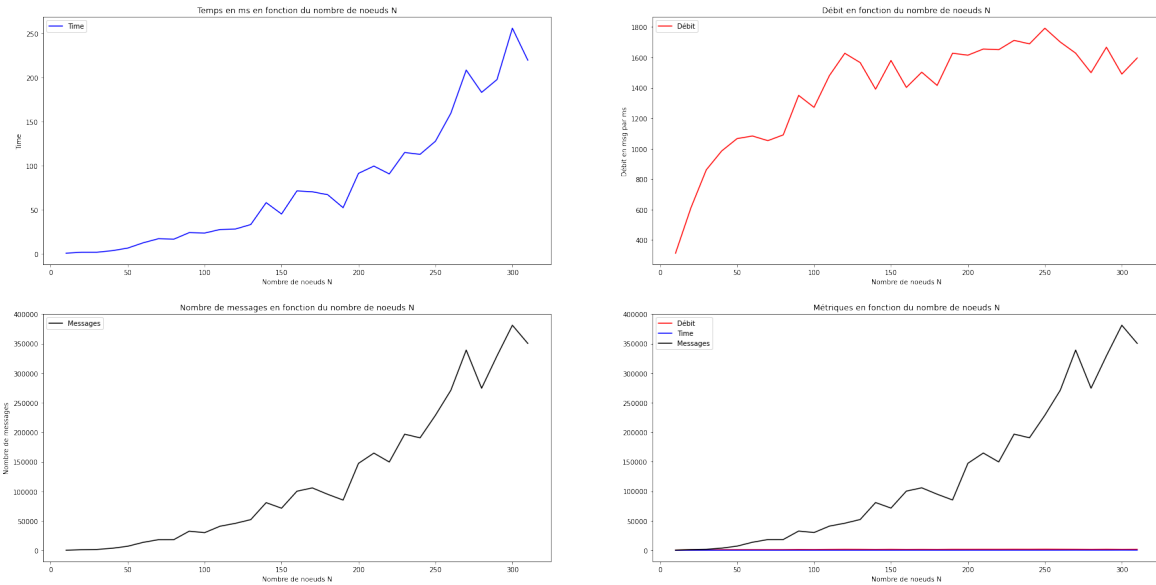


FIGURE 12 – FaultWithDrop50%

Interprétation de la figure ci-dessus

Nous pouvons voir que la courbe de latence moyenne en fonction du nombre de nœuds (courbe bleue) semble être croissante de façon linéaire.

Nous pouvons voir que la courbe du débit moyen en fonction du nombre de nœuds (courbe rouge) semble croître rapidement avant de converger vers 1600 messages par ms à partir de 130 nœuds.

Nous pouvons voir que la courbe du nombre de messages en fonction du nombre de nœuds (courbe noire) semble avoir la même tendance que le temps.

Comparaison

Comme nous nous en sommes douté, nous avons pu observer une augmentation des 3 métriques considérées dans cette étude.

2.4.2 Conclusion de l'expérience

Tout d'abord nous allons commencer par discuter de l'injection ou non de faute. En comparant les figures 7 et 8 (probabilité de 10% de panne de noeud), nous remarquons que l'injection de faute réduit à la fois le temps et le nombre de messages et donc par la même occasion le débit. Nous remarquons les mêmes observations pour les figures 9, 10 (probabilité de 25% de panne de noeud) et pour les figures 11, 12 (probabilité de 50% de panne de noeud). Cela semble logique : en effet l'injection de fautes met hors-jeu un certain nombre de noeuds qui alors n'enverront plus de messages. Or pour arriver à un consensus, il suffit seulement d'atteindre un quorum qui est égal à la moitié du nombre de noeuds + 1. Ainsi, le fait de ne plus avoir ces noeuds actifs dans le protocole permet de réduire le nombre de messages et, en plus du fait que l'on a moins de noeuds actifs, de réduire le temps de convergence.

Passons maintenant à la probabilité qu'un noeud tombe en panne. En comparant les figures 7, 9 et 11 (sans injection de faute), nous remarquons que de 10% à 50% en passant par 25% le temps croît puis diminue, de même pour le nombre de messages et c'est le cas inverse pour le débit. En comparant les figures 8, 10 et 12 (avec injection de faute), nous remarquons que le temps croît légèrement, de même pour le nombre de messages. Par contre, le débit reste à peu près constant peu importe la probabilité. Nous ne savons pas trop comment interpréter ces résultats. Il nous semble que dans le cas sans injection de fautes, vers 25% nous atteignons un pic de temps et de messages, comme si nous avions à la fois pas assez de probabilité pour retirer assez de noeud pour diminuer le temps et le nombre de messages mais aussi trop de probabilité de tomber en panne pour que la convergence ce fasse sans encombre.

Passons finalement au nombre de noeuds. Comme nous pouvons le voir sur tous les schémas, l'augmentation du nombre de noeud augmente à la fois temps et le nombre de messages, donc par conséquent le débit.