Comparaison des performances d'une base de données NoSQL et d'une base de données basée sur les technologies de la blockchain

Thomas Beauchamp
dept. Informatique et Logiciel
de Polytechnique Montréal
Montréal, Canada
thomas.beauchamp@polymtl.ca

Olivier Coulombe-Raymond
dept. Informatique et Logiciel
de Polytechnique Montréal
Montréal, Canada
olivier.coulombe-raymond@polymtl.ca

Sébastien Dagenais
dept. Informatique et Logiciel
de Polytechnique Montréal
Montréal, Canada
sebastien.dagenais@polymtl.ca

Oussama Ressak
dept. Informatique et Logiciel
de Polytechnique Montréal
Montréal, Canada
oussama.ressak@polymtl.ca

***Abstrait*—La mondialisation des industries et des services rend le besoin de rendre accessible les centres et les bases de données à des clients à travers le monde crucial. La demande grandissante de solutions supportant des ensembles de données extrêmement grands, le BigData, a rapidement encouragé l'émergence de nombreuses solutions permettant des transactions rapides dans des bases de données immenses. La distribution de ces ensembles de données apporte un nouveau problème que la blockchain pourrait résoudre relativement facilement… Elle a ses avantages et désavantages du à sa nature même et l'étude proposée ici vise à évaluer les performances brutes de la blockchain contre des solutions déjà globalement utilisées, soit les bases de données non-relationnelles (NoSQL) distribuées.**

I. Introduction

Dans ce travail, nous proposons une comparaison des performances de lecture et d'écriture d'une base de données NoSQL MongoDB et d'une base de données distribuée Hyperledger Fabric.

A. Motivation

Les avantages que prodigue une base de données distribuées sont nombreux, mais principalement centrés autour de 3 grands axes : la disponibilité du service, la confidentialité des données et l'intégrité des transactions et des données. L'utilisation des technologies de la blockchain permet de garantir une satisfaction de ces 3 caractéristiques.

D'abord, la disponibilité du service est assurée par la répartition des nœuds et des instances de la base de données. Puisqu'elle est distribuée, l'échec d'un nœud et d'une copie de la base de données peut être remplacé temporairement ou de façon permanente par une autre instance [1]. Avec la distribution, on élimine le point d'échec unique, car toutes les lectures et écritures peuvent se faire sur n'importe quel nœud, sans avoir besoin d'un orchestrateur au centre.

Ensuite, la confidentialité des données dans ce réseau est obtenue par l'utilisation d'un système de clés publiques/privées [1]. Chaque membre de la chaîne possède sa propre paire de clés qui lui permettent de crypter toutes ses communications avec le réseau, donc cela rend la tâche extrêmement plus difficile pour un acteur externe au réseau d'intercepter un message ou une transaction. Les utilisateurs mêmes du réseau ne peuvent pas déchiffrer la clé d'un autre utilisateur, donc ils sont tous protégés au sein de leur propre réseau aussi.

Finalement, l'intégrité du service est garantie par l'utilisation de plusieurs nœuds dans la chaîne. Chacun de ces nœuds possède une copie exacte des données et il est donc possible d'assurer une redondance des données très facilement [2]. Une erreur dans la copie d'un des nœuds sera invalidée par consensus et la perte de l'un des nœuds n'a pas d'impact sur la santé du réseau dans son ensemble [2]. Avec une architecture non monolithique, il est très difficile de corrompre les données de la base de données, puisque la corruption doit être acceptée par tous les nœuds du réseau.

B. Caractéristiques

Les bases de données distribuées ont plusieurs avantages. L'avantage principal est probablement la disponibilité du réseau. Comme un système distribué est un ensemble de nœuds physiques, il est facile de redistribuer la charge de travail sur d'autres composantes. Ceci a donc pour effet d'améliorer la performance du système au niveau de la latence et du débit, mais aussi d'assurer le temps d'activité du système.

Ces systèmes ont aussi l'avantage d'être flexibles et modulaires. Ainsi il est simple de faire une mise à l'échelle de la base de données. De plus, il n'est pas nécessaire d'investir sur des composantes coûteuses pour augmenter la capacité du système puisqu'il se base sur la force du nombre et non la qualité des nœuds de façon individuelle.

Il y a aussi un grand avantage au niveau de la sécurité et de l'intégrité des informations stockées. Par principe de redondance, les bases de données dupliquent les informations, permettant de rendre difficile de modifier l'information pour un attaquant. Cette duplication permet aussi d'augmenter l'efficacité du système puisque plusieurs lectures peuvent se faire au même moment.

C. Objectifs

L'objectif de ce travail est donc de faire une analyse entre 2 modèles de base de données distribués afin d'en tirer les avantages et inconvénients de chacun. Les 2 modèles sont le blockchain et le NoSQL. Pour l'architecture en blockchain nous utiliserons Hyperledger Fabric et pour NoSQL nous utiliserons MongoDB. L'analyse portera sur les différences.

II. CONTEXTE

A. *Hyperledger Fabric*

Hyperledger Fabric est une base de données open source basée sur les technologies de la blockchain, elle est donc distribuée [3]. L'objectif principal de Hyperledger Fabric est de fournir une base de données flexible et évolutive [3]. Elle n'est pas dépendante d'un langage de programmation ou d'une cryptomonnaie spécifique, la rendant très facilement adaptable à n'importe quelle industrie [3].

Une autre grande caractéristique qui distingue Hyperledger Fabric de ses alternatives est la possibilité de créer des canaux différents pour certaines entreprises/compétiteurs, sous la forme d'organisations [3]. Ainsi, un certain canal pourrait être réservé pour une entreprise et seuls les nœuds appartenant à cette entreprise posséderont une copie des informations [3]. Cela offre la possibilité d'utiliser le même réseau pour une industrie complète au lieu d'avoir un réseau par entreprise [3].

Le fonctionnement des opérations en lecture et en écriture est beaucoup moins simple et direct que dans une base de données NoSQL, à cause des restrictions qu'impose l'utilisation d'une blockchain sécurisée. Les activités d'écriture sont effectuées en utilisant un peer et son ledger afin de soumettre une nouvelle transaction et ainsi créer un nouveau bloc, puis l'ajouter une fois qu'il sera accepté par tous ses pairs dans le réseau.

Hyperledger Fabric utilise un consensus de type « practical Byzantine Fault Tolerance (pBFT) » [4]. Ce type de consensus fonctionne de façon asynchrone afin de limiter le temps immédiat d'attente pour la confirmation d'une lecture ou écriture dans la chaîne [4]. C'est une caractéristique très intéressante pour une base de données, car cela réduit de beaucoup le temps de surcharge. C'est un problème que d'autres méthodes de consensus, comme la méthode plus traditionnelle de Proof of Work (PoW) qui est nécessaire avec un réseau de cryptomonnaies [5], car l'approbation de tous les membres de la chaîne est beaucoup plus critique pour accepter une transaction. Un autre avantage de cette méthode de consensus est la consommation de ressources réduites de l'algorithme, car PoW demande de faire des calculs très complexes pour atteindre un consensus [5].

En lecture, le flot des opérations (figure 1) est loin d'être aussi simple qu'un seul simple accès à une ressource par index comme avec MongoDB que nous présenterons plus tard. Puisque c'est un réseau normalement décentralisé, la première étape après l'envoi de la transaction est de trouver un peer pouvant endosser la transaction. Ensuite, une fois la confirmation qu'un peer prendra la responsabilité de la transaction, le client pourra envoyer sa transaction au bon peer, puis attendre sa réponse à sa demande de lecture. Ces étapes ajoutent beaucoup de tâches et de temps supplémentaire en lecture et nous prévoyons que l'utilisation de la blockchain pour faire des requêtes en rafale en lecture sera probablement loin d'être idéale et aussi rapide que la base de données MongoDB.

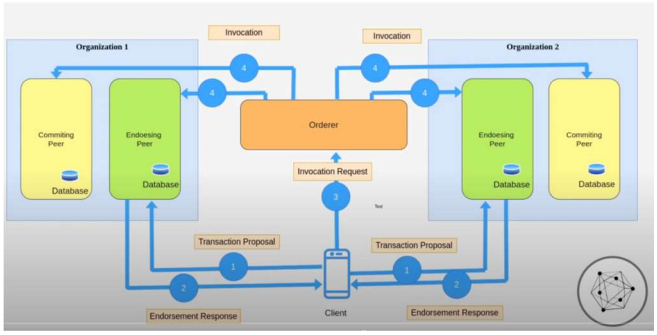

Fig. 1. Flot d'une transaction de lecture dans un réseau distribué Hyperledger Fabric [6]

B. *MongoDB*

MongoDB est, pour sa part, une base de données NoSQL open source [7]. C'est une base de données adaptée pour la tendance du Big Data. Elle fonctionne par indexation sur des collections de données et les opérations sur les champs sont faites de façon atomique (tous les changements sont effectués ou aucun) [6]. Un grand avantage de MongoDB et de sa façon d'indexer chaque donnée est que cela rend possible une séparation de la base de données en shards [6]. Ces shards sont chacun un ensemble d'entrées dans la base de données, séparés dans différents ensembles/systèmes de fichiers [8]. Ainsi, il est possible de séparer l'ensemble de la base de données dans plusieurs shards et de leur assigner un opérateur/une machine primaire différente pour chacun [8]. Toutes les opérations pour chaque groupe d'index seront donc séparées et la charge totale de la base de données sera distribuée.

La base de données que nous avons déployée, présentée plus en détail dans une section future, utilise aussi la capacité de l'architecture de MongoDB de créer des ensembles de machines pour répartir les tâches applicables à un shard. Des instances pourront rouler et seulement effectuer des requêtes de lecture afin de distribuer la charge de travail, alors que l'écriture pour ce shard se fait seulement par une instance primaire. Ce type d'architecture est appelé un ReplicaSet, puisque les instances secondaires de lecture utilisent une réplique des données modifiées par l'instance primaire.

MongoDB est une base de données faite pour stocker de relativement petits éléments, allant jusqu'à une taille maximale de 16 MB [9].

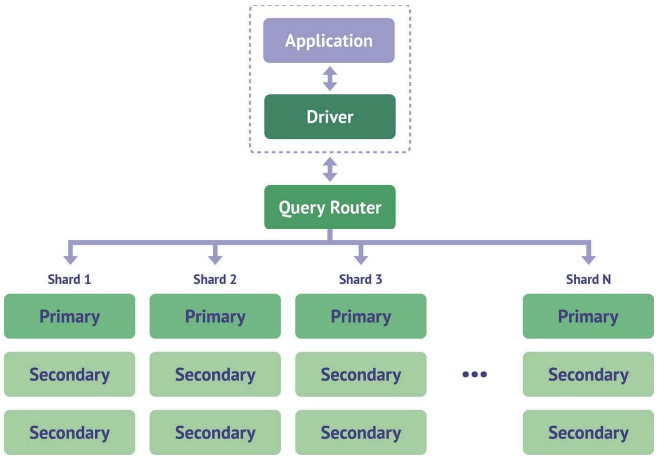

Fig. 2. Représentation d'un déploiement d'une base de données MongoDB distribuée [10]

La figure 2 représente l'architecture distribuée d'une base de données MongoDB. Comme mentionné, la base de données est séparée en plusieurs shards, gérés tous par une instance primaire avec leurs propres ReplicaSet et ensemble d'instances secondaires pour la lecture.

C. *Comparaison*

Plusieurs approches sont possibles pour comparer ces deux architectures distribuées. Les modèles se ressemblent sur plusieurs points. Cependant, une différence majeure crée un écart considérable au niveau de la latence des requêtes et de la sécurité. Un modèle en blockchain doit traiter chaque accès avec un concept de consensus à la grandeur du réseau. En d'autres mots, une majorité des nœuds physiques doivent approuver un accès avant de le faire.

Au niveau de la latence, cette différence rend le NoSQL bien plus rapide que le blockchain. En effet, le NoSQL est organisé en pair de clé-valeur ou orienté document, cela a pour effet de rendre l'accès très efficace, voir instantané. Cependant, le blockchain doit relever la demande à plusieurs nœuds afin d'avoir une majorité d'approbations.

Au niveau de la sécurité, cette chaîne permet de conserver l'intégrité du réseau en assurant la validité de chaque accès. Ainsi le blockchain est bien plus sécuritaire que le NoSQL.

III. Expérimentations

A. *Déploiement d'Hyperledger Fabric*

Afin de réaliser nos tests, nous avons choisi de faire un déploiement dans une instance EC2 d'AWS. Nous avons opté pour une instance de 30 GB stockage SSD, afin de limiter les risques de manque d'espace de stockage et de limiter les impacts de la rapidité de lecture et d'écriture sur un disque dur. C'est une instance de type t2-large qui opère avec Ubuntu 22.04 LTS. La version d'Hyperledger Fabric que nous avons utilisé provient de l'image docker hyperledger/fabric-ccenv:1.4.4.

L'architecture déployée est composée de 2 organisations, d'un peer chacune. Une organisation est, essentiellement, un groupe de travailleurs (peer), qui ont eu les responsabilités d'effectuer les transactions. Les instances de type org n'ont pas énormément de responsabilités dans les tests que nous avons réalisés, puisque notre objectif était simplement d'évaluer le temps de réponse de la base de données comparativement à MongoDB. Puisque MongoDB ne possède pas d'options de ce type, nous n'avons pas exploité cette fonctionnalité. Les peers déployés utilisent les configurations par défaut fournies par l'outil.

Avec cette configuration de déploiement, l'organisation et les instances dockers devraient être organisés comme dans la figure 3.

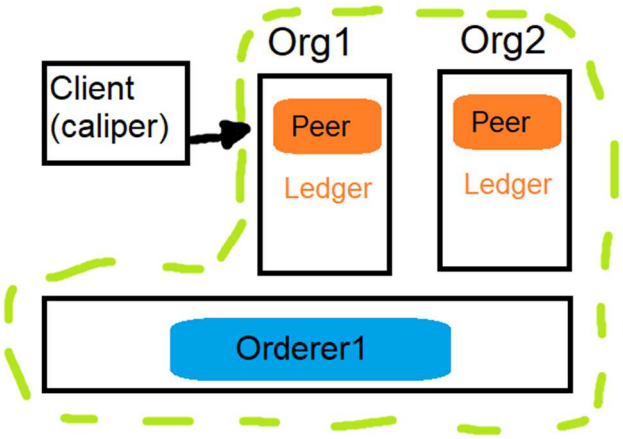

Fig. 3. Diagramme de déploiement de Hyperledger Fabric et Caliper [11]

B. *Déploiement de MongoDB*

Le déploiement de notre base de données MongoDB a été réalisé sur une instance EC2 d'AWS, très semblable à notre instance pour Hyperledger Fabric. La seule différence est que cette nouvelle instance utilise une version ultérieure d'Ubuntu, soit 20.04 LTS, pour des raisons de problèmes de configuration avec une version plus récente d'Ubuntu avec le logiciel de test YCSB. Le déploiement de la base de données a été réalisé avec Docker. Nous avons lancé quatre conteneurs, avec l'outil MongoD. Ainsi, comme il a été mentionné précédemment, nous avons été en mesure de tirer profit des avantages offerts par un ReplicaSet. Un des conteneurs est notre conteneur primaire, qui effectuera toutes les opérations d'écriture, alors que les trois conteneurs restants sont configurés comme des machines secondaires, en lecture et réplication seulement. Ces 4 conteneurs roulent localement sur l'instance EC2.

C. *Tâches et workloads choisis*

Les tâches et workloads utilisés afin de tester les temps de réponse et d'exécution des transactions pour les deux bases de données sont composés de différentes répartitions, permettant d'idéalement trouver les goulots d'étranglement selon les charges de travail. Les paramètres importants que nous avons considérés sont le nombre d'opérations totales de chaque test, le nombre d'éléments dans la base de données au départ du test, le type de distribution des requêtes et les proportions de lecture et d'insertion. Pour tous les tests de MongoDB, le nombre d'opérations lecture/écriture total est de 10 000 opérations et nous avons choisi une taille de 1000 éléments déjà présents dans la base de données et une distribution de type Zipfian. Les paramètres correspondants à modifier pour les différents workloads sont les suivants pour configurer l'outil YCSB :

- recordcount
- operationcount
- readproportion
- insertproportion
- requestdistribution

La configuration du benchmark pour l'outil Caliper se fait, pour sa part, en modifiant le fichier de configuration yaml utilisé pour le déploiement. Pour évaluer les performances de

Hyperledger Fabric, nous avons opté pour un échantillon dix fois plus petit, soit de seulement 1 000 transactions. La raison de ce changement est simplement dû au fait que nous avons remarqué, une fois les tests de MongoDB terminés, que 10 000 transactions étaient clairement trop élevées pour un test de Hyperledger Fabric, qui avait de la difficulté à garder un bon rythme d'exécution. Le temps de latence pour une requête explosait lorsque nous avons testé avec un échantillon de 10 000 transactions, principalement puisque les validations nécessaires pour accepter et prendre en charge les transactions avec les méthodes de validation nécessaires avec une blockchain engorgent le système de 2 peers. Avec le nombre de transactions 10x plus grand, nous avons observé une augmentation de temps de latence d'au moins 100 fois. Nous avons tiré la conclusion qu'un aussi petit réseau ne pouvait pas supporter une aussi grosse charge. Nous n'avons pas corrigé le nombre de transaction dans les tests de MongoDB, car, premièrement, les tests et résultats avaient déjà été compilés avant le début de la configuration de Hyperledger Fabric et Caliper, puis, deuxièmement, car les résultats ne varient pas énormément entre les deux charges différentes. Nous sommes de l'opinion que les résultats offrent tout de même une comparaison valide. Les paramètres à modifier dans les fichiers de configuration seulement les nombres de transactions (TxNumber) pour les tâches Init et Query.

D'abord, nous avons choisi un test simple composé seulement de lectures dans les bases de données. Cela devrait nous permettre de déterminer notre valeur normale de lecture, sans interruption potentielle due à des tâches mixtes.

Ensuite, nous avons choisi un test composé de 75% en lecture et de 25% en écriture. C'est une tâche mixte permettant d'observer un comportement qui ressemblerait plus à une charge typique de la base de données une fois déployée. Lors de l'utilisation réelle d'une base de données, il devrait en général y avoir beaucoup plus de lectures seront effectuées que d'écriture.

Par la suite, nous avons choisi un test composé de 50% en lecture et 50% en écriture. Cela nous permet d'avoir des résultats pour faire une approximation du temps de réponse en lecture et en écriture des deux bases de données en fonction de la charge de travail, c'est un test "au centre" de notre répartition.

Pour suivre notre répartition des tests, le prochain test sera un test de 25% en lecture et 75% en écriture. C'est notre premier workload avec une répartition plus lourde en écriture qu'en lecture et nous nous attendons à des temps de réponse plus élevés que les tests précédents.

Finalement, notre dernier test est un test de charge composé uniquement d'écritures. Cela devrait être le test le plus difficile pour les deux bases de données, puisque l'opération d'écriture nécessite des conditions d'exécutions plus contraignantes qu'une écriture.

Workload	Opération	Pourcentage
100/0	Lecture	100
	Écriture	0
75/25	Lecture	75
	Écriture	25
50/50	Lecture	50
	Écriture	50
25/75	Lecture	25
	Écriture	75
0/100	Lecture	0
	Écriture	10

Fig. 4. Distribution des opérations dans les différents tests

Chaque fichier de configuration et de Workload pour YCSB et les fichiers de configuration pour Caliper sont disponibles sur le GitHub public à l'adresse suivante : https://github.com/sebasdawg/LOG8430-TP3

D. Hébergement de Hyperledger Caliper

Nous avons utilisé la version hyperledger/caliper-cli@0.4 de l'outil Hyperledger Caliper, que nous avons téléchargé grâce au gestionnaire de paquet NPM. Afin de lancer les benchmarks, nous avons utilisé le fichier de configuration yaml de Caliper disponible dans le répertoire d'exemples de benchmarks d'Hyperledger Fabric.

Afin de lancer les différents benchmarks présentés dans la section précédente, nous avons modifié le fichier de configuration yaml utilisé afin d'augmenter les tâches de lecture/écriture selon les besoins du test actuel.

E. Configuration de YCSB

L'outil YCSB ne demande pas énormément de préparation. Nous avons utilisé la version release 0.17.0 disponible sur GitHub, et nous avons été en mesure de lancer directement des tests de type "load" et de type "run" dès que les conteneurs de MongoDB étaient opérationnels.

IV. RÉSULTATS

Afin d'évaluer et de comparer les deux bases de données, nous sommes dans l'obligation de faire des approximations et hypothèses en lien avec les opérations disponibles dans les deux architectures. D'abord, une base de données NoSQL fonctionne par le biais d'opérations : une lecture, une insertion, une mise à jour, une suppression, ce sont toutes des opérations. L'équivalent avec une base de données basée sur les technologies de la blockchain est, dans notre étude, une transaction, du moins pour les opérations de lecture et d'écriture.

Nous ne considérons pas les autres transactions possibles avec MongoDB et avec Hyperledger Fabric, comme les insert/update, car ils n'ont pas tout à fait le même comportement dans les deux cas. Par exemple, une mise à jour avec les technologies de la blockchain nécessite tout de même l'ajout d'un bloc de transaction, alors que dans MongoDB la mise à jour d'un champ est possible dans la même entrée déjà existante.

Les mesures que nous avons retenu pour les tests réalisés sont :

- Latence de lecture
- Latence d'écriture
- Temps total d'exécution
- Temps moyen par opération/transaction
- Nombre d'opérations/transactions par secondes
- Consommation CPU (pourcentage)
- Consommation mémoire (MiB et pourcentage).

A. Analyse des résultats et comparaisons

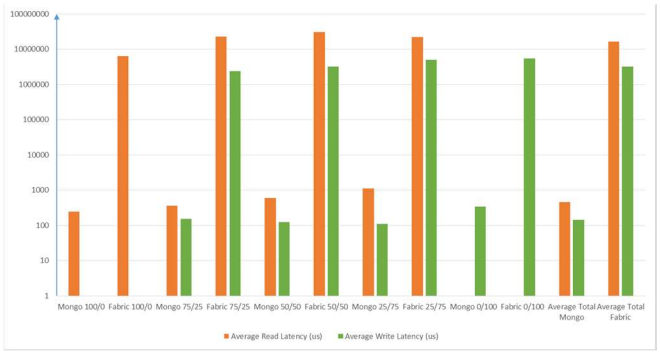

Fig. 5. Résultats de temps de latence (microsecondes) moyen en lecture (orange) et en écriture (vert)

Tout d'abord, à la figure 5 on peut observer que MongoDB est extrêmement rapide comparativement à Hyperledger Fabric au niveau des lectures et des écritures, peu importe le test effectué. Afin de juxtaposer les résultats de MongoDB avec les résultats d'Hyperledger Fabric, une échelle logarithmique a été utilisée. En moyenne, MongoDB effectue ses opérations de lecture 35 000 fois plus rapidement et 22 000 fois pour ses opérations en écriture.

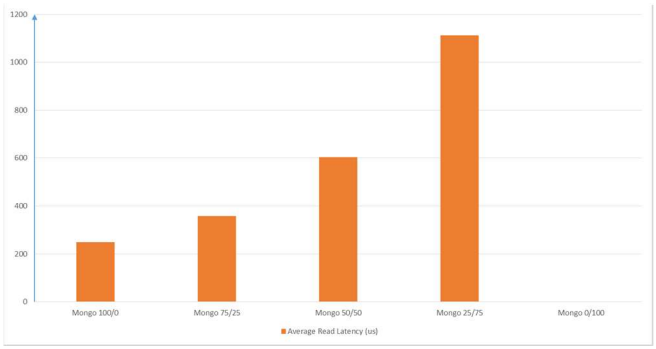

Fig. 6. Résultats de temps de latence (microsecondes) moyen en lecture pour MongoDB

Un comportement à l'opposé de nos attentes est la latence moyenne en lecture réduite, pour la base de données MongoDB, lorsque la charge en lecture augmente, à la figure 6. Ainsi, nous avons des mesures de latence plus basses pour les tests avec 100% de lecture que pour des tests avec seulement 25% de lectures. Une hypothèse que nous pouvons poser est que les accès en lecture, durant un test de 100% de lecture, ne sont jamais interrompus par la réplication du ReplicaSet principal vers les ReplicaSet secondaires. Ainsi, puisque les instances secondaires sont 100% du temps en lecture et jamais en écriture afin d'appliquer les modifications à l'ensemble répliqué, la latence moyenne est plus petite car les ressources sont toujours disponibles et jamais bloquées.

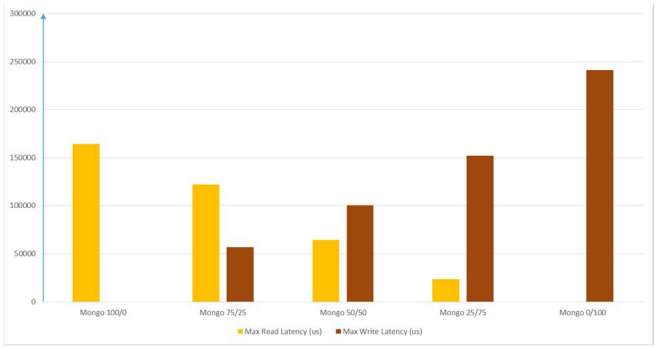

Fig. 7. Résultats de temps de latence (microsecondes) maximal en lecture (jaune) et en écriture (brun) pour MongoDB

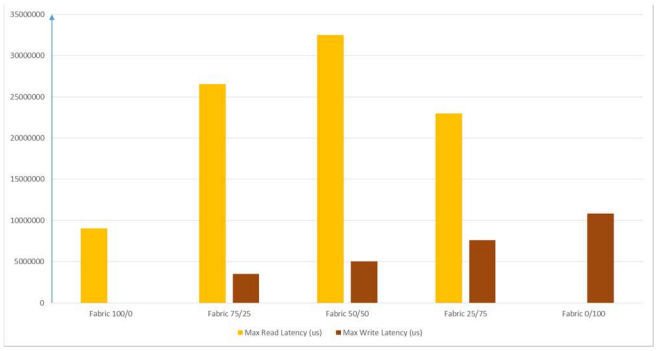

Fig. 8. Résultats de temps de latence (microsecondes) maximal en lecture (jaune) et en écriture (brun) pour Hyperledger Fabric

Les figures 7 et 8 représentent les temps de latence en lecture et en écriture maximaux pour les deux bases de données. En écriture, la tendance pour les deux bases de données est la même : plus le nombre d'écriture augmente, plus le temps de latence maximale augmente. On remarque par contre que, pour la base de données MongoDB, nous obtenons un écart plus grand entre le test de 75% de lecture et 25% d'écriture (faible écriture) et le test de 100% d'écriture qu'avec la base de données Hyperledger Fabric. Nous observons une augmentation de près du double du temps de latence maximal observé à faible charge d'écriture, alors que Fabric augmente seulement de 1,5 fois. Une raison expliquant cette croissance plus forte pour MongoDB est due à la façon dont nous avons déployé nos deux systèmes. MongoDB ne profite que d'une machine de type primaire, qui s'occupe de faire toutes les écritures. Ainsi, une hypothèse très probable est que les requêtes d'écritures sont coincées plus longtemps dans la queue d'exécution de notre instance primaire. Hyperledger profite de deux instances de type peer, qui sont responsables de soumettre les nouvelles écritures au réseau grâce à leurs ledgers. Ainsi, une augmentation du nombre de requêtes n'a pas un impact aussi important sur la latence maximale du système, car il est possible de mieux répartir l'augmentation qu'avec un seul point central d'écriture.

La base de données MongoDB suit la même tendance en écriture et lecture pour ce qui a trait à la latence maximale : Plus la charge est importante, plus la latence maximale augmente. La base de données Fabric est pour sa part très différente en ce qui concerne la latence maximale en lecture. Selon la figure 8, cette valeur maximale semble suivre une loi normale. Nous n'avons pas été en mesure de déterminer une raison logique pour ce comportement, puisque nous n'avons pas été en mesure de trouver des informations en lien avec le partage des ressources des différents peers qui pourrait aider le système de la sorte.

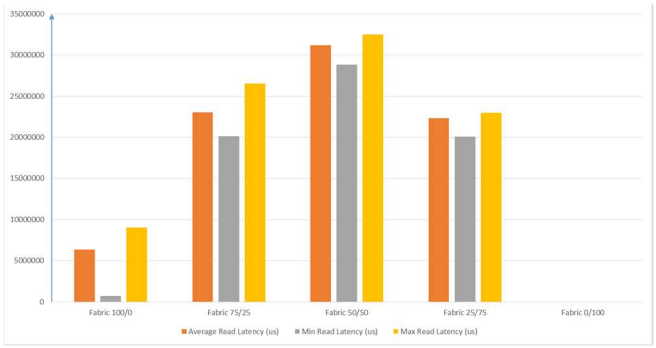

Fig. 9. Résultats de temps de latence (microsecondes) moyen (orange), minimal (gris) et maximal (jaune) en lecture pour Hyperledger Fabric

La figure 9 représente les résultats minimaux et maximaux du temps de latence en lecture pour la base de données Hyperledger Fabric. On constate avec cette figure que l'écart entre les valeurs maximales et minimales est très petit. Cela a pour impact de donner des performances, bien que définitivement moins bonnes, beaucoup plus prévisibles. En effet, avec des écarts de min/max aussi petits, on peut prévoir le temps de réponse beaucoup plus facilement qu'avec des écarts énormes entre les minimums et maximums. C'est d'ailleurs ce cas que l'on rencontre en écriture avec Hyperledger Fabric en écriture et avec MongoDB en lecture et en écriture. Nous avons de très grands écarts entre les valeurs minimales et maximales, qui ne peuvent pas être représentés dans une figure.

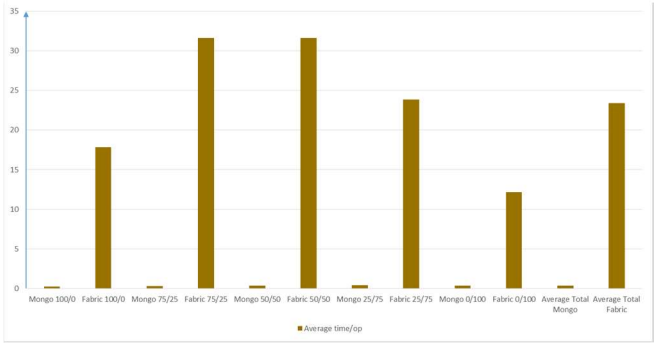

Fig. 10. Résultats de temps de latence (microsecondes) moyen (orange), minimal (gris) et maximal (jaune) en lecture pour Hyperledger Fabric

Sans grande surprise, le temps moyen par opération est extrêmement plus lent pour Hyperledger Fabric, d'un facteur de près de la centaine, visible dans la figure 10.

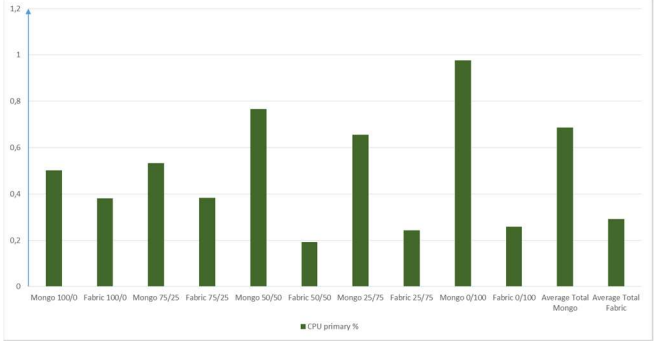

Fig. 11. Résultats du pourcentage d'utilisation moyen du processeur des instances primaires pour MongoDB et Hyperledger Fabric

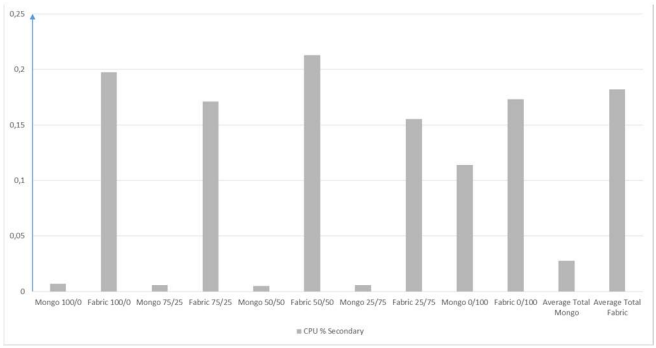

Fig. 12. Résultats du pourcentage d'utilisation moyen du processeur des instances secondaires pour MongoDB et Hyperledger Fabric

Les figures 11 et 12 représentent l'utilisation du processeur par les différentes instances Docker déployées pour MongoDB et Hyperledger Fabric. La figure X+6 contient les performances de l'instance primaire de MongoDB et la moyenne des deux instances de type Org de Fabric. On remarque qu'en moyenne, l'instance primaire de MongoDB est plus sollicitée que les instances Org, puisque l'instance primaire a un rôle beaucoup plus important dans l'architecture générale que les Org. Avec les tests réalisés, nous n'avons pas particulièrement mis de charge sur les Org, puisque notre objectif n'était pas de tester cette fonctionnalité de Fabric, mais simplement de vérifier les performances d'écritures et de lectures sans se soucier des organisations ayant accès à ces ressources. Avec des restrictions sur les organismes ayant droit aux ressources, plus de stress aurait été placé sur les Org. MongoDB utilise l'instance primaire afin de faire toutes les écritures et afin de mettre à jour les ReplicaSet des machines secondaires, donc un haut pourcentage d'utilisation s'aligne avec nos attentes.

La figure 12 affiche les moyennes de pourcentage d'utilisation des trois instances secondaires de MongoDB et des deux instances de type peers d'Hyperledger Fabric. Les instances de Fabric sont beaucoup plus actives que les instances de MongoDB puisque, selon l'architecture, les peers sont responsables de plus d'opérations que les instances secondaires. Les instances secondaires de MongoDB sont responsables seulement des lectures dans leurs ensemble répliqués alors que les peers sont responsables de toutes les lectures, les écritures et les validations de transactions grâce à leurs ledgers.

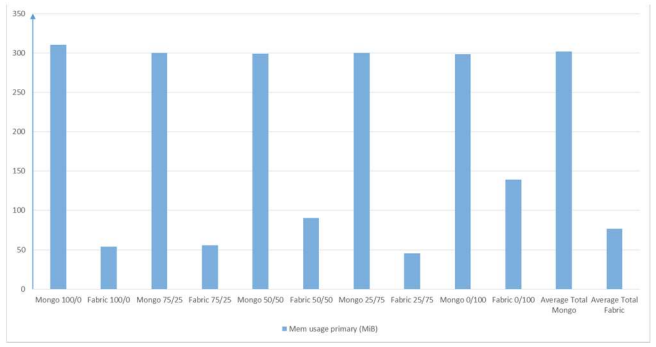

Fig. 13. Résultats de l'utilisation moyenne de la mémoire par les instances primaires pour MongoDB et Hyperledger Fabric

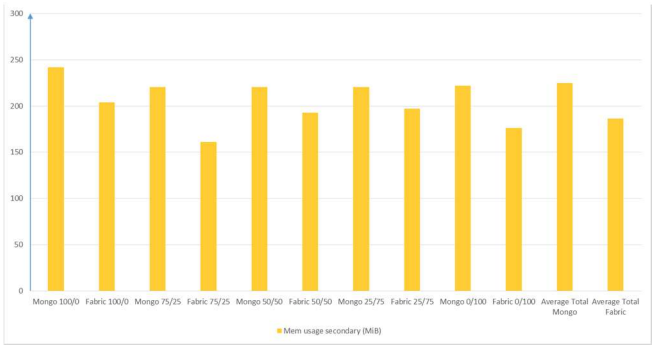

Fig. 14. Résultats de l'utilisation moyenne de la mémoire par les instances secondaires pour MongoDB et Hyperledger Fabric

Finalement, la dernière métrique évaluée pour les deux bases de données est l'utilisation moyenne de la mémoire disponible par chaque type d'instance. À la figure 13, nous observons que Fabric utilise beaucoup moins de mémoire vive que MongoDB pour ses instances primaires, et ce, même lorsque l'instance primaire est moins utilisée avec une tâche de 100% d'écriture. L'instance de type primaire de MongoDB utilise toujours près de de la totalité de la mémoire qui lui est allouée, afin de garder en mémoire des indices récemment utilisés pour le ReplicaSet courant [12]. Ainsi, cela permet d'accéder à ces indices plus rapidement pour les requêtes subséquentes. Hyperledger Fabric ne semble pas suivre tendance et elle utilise donc beaucoup moins de mémoire en cours d'exécution, du moins pour les instances Org. Comme vu précédemment, ces instances ne sont pas les instances effectuant la majorité des opérations dans la base de données, donc une réduction de la mémoire vive utilisée est logique.

Nous observons une répartition semblable pour les deux bases de données dans la figure 14 pour les instances secondaires et les peers, mais moins élevée pour Fabric. Nous avons quelques hypothèses en lien avec cette baisse, dont la taille de l'échantillon utilisé. En effet, les tests avec MongoDB ont été effectués avec 10 000 transactions alors que les tests pour Hyperledger Fabric ont été effectués avec 1000 transactions. Cette diminution de la taille des tests pourrait avoir un impact sur la taille de la cache utilisée par les deux systèmes. MongoDB pourrait garder en mémoire plus d'éléments dans sa cache vu le plus grand nombre de requêtes effectuées dans le court laps de temps du test.

V. Discussion

A. *Limitations et obstacles à la validité*

Évidemment, le premier obstacle à la validité est, comme nous l'avons mentionné précédemment, le nombre de transactions et opérations utilisées pour les deux systèmes lors des tests. Considérant le très grand écart de performances des deux systèmes, que nous avons remarqué dès la réalisation des premiers tests avec Hyperledger Fabric, il nous a semblé raisonnable de limiter le nombre de transactions effectuées pour Hyperledger Fabric. Les tests avec MongoDB, déjà complétés avant le début de ceux avec Hyperledger Fabric, ont été réalisés avec un total de 10 000 opérations. Nous avons choisi délibérément de limiter le nombre de transactions à 1000 pour les tests d'Hyperledger Fabric, à des fins pratiques pour l'équipe, pour des raisons de performance (expliquées précédemment) et puisque les résultats obtenus avec un plus petit échantillon était déjà très révélateurs des différences des deux bases de données.

Un second élément important à considérer pouvant avoir un impact sur les performances globales des deux systèmes est la taille des instances déployées. Nous n'avons pas été en mesure de tester un système d'Hyperledger Fabric déployé dans de grandes organisations avec plus de peers. La blockchain a été conçue, d'abord en avant tout, afin de favoriser l'utilisation d'un réseau distribué de taille significative, et non pour un système composé de cinq petites instances. La mise à l'échelle pourrait définitivement avoir un impact positif sur les opérations, du moins en lecture. Puisque chaque peer est indépendant lors des opérations de lectures, une augmentation du nombre d'instances peer pourrait avoir un impact très positif sur la latence en lecture. Par contre, l'écriture dans un système plus grand pourrait ne pas bénéficier autant de la mise à l'échelle, à cause de sa méthode de consensus. pBFT est une méthode de consensus efficace dans de petits réseaux, car elle demande une validation de tous les nœuds du réseau afin d'accepter une transaction, sans toutefois en avoir besoin immédiatement. Plus le réseau grossit, plus le temps pour accepter une transaction augmentera avec une croissance O(n^k) (n = nombre de messages, k = nombre de nœuds). Pour sa part, MongoDB n'a pas ce même problème grâce à la possibilité de fractionner en plusieurs shards sa base de données et de laisser un nouveau nœud pour écrire toutes les transactions pour ce shard particulier. Cette méthode a aussi ses limitations en écriture, mais ces ralentissements se produisent seulement localement pour une section spécifique de la base de données et non globalement comme dans un réseau de blockchain.

VI. Conclusion

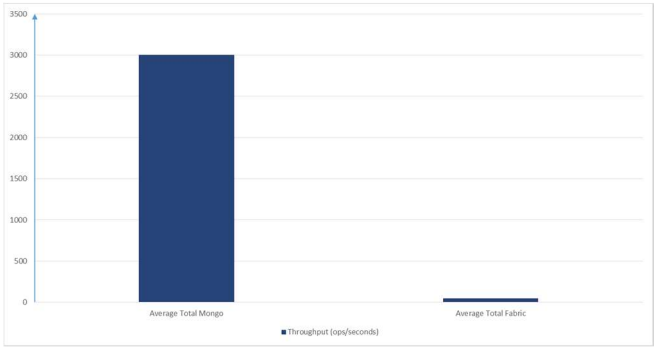

Fig. 15. Nombre d'opérations par secondes moyen au travers des 5 tests pour MongoDB et Hyperledger Fabric

Sans d'énormes surprises, nous pouvons conclure que la base de données Hyperledger Fabric, basée sur les technologies de la blockchain, est très loin d'atteindre les performances d'une base de données de type NoSQL telle que MongoDB. La figure 15 représente parfaitement cette conclusion : 3002 opérations par secondes en moyenne pour MongoDB et 49 opérations par secondes pour Hyperledger Fabric. Une blockchain a de nombreux avantages de sécurité et de protection des données stockés que l'on ne peut pas retrouver dans une base de données non-relationnelle, donc il y a définitivement un argument à faire en faveur d'une architecture distribuée comme Fabric. Le concept d'organisations qu'apporte Fabric offre énormément de souplesse et la possibilité de partager l'infrastructure à plusieurs industries différentes, chose impossible à faire avec une base de données traditionnelle. La performance est définitivement inférieure au niveau de la rapidité d'exécution,

mais une place existe pour une telle architecture dans un domaine dans lequel cette caractéristique n'est pas un enjeu aussi important.

Les bases de données non-relationnelles sont toujours extrêmement plus rapides que ce que peut offrir les bases de données basées sur la blockchain, mais avec l'amélioration des algorithmes de validation des transactions, c'est une solution qui pourra devenir de plus en plus viable. Les deux types de bases de données ne devraient pas être mutuellement exclusives et les deux devraient simplement être considérés dans des situations différentes selon les besoins des clients, afin de répliquer la même coexistence des bases de données relationnelles SQL et non-relationnelles NoSQL.

RÉFÉRENCES

[1] R. Zhang, R. Xue et L. Liu, "Security and Privacy on Blockchain," ACM Comput. Surv., vol. 1, no. 1, Article 1, Jan. 2019, 35 pages. [En ligne]. Disponible: https://doi.org/10.1145/3316481. [Consulté le: 15 avril 2023].

[2] M. Rasolroveicy et M. Fokaefs, "Performance Evaluation of Distributed Ledger Technologies for IoT data registry : A Comparative Study," 2020 Fourth World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4), Londres, Royaume-Uni, 2020, pp. 137-144, doi: 10.1109/WorldS450073.2020.9210358.

[3] Hyperledger, "Introduction to Hyperledger," Hyperledger, 2018. [En ligne]. Disponible: https://www.hyperledger.org/wp-content/uploads/2018/08/HL_Whitepaper_IntroductiontoHyperledger.pdf. [Consulté le: 15 avril 2023].

[4] GeeksforGeeks, "Practical Byzantine Fault Tolerance (pBFT)," GeeksforGeeks, [En ligne]. Disponible: https://www.geeksforgeeks.org/practical-byzantine-fault-tolerancepbft/. [Consulté le: 15 avril 2023].

[5] The Motley Fool, "Proof of Work," The Motley Fool, [En ligne]. Disponible: https://www.fool.com/investing/stock-market/market-sectors/financials/cryptocurrency-stocks/proof-of-work/. [Consulté le: 15 avril 2023].

[6] P. Adhav, "Transaction Flow in Hyperledger Fabric: Basic Network Fabric v2.0," Cours en ligne (6 mai 2020), Disponible: https://www.youtube.com/watch?v=4ylRWcNNTII [Consulté le: 16 avril 2023].

[7] MongoDB, Inc., "Introduction," MongoDB, Inc., [En ligne]. Disponible : https://www.mongodb.com/docs/manual/introduction/. [Consulté le: 15 avril 2023].

[8] MongoDB, Inc., "Sharding," MongoDB, Inc., [En ligne]. Disponible : https://www.mongodb.com/docs/manual/sharding/#std-label-sharding-introduction. [Consulté le: 15 avril 2023].

[9] MongoDB, Inc., "Storage," MongoDB, Inc., [En ligne]. Disponible : https://www.mongodb.com/docs/manual/storage/. [Consulté le: 15 avril 2023].

[10] Aditi, "MongoDB Architecture and Characteristics," Medium, TechieAhead (2 février 2023), [En ligne], Disponible: https://medium.com/techieahead/mongodb-architecture-and-characteristics-32a3798c2b49. [Consulté le: 15 avril 2023].

[11] Kim, J.; Lee, K.; Yang, G.; Lee, K.; Im, J.; Yoo, C. QiOi: Performance Isolation for Hyperledger Fabric. Appl. Sci. 2021, 11, 3870. https://doi.org/10.3390/app11093870.

[12] MongoDB, Inc., "WiredTiger Storage Engine," MongoDB, Inc., [En ligne]. Disponible : https://www.mongodb.com/docs/manual/core/wiredtiger/#memory-use. [Consulté le: 15 avril 2023].