Comparaison entre NoSQL et Blockchain en tant que bases de données distribuées

Dylan Batista-Moniz *dep. Informatique et Logiciel*
Polytechnique Montréal
Montréal, Canada
Nazim Azeli *dep. Informatique et Logiciel*
Polytechnique Montréal
Montréal, Canada
Vy-Phuong Nguyen *dep. Informatique et Logiciel*
Polytechnique Montréal
Montréal, Canada
Céline Ly *dep. Informatique et Logiciel*
Polytechnique Montréal
Montréal, Canada

***Abstrait*—Le but de cet article est de montrer les différences de performances entre des bases de données traditionnelles et celles distribuées plus précieusement entre MongoDB et Hyperledger Fabric. Afin d'arriver à des conclusions significatives, trois charges de travail contenant des pourcentages d'écriture et lecture différente ont été utilisées. Deux classes de métriques de performances ont donc été incorporées afin de faciliter cette analyse. La première se rattache à la latence du système face à différentes charges de travail et la seconde à l'utilisation des ressources. On constate alors que MongoDB est plus performant en termes de latence et garde une utilisation de ressource assez constante quant aux écritures et lectures, alors que pour HyperLedger les lectures des transactions ajoutent une charge importante au système. Cette différence est facilement expliquée par l'architecture de chacune de ses technologies.**

***Mots-clés*—NoSQL, Blockchain, Hyperledger Fabric, MongoDB, Analyse, Performance, Base de données**

I. Introduction

Avec l'évolution exponentielle des technologies de nos jours, les applications modernes génèrent de gros volumes de données. Ce phénomène entraîne alors un besoin de trouver des moyens innovants pour stocker et accéder à ces immenses données de façon efficace et optimale. Les bases de données traditionnelles ne semblent pas suffisantes pour répondre aux demandes élevées de transactions des utilisateurs. C'est pourquoi les développeurs s'intéressent de plus en plus aux bases de données distribuées. Les bases de données NoSQL commencent à gagner en popularité. Les bases de données distribuées permettent de stocker des données sur différents sites. Elles favorisent alors l'évolutivité des données. En d'autres termes, elle offre une bonne gestion des demandes accrues. C'est donc le principal avantage d'utiliser ce type de système. [1]

Cet article a pour objectif d'analyser et d'évaluer les performances d'une base de données NoSQL standard telle que MongoDB et de la comparer à celle d'un système de *blockchain* Hyperledger Fabric. Après avoir effectué les tests d'analyse de performance, il sera alors possible de répondre à la question suivante :

1) Comment le système de *blockchain* Hyperledger Fabric se compare à une base de données MongoDB en termes de performances, si nous ignorons les avantages de sécurité ?

Le présent article sera divisé en plusieurs sections. La section 2 introduira une brève description des principales caractéristiques d'Hyperledger Fabric et de MongoDB. La section 3 décrira les espaces de travail utilisés pour réaliser l'expérience et montrera les résultats obtenus. L'analyse des résultats des performances pour chaque plateforme sera discutée dans la section 4. Pour terminer, la conclusion du travail sera présentée dans la section 5.

II. Contexte - Background

A. Hyperledger Fabric

Hyperledger Fabric est une technologie qui est basée sur le *blockchain*. Le *ledger* est en fait un record de consensus maintenu et validé par des nœuds. La

décentralisation du *ledger* permet d'avoir plusieurs participants dans le réseau et il est aussi possible pour ceux-ci d'avoir leur identité vérifiée.

Les aspects notables de Hyperledger Fabric sont la modularité, l'utilisation de permission et la confidentialité. La modularité permet aux entreprises qui l'utilisent de pouvoir configurer Hyperledger Fabric à leur besoin. Un exemple de ça est le support des DBSM pouvant être facilement atteignable à travers des requêtes à l'intérieur du *ledger*. Il est possible de sauvegarder des données sous le format json dans la CouchDB qui est une une base de données NoSQL. Le *ledger* contient donc une base de données permettant de représenter son état (*state Database*).

L'utilisation des permissions permet d'offrir à la fois un *blockchain* admettant n'importe qui à y participer et où chaque participant est anonyme tandis qu'un *Permissioned blockchain* admet des participants déjà identifiés. Ceci augmente le niveau de confiance.

La confidentialité peut être assurée grâce à l'architecture des canaux et de l'utilisation de données privées. Il est possible pour les participants du réseau de définir un sous-réseau ou chacun des membres aura une visibilité réduite à une certaine collection de transactions. Seuls les nœuds participant à ce canal auront donc accès au contrat et aux données de la transaction rattachée. Les données privées permettent l'accès à des collections entre membres du même canal, assurant ainsi la même confidentialité sans le besoin de créer et maintenir un canal séparé. [2]

B. MongoDB

Le terme *NoSQL* fait souvent référence à tout autre système alternatif aux bases de données SQL traditionnelles. [3] En fait, contrairement au SQL, NoSQL ne stocke pas les données sous forme de tables relationnelles. [5]

Depuis NoSQL, il y a quatre types de bases de données qui existent aujourd'hui:

- **La base de données documentaires :** stockent les données dans des documents similaires aux objets JSON.
- **La base de données clé-valeur :** chaque élément contient des clés et des valeurs.
- **La base de données à colonnes larges :** stockent les données dans des tableaux, des lignes et des colonnes.
- **La base de données de graphes :** stockent les données dans des nœuds et des arêtes.

Pour répondre à notre question de recherche, la base de données NoSQL utilisée est MongoDB. C'est une base de données documentaire reconnue pour sa flexibilité et son évolutivité. Les données sont stockées sous forme de documents en format JSON. Ces documents sont regroupés dans des collections.

C. Comparaison entre Hyperledger Fabric et MongoDB

Il y a six caractéristiques importantes à comparer entre les bases de données de Hyperledger et de MongoDB. Ces caractéristiques sont l'intégrité des données, les transactions, la performance des requêtes, les autorités, l'architecture et le coût.

a) ***Intégrité des données****:* Hyperledger : Il est quasi impossible de modifier les données dans un *blockchain* sans briser la chaîne.
MongoDB : Un attaquant malveillant peut modifier les données si des mesures de prévention ne sont pas établies.

b) ***Transactions****:* Hyperledger : Les données peuvent seulement être lues ou ajoutées (écrites) à la *blockchain*, mais jamais modifiées. Si un utilisateur veut changer une transaction antérieure stockée dans un bloc, il doit simplement ajouter une nouvelle transaction dans un nouveau bloc avec les modifications sans pouvoir changer la transaction antérieure.
MongoDB : Il est possible de créer, de lire, de mettre à jour ou de supprimer les données.

c) ***Performance des requêtes****:* Hyperledger : La performance d'un *blockchain* peut être ralentie par les méthodes de vérification qui servent à assurer l'intégrité de données.
MongoDB : Les données peuvent être accédées presque instantanément.

d) ***Autorité****:* Hyperledger : La structure est entièrement décentralisée et elle ne dépend sur aucune forme d'autorité centrale. Elle offre ainsi une transparence totale.
MongoDB : Les bases de données sont gouvernées de façon centralisée, donc seul un administrateur possède les privilèges pour contrôler et manipuler les données. En conséquence, les bases de données n'offrent pas de transparence. [4]

e) ***Architecture****:* Hyperledger : Il utilise un réseau de *ledger* distribué.
MongoDB : Les bases de données utilisent une architecture client-serveur.

f) ***Coût****:* Hyperledger : Il est très coûteux d'implémenter et de maintenir un système de *blockchain* autant en termes de financement qu'en termes d'énergie et de ressources.

MongoDB : Les bases de données sont relativement faciles à implémenter et à maintenir étant une ancienne technologie. [6]

III. Expérimentations

A. Répertoire GitHub

Le répertoire contenant les scripts d'exécution des tests se retrouve ici:

https://github.com/DyBat/LOG8430-TP3-Whiskey

B. Déploiement Hyperledger Fabric

Le déploiement de Hyperledger Fabric a été fait par une instance AWS EC2. Cette instance a été du type t2.large, qui possède un processeur de 2 cœurs virtuels, 8 GO de mémoire vive et avec 30 GO de stockage. Cette configuration possède assez de puissance pour les tests qui ont été choisis. Le système d'exploitation choisi pour cette instance a été Ubuntu 20.04 LTS. Les configurations de réseau de cette instance EC2 permettent tout trafic entrant et sortant. Le système Hyperledger Fabric a été déployé sur Docker. Le réseau du *blockchain* a été configuré avec 2 *peers* et 2 organisations. Pour mesurer la performance d'Hyperledger Fabric, l'outil Hyperledger Caliber a été utilisé. Celui-ci permet de mesurer la latence, le débit de lecture et de transaction et les métriques de consommation de ressources (CPI, mémoire, IO réseau).

C. Déploiement MongoDB

Le déploiement de notre base de données MongoDB a été fait sur une seconde instance AWS EC2 qui avait la même configuration que l'instance EC2 que not environnement pour Hyperledger Fabric, soit une instance du type t2.large avec 30 GO de stockage avec l'environnement Ubuntu 20.04 LTS. La version de MongoDB utilisée lors de l'expérimentation est la version 5.0. Les configurations de réseau de cette instance EC2 permettent tout trafic entrant et sortant. Dans cette instance EC2, MongoDB a été déployé à l'aide de Docker. La version de l'image Mongo correspond à la version 5.0. Le déploiement a été possible par l'usage d'un fichier docker-compose.yml contenant les instructions nécessaires pour la configuration de l'environnement et du déploiement des services pour MongoDB. L'outil Yahoo! Cloud Serving Benchmark (YCSB) a été installé sur l'instance afin de pouvoir évaluer les performances de MongoDB en termes de latence et le débit de lecture et métriques de consommation de ressources. L'usage de conteneurs Docker pour MongoDB à fait en sorte qu'il a fallu faire des configurations au niveau réseautage de l'instance afin de pouvoir accéder aux conteneurs par YCSB.

D. Description des tâches et des workloads sélectionnés

Pour Hyperledger, les configurations associées aux différentes charges utilisées par Caliper sont présentées dans le tableau suivant.

TABLE I
Description des workloads pour Hyperledger.

Nom	Caliper Workload A	Caliper Workload B	Caliper Workload C
Nombre de workers	5	5	5
txNumber	N/A	500	945
txDuration	40	20	4
tps	25	25	25
Proportion de lectures	100%	50%	10%
Proportion d'écriture	0%	50%	90%

En ce qui concerne les *workloads* choisis pour les métriques capturées par YCSB pour MongoDB, nous avons créé 3 *workloads*. Ces configurations sont expliquées dans le tableau suivant.

TABLE II
Description des workloads pour MongoDB.

Nom	Mongo Workload A	Mongo Workload B	Mongo Workload C
Nombre d'enregistrements	1000	1000	1000
Nombre d'opérations	1000	1000	1000
Proportion de lectures	100%	50%	10%
Proportion de mise à jour de documents	0%	50%	90%
Proposition de scans	0%	0%	0%
Proposition d'insertions	0%	0%	0%

Pour les deux systèmes Workload A correspond à une configuration qui ne contient que des lectures de documents sur la base de données. Workload B correspond à

une configuration qui contient un nombre égal de lectures et de mises à jour de documents sur la base de données. Workload C correspond à une configuration qui contient presque que des mises à jour de documents.

E. Configuration Hyperledger Caliper

Pour la configuration de l'outil Hyperledger Caliper, cela nécessite d'avoir la version 2.7 de Python. On a tout d'abord cloné le répertoire GitHub "Hyperledger Caliper Benchmarks" qui contient des exemples de benchmark qui seront essentiels pour lancer nos tests de performance désirés. Le référentiel git utilisé est "d02cc8bbc17afda13a0d3af1122d43bfbfc45b0a".

Dans ce répertoire, on utilise principalement les fichiers de configuration fournis dans le dossier benchmarks/samples/fabric/marbles/. On y retrouve un fichier nommé config.yaml qui servira comme configuration pour notre benchmark et deux fichiers .js : query.js et init.js. Ces deux fichiers représentent respectivement les modules de charges de travail pour les lectures et les écritures.

Pour la configuration réseau, le fichier a été généré dans le dossier caliper-benchmarks/networks/fabric/config_solo_raft/ en lançant la commande ./generate.sh.

Certains autres prérequis doivent être installés avant de pouvoir rouler le benchmarking. Cependant, nous avons rédigé un script caliper_script.sh afin de simplifier le processus de configuration.

Une fois que la configuration a été complétée, il faudra installer l'outil Hyperledger Caliper dans le répertoire courant avant d'exécuter nos tests de benchmark. Pour ce faire, on a installé la version 0.4 de Caliper CLI. Par la suite, on a lancé la commande npx caliper launch manager en mode root en indiquant le fichier de configuration et le profil de connexion réseau utilisés :

sudo npx caliper launch manager --caliper-workspace . --caliper-benchconfig benchmarks/samples/fabric/marbles /config.yaml --caliper-networkconfig networks/fabric/v1 /v1.4.4/2org1peercouchdb_raft/fabric-go-tls-solo.yaml

Afin d'automatiser le processus de configuration et d'exécution des tests, le script caliper_script.sh mentionné plus haut inclut également l'exécution des tests et la sauvegarde des résultats dans un dossier nommé results.

F. Configuration Yahoo! Cloud Serving Benchmark (YCSB)

Cette configuration sous-entend que l'environnement sur lequel le système est installé, soit AWS, contient tous les paquets nécessaires. Pour télécharger l'outil YCSB, la commande suivante a été lancée:

curl -O --location https://github.com/brianfrankcooper/YCSB/releases/ download/0.17.0/ycsb-0.17.0.tar.gz

YCSB nécessite une liste de prérequis avant de pouvoir rouler un benchmarking. Cependant, nous avons rédigé un script mongo_script.sh afin de simplifier le processus de configuration.

Ensuite, nous pouvons exécuter nos tests par le lancement de la commande suivante dans le dossier contenant YCSB:

./bin/ycsb load mongodb -s -P ../Mongo/workloads_custom/workload_custom_a -p recordcount=1000 -p mongodb.upsert=true -p mongodb.url=mongodb://primary:27017,secondary1:27017, secondary2:27017,secondary3:27017 /?replicaSet=myReplicaSett

Afin d'automatiser le processus de configuration et d'exécution des tests, le script mongo_script.sh mentionné plus haut inclut également l'exécution des tests et la sauvegarde des résultats dans un dossier nommé results.

Le tableau suivant présente les latences pour les différents *workloads* de Hyperledger.

TABLE III
LATENCES DU SYSTÈME HYPERLEDGER.

	Caliper Workload A		Caliper Workload B		Caliper Workload C	
Name	Read Assets	Write Assets	Read Assets	Write Assets	Read Assets	Write Assets
Succ	1005	-	505	500	945	105
Fail	0	-	0	0	0	0
Send Rate (TPS)	25.1	-	25.2	25.2	25.1	26.1
Max Latency (s) (TPS)	0.35	-	27.61	0.66	0.87	9.73
Min Latency (s)	0.01	-	1.57	0.18	0.18	3.06
Avg Latency (s)	0.04	-	20.32	0.4	0.38	8.37
Throughput (TPS)	25.1	-	12.9	25	25	8.6

Le tableau suivant présente les utilisations de ressources pour les différents *workloads* de Hyperledger.

TABLE IV
UTILISATIONS DES RESSOURCES DU SYSTÈME HYPERLEDGER.

	Caliper Workload A		Caliper Workload B		Caliper Workload C	
Name	Read Assets	Write Assets	Read Assets	Write Assets	Read Assets	Write Assets
CPU (%)	19.14	-	89.13	26.55	16.37	85.78
Memory (%)	0.55	-	3.86	0.56	1.6	1.92
Network Data Received	64.6 kO	-	1.14 MO	299 kO	1.42 MO	1.42 MO
Network Data Data Transmitted	73 kO	-	5.69 MO	332 kO	5.98 MO	5.98 MO
Data written to block	0 b	-	2.97 MO	2.97 MO	0 O	0 O
Data read from block	1.11 MO	-	5.89 MO	3.26 MO	10.4 MO	10.4 MO

Le tableau suivant présente les latences pour les différents *workloads* du système MongoDB.

TABLE V
LATENCES DU SYSTÈME MONGODB.

	MongoDB Workload A		MongoDB Workload B		MongoDB Workload C	
Name	Read Assets	Write Assets	Read Assets	Write Assets	Read Assets	Write Assets
Succ	1000	-	478	522	102	898
Fail	0	-	0	0	0	0
Max Latency (s)	0.15	-	0.13	0.021	0.22	0.069
Min Latency (s)	0.00043	-	0.00043	0.0048	0.00054	0.0054
Avg Latency (s)	0.00095	-	0.0012	0.0076	0.0038	0.011
Throughput (TPS)	578.7	-	190.25	190.25	88.52	88.52

Le tableau suivant présente les utilisations de ressources pour les différents *workloads* du système MongoDB.

TABLE VI
UTILISATIONS DES RESSOURCES DU SYSTÈME MONGODB.

	MongoDB Workload A		MongoDB Workload B		MongoDB Workload C	
Name	Read Assets	Write Assets	Read Assets	Write Assets	Read Assets	Write Assets
CPU (%)	24.45	-	29.06	32.04	30.46	33.34
Memory (%)	3.12	-	3.06	3.01	3.03	3
Network Data Received	69.6 MO	-	64.7 MO	53.5 MO	54.4 MO	56.2 MO
Network Data Data Transmitted	56 MO	-	56.2 MO	46.25 MO	52.25 MO	49 MO
Data written to block	1.53 MO	-	64.1 MO	1.34 MO	1.43 MO	1.47 MO
Data read from block	261 MO	-	345 MO	306 MO	329 MO	306.75 MO

Le graphique suivant présente les latences des systèmes Hyperledger et MongoDB selon le *workload* lors de l'écriture de données.

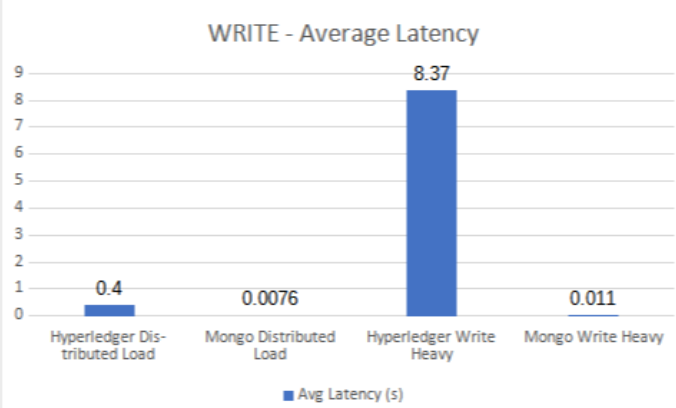

Fig. 1. Latences moyennes des deux systèmes lors des écritures.

Le graphique suivant présente les latences des systèmes Hyperledger et MongoDB selon le *workload* lors de la lecture de données.

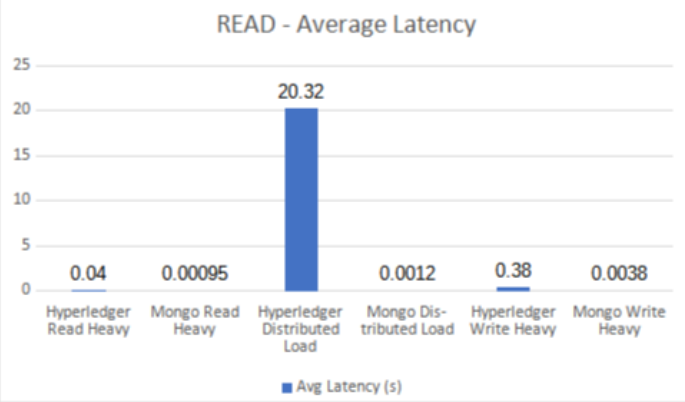

Fig. 2. Latences moyennes des deux systèmes lors des écritures.

Le graphique suivant présente l'utilisation des ressources des systèmes Hyperledger et MongoDB selon le *workload* lors de la l'écriture de données.

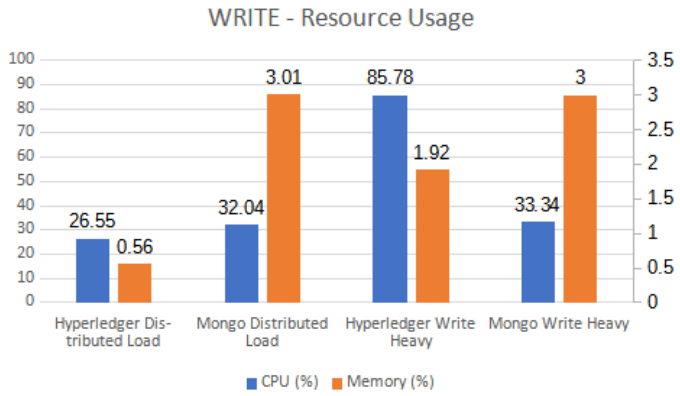

Fig. 3. Latences moyennes des deux systèmes lors des écritures.

Le graphique suivant présente l'utilisation des ressources des systèmes Hyperledger et MongoDB selon le workload lors de la lecture de données.

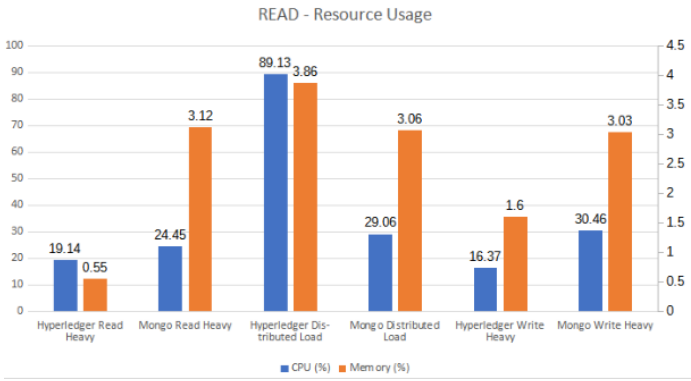

Fig. 4. Latences moyennes des deux systèmes lors des écritures.

IV. Discussion

A. Workload A (Lecture seulement)

Pour la charge de travail de lecture unique, on constate que la latence de lecture est plus élevée dans la base de données Hyperledger Fabric. En effet, la latence maximale détectée par l'outil Caliper est de 0.35 seconde, tandis que celle observée par l'outil YCSB est de 0.15 seconde. Or, la base de données MongoDB semble consommer plus de ressources que le système de *blockchain*. Au niveau du CPU et de la mémoire, MongoDB en utilise respectivement 24.45% et 3.12%, alors que Hyperledger Fabric en consomme 19.14% et 0.55%. De plus, le débit de lecture est beaucoup plus rapide sur MongoDB. On remarque alors que pour cette charge de travail précise, la base de données NoSQL est plus performante que la plateforme de *blockchain*.

B. Workload B (50/50 lecture/écriture)

Concernant la charge de travail équitablement répartie entre les lectures et les écritures (50/50), on aperçoit des résultats de latence similaires à ceux qu'on a vus pour la charge de travail précédente. Effectivement, le délai des transactions de lecture et d'écriture sur MongoDB est inférieur à celui sur Hyperledger Fabric. Toutefois, cette fois-ci, on remarque que la consommation de CPU pour la lecture est très élevée pour Hyperledger Fabric. Celui-ci en consomme 89.13% alors que MongoDB n'en utilise que 29.06%. Cette différence est très significative. Pour ce qui a trait à la lecture, MongoDB semble utiliser un peu plus de CPU que l'autre système. Pour l'utilisation de la mémoire, les résultats sont presque identiques pour les deux systèmes lors des lectures. La différence réside dans les transactions des écritures sur les bases de données. MongoDB utilise 3.01% de la mémoire alors que Hyperledger Fabric en utilise 0.56%. Les débits de lecture et d'écriture restent encore rapides pour MongoDB.

C. Workload C (10/90 lecture/écriture)

Pour le dernier test de performance, on a utilisé une charge de travail ayant une répartition de 10% en lecture et 90% en écriture. Encore une fois, les transactions de lecture et d'écriture sur MongoDB sont beaucoup plus rapides, car la latence de lecture et d'écriture sont plus petites. Aussi, tout comme les résultats observés précédemment, les débits des transactions sont plus performants sur la base de données NoSQL. Pour l'utilisation des ressources, on constate que le système de *blockchain* consomme plus de CPU lors de ses transactions d'écriture que MongoDB, mais ce dernier en utilise davantage lors de ses lectures.

D. Remarques générales

Nous pouvons constater que la consommation de mémoire pour le système MongoDB est plus constante que sur le système Hyperledger. En effet, pour tous les workloads, la consommation de mémoire de MongoDB se situe entre 3.00% et 3.12% en écriture et lecture, et qu'il n'y a pas présence de pics inhabituels dans cette consommation de mémoire. Du côté de Hyperledger, la consommation de mémoire n'est pas constante dans tous les workloads et il y a présence de sauts en usage de mémoire selon les workloads et les actions en jeux.

E. Justification des différences

En bref, après avoir exécuté et analysé les métriques de performance, la base de données MongoDB semble être une base de données distribuée plus performante que la plateforme Hyperledger Fabric. Les transactions des lectures et des écritures sont plus rapides car les délais sont plus petits et les débits sont plus élevés. La consommation du CPU semble presque similaire pour les deux systèmes. Cependant, MongoDB utilise plus de mémoire, mais son usage est similaire pour tous ces workloads, tandis que l'usage de mémoire pour Hyperledger varie grandement selon le workload.

La raison pour laquelle le système Hyperledger utilise moins de mémoire peut être dû au fait que MongoDB réserve et utilise une portion de la mémoire vive disponible pour le *storage engine*. [7] Il est également possible de constater que Hyperledger utilise généralement plus de mémoire lorsque le workload contient des opérations d'écriture et encore plus lorsqu'il contient un mixe d'opérations de lecture et écriture. Ceci peut être dû au fait que chaque opération d'écriture sur le système Hyperledger demande à ce que le *blockchain* log du système soit mis à jour et que l'état de la base de données soit mis à jour sur la *state database* (CloudDB). De plus, lors d'opérations de lectures, le client doit confirmer que l'état qu'il connaît est le même que l'état le plus récent sur la *state databas*e. Si celui-ci est différent, les membres du réseau doivent *query* le nouvel état. Ceci est tout de même plus efficace que les *blockchains* traditionnelles, telles que la *blockchain* du bitcoin, qui nécessite que l'état de la *chain* soit calculé en passant par toutes les transactions du *ledger*.

De plus, les latences élevées pour Hyperledger peuvent être expliquées par le temps nécessaire pour les mesures de sécurité du *ledger*. Soit l'encryption des données au niveau du peer et la vérification par canaux privés pour que seuls les nœuds ayant le droit d'accès aux données en jeux puissent y faire requête.

De plus, les latences peuvent également être dues à l'algorithme du protocole de consensus utilisé lors de transactions sur la *blockchain* du système.

Le temps additionnel nécessaire pour la mise à jour du *blockchain* log vient également impacter le temps de latence du système Hyperledger. En effet, les résultats montrent que la latence du système Hyperledger augmente grandement lors d'une écriture dans un workload avec un nombre élevé d'écritures. Ceci a du sens lorsque nous considérons que le *ledger* doit constamment mettre à jour l'historique des transactions lors d'une nouvelle transaction, soit lors d'une écriture, ce qui augmente le temps requis pour l'action. Donc, un nombre élevé d'écritures impacte la latence du système Hyperledger.

La latence du système Hyperledger est également impactée par la mise à jour dans la *state database* du système. En effet, lors d'une lecture dans un workload

avec un nombre d'écritures et de lectures égaux, la latence du système augmente grandement. Ceci fait du sens, car lors d'une écriture, l'état de la *blockchain* est modifié au niveau de la *state database*, puis lors d'une lecture, le *node* faisant la requête de lecture doit confirmer que l'état qu'il connaît est le plus récent. Si l'état n'est pas le bon, il doit être mis à jour afin de compléter la lecture. Ceci augmente la latence lors d'une lecture dans un workload qui contient un grand nombre d'écritures.

La raison pour laquelle la consommation de mémoire de Hyperledger n'est pas constante comme celle de MongoDB à travers tous les workloads et actions est que la consommation de mémoire de MongoDB est dû à son architecture. En effet, MongoDb alloue un espace de mémoire vive fixe pour son *storage engine*, soit WiredTiger, qui gère le stockage et accès de données en mémoire cache, vive et sur disque. L'espace de base alloué pour WiredTiger est de 256 MO. [7] Cette valeur correspond environ à la quantité de mémoire utilisée par notre système MongoDB dans tous les workloads testés. En effet, notre instance AWS possède 8 GO de mémoire vive et 3% de 8 GO équivaut à environ 240 MO d'alloué pour WiredTiger.

F. Limitations

Les limitations sont liées aux outils utilisés pour analyser la performance des différentes bases de données . En effet, la méthode utilisée pour la capture de la consommation de ressources, soit l'exécution de la commande docker stats, n'a pas été la plus efficace. Cette commande affiche le données des ressources utilisées par des conteneurs Docker à un intervalle de secondes spécifié. Cette commande a été intégrée au script de déploiement des instances de manière à ce que les données soient écrites dans un fichier CSV. Le problème avec cette manière de faire est que l'intervalle d'affichage des données est fixe et qu'il se peut que les vraies données maximales des conteneurs visés aient été manquées.

Il faudrait également noter que les données de latence peuvent être influencées à un certain degré par de la latence au niveau des conteneurs Docker utilisés pour exécuter les services. Alors, il serait bien de déterminer la latence ajoutée par les conteneurs afin de déterminer les vraies latences de Hyperledger et MongoDB.

De plus, le fait que nos instances sont déployées sur AWS du type t2.large signifie que nos systèmes roulent sur des machines virtuelles. Il aurait été intéressant de refaire nos expériences sur des serveurs *bare metal*, pour éliminer toute latence ou manque d'optimisation causé par la virtualisation de l'environnement des systèmes.

V. Conclusions

Pour conclure, nos expériences démontrent que, en termes de latences, MongoDB est plus performant que Hyperledger Fabric. L'utilisation de mémoire vive de MongoDB est plus élevée que celle de Hyperledger, mais ne fluctue pas selon les workloads, tandis que Hyperledger n'est pas consistent dans son usage de mémoire vive.

Le choix entre Hyperledger Fabric et MongoDB dépend selon les besoins d'un projet. Pour un projet nécessitant de la latence basse et une utilisation de ressource consistent dans toutes les situations, MongoDB est le meilleur choix. Cependant, si un projet nécessite un degré de sécurité plus élevé, plus d'intégrité au niveau des données, sans avoir à tant se soucier de la performance du système, Hyperledger Fabric est le meilleur choix.

Il serait intéressant de refaire les expériences de ce rapport sur un environnement *bare metal*, avec un nombre d'opérations plus élevé de 1000 et avec plus de variations de workloads.

REFERENCES

[1] Ahmet Ercan Topcu, Aimen Mukhtar Rmis, ”Analysis and evaluation of the riak cluster environment in distributed databases,” Computer Standards Interfaces, Volume 72, 2020, 103452, ISSN 920-5489, (En ligne) https://doi.org/10.1016/j.csi.2020.103452.

[2] HYPERLEDGER FABRIC, “Introduction,” Hyperledger Fabric Docs Release 2.5, (En ligne) https://hyperledger-fabric.readthedocs.io/en/release-2.5/whatis.html.

[3] MongoDB, “Types of Databases,” (En ligne) https://www.mongodb.com/databases/types.

[4] MongoDB, “Blockchain Database: A Comprehensive Guide,” (En ligne) https://www.mongodb.com/databases/blockchain-databases.

[5] MongoDB, “What is NoSQL,” (En ligne) https://www.mongodb.com/nosql-explained.

[6] 101BLOCKCHAINS.COM, “Blockchain VS Database,” Blockchain Infographics, 2019, (En ligne) https://101blockchains.com/wp-content/uploads/2019/04/Blockchain-vs-Database.jpg.

[7] MongoDB, “WireTiger Storage Engine,” (En ligne) https://www.mongodb.com/docs/manual/core/wiredtiger/.