POLYTECHNIQUE MONTRÉAL
UNIVERSITÉ D'INGÉNIERIE

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

Xavier Dupuis, William Lévesque, Marie Noël et Mathieu Céraline, *Polytechnique Montréal*

***Abstrait*—Ce document décrit les différences quantitatives et qualitatives entre une base de données NoSQL (Cassandra) et une base de données Blockchain (Hyperledger Fabric) du point de vue de l'architecture des bases de données distribuées. L'évaluation est réalisée par le biais des outils YCSB et Hyperledger Caliper pour trois différents types de charges en lecture et en écriture. La comparaison s'appuie sur des variables comme le taux de traitement du système, le temps d'exécution, le nombre d'erreurs, la latence, consommation RAM, consommation CPU, etc.**

***Abstract*—This paper describes the quantitative and qualitative differences between a NoSQL database (Cassandra) and a Blockchain database (Hyperledger Fabric) from the perspective of distributed database architectures. The assessment is performed through the YCSB and Hyperledger Caliper tools for three different types of read and write loads. The comparison is based on variables such as system processing rate, execution time, number of errors, latency, RAM consumption, CPU consumption, etc.**

I. INTRODUCTION

A. Motivation des bases de données distribuées

Au départ, les premiers systèmes informatiques avec des regroupements de données étaient constitués d'une base de données centralisée. Ces bases de données ayant leurs avantages et leurs inconvénients entre autres, les risques de pertes d'information en cas de panne ou de bris de l'équipement supportant la base de données. De plus, à travers le temps, les données évoluent en quantité d'information à sauvegarder et en complexité de leur organisation. Ce risque combiné aux besoins toujours grandissants d'espace ont favorisé l'apparition des bases de données réparties.

B. Caractéristiques des bases de données distribuées

Ces bases de données ont donc entre autres comme objectif de faciliter la gestion de très grande quantité de données, sous différentes structures organisationnelles. L'utilisation d'une base de données centralisée ou répartie n'est pas un changement apparent pour l'utilisateur, mais plutôt dans sa conception et son maintien. En effet, la base de données répartie est constituée de différents nœuds, ce qui permet d'éviter les pertes si une machine était en panne. Il est également plus facile de faire évoluer la base de données en lui ajoutant des nœuds ou en en retirant selon les besoins. Il est également plus facile de faire évoluer la base de données, en changeant un appareil plutôt que de changer le système en entier. En travaillant avec une base de données répartie, cela amène aussi des avantages de performances, puisque cela permet des accès concurrents. Il existe plusieurs compagnies et structures de base de données réparties.

C. Objectifs et contributions

Dans le cadre de ce rapport, nous effectuerons une étude comparative entre deux types de base de données. Nous nous intéresserons à la base de données NoSQL Cassandra et à la base de données Blockchain de Hyperledger Fabric. Nous tenterons de déterminer si les technologies sont équivalentes ou vouées à des cas d'utilisation différents selon les besoins des entreprises. Nous tenterons d'amener nos lecteurs à mieux comprendre les enjeux de latence et de temps de réponse en fonction du type d'architecture de la base de données.

II. CONTEXTE

A. Hyperledger Fabric (Base de données Blockchain)

Projet Open Source développé et distribué depuis 2015 par la fondation Linux, Hyperledger est une plateforme regroupant plusieurs cadriciels et outils offrant aux développeurs et entreprises la possibilité de développer, puis partager des contrats intelligents par le biais du blockchain. Ces cadriciels font généralement partie de l'une des catégories suivantes : *« ledgers »*, *librairies*, *outils*, *« domain-specific »* [1] [2].

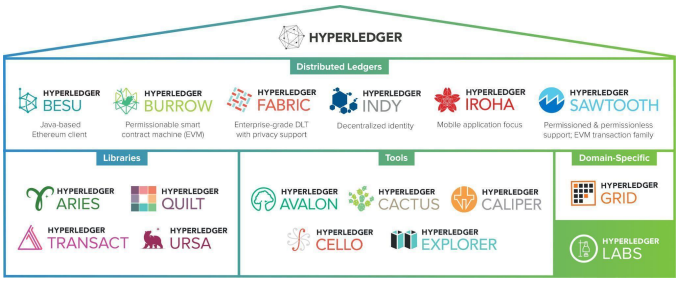

Fig 1. *Représentation des principaux outils Hyperledger [3]*

Dans le cadre de cette étude comparative, nous nous sommes servis de l'un de ces cadriciels, Hyperledger Fabric, principalement développé par IBM dans le cadre du projet parent à code source libre. Comparativement aux autres outils exploitant la technologie Blockchain, Hyperledger Fabric a été pensé spécifiquement pour les entreprises qui ont besoin de limiter ou gérer les permissions d'accès au réseau d'un groupe de participants. Toujours dans cette même optique, un accent particulier a été porté sur les notions d'évolutivité, de

confidentialité et de sécurité dans le développement de cette plateforme.

B. Cassandra (Base de données NoSQL)

Également à code source libre, Cassandra est un système de base de données NoSQL. Développée et distribuée par la fondation Apache, elle a été particulièrement conçue pour gérer de très larges quantités de données à travers de nombreux serveurs. [4]

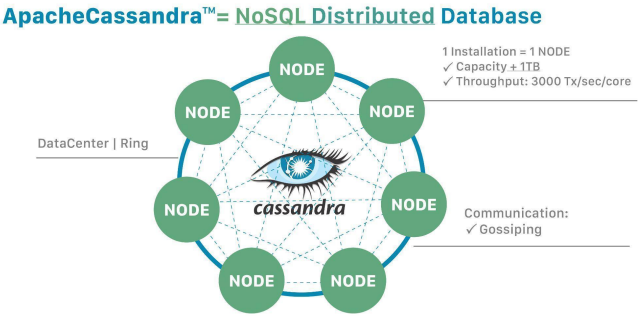

***Fig 2**. Réplication des données entre tous les nœuds/serveurs du Réseau de Cassandra [4]*

Utilisé par de nombreuses larges organisations telles que Netflix, Apple ou encore Twitter, Cassandra offre aux entreprises la garantie d'une très forte tolérance aux pannes tout en permettant facilement l'évolutivité du système, que ce soit pour gérer des pics occasionnels de demande ou pour permettre l'évolution permanente du système.

C. Comparaisons

Pour ce qui est du niveau de performance offert par la base de données NoSQL Cassandra et la base de données Blockchain Hyperledger Fabric, une revue de la littérature actuelle montre que Cassandra serait capable de traiter jusqu'à 3200 opérations par secondes dans un déploiement de type standard à 3 nœuds. Cela représente un niveau de performance significativement supérieur aux autres bases de données testées dans la même étude, variant entre 225 opérations par seconde pour MongoDB et 480 opérations par seconde pour Riak. L'étude montre aussi que, comparé à un déploiement Cassandra simple-nœud, le déploiement à 3 nœuds permet un niveau plus élevé de performance d'écriture, et donc que le travail supplémentaire de coordination entre les nœuds est inférieur à l'augmentation de performance offerte par le matériel supplémentaire. [5]

Une étude comparative de la performance de plusieurs bases de données avec l'outil d'analyse des performances YCSB. Dans celle-ci, les auteurs montrent que dans un scénario lourd en écriture composé de 50 % en lecture et 50 % en écriture, la base de données Cassandra est capable de maintenir une latence minimale jusqu'à 11000 opérations de lecture par seconde et 12000 opérations d'écriture par seconde. Et que dans un scénario lourd en lecture, soit 95 % en lecture et 5 % en écriture, la base de données Cassandra maintient un niveau de latence minimal jusqu'à 8000 opérations de lecture par secondes et 7000 opérations d'écriture par seconde. Ces résultats de test montrent donc assez clairement l'optimisation de Cassandra pour les opérations d'écriture. [6]

Comparativement, une autre étude sur la performance de la version 0.6 de Hyperledger Fabric montre qu'il y aurait une limite d'environ 300 transactions par secondes dans un scénario sans fautes. Mais l'étude note que le consensus entre les pairs pourrait devenir problématique avant d'atteindre cette limite et que d'approcher cette limite a tendance à faire exploser la latence d'écriture et pourrait même occasionner des fautes critiques où des transactions « acceptées » pourraient par la suite être silencieusement abandonnées. [7]

La présence de cette limite est aussi confirmée par une autre étude qui a trouvé que Hyperledger Fabric offre un temps de latence minimal jusqu'à 200 transactions par secondes, puis la latence augmente significativement jusqu'à 300 transactions par secondes. À ce point, le taux de traitement (Throughput) effectif du système est inférieur au taux de traitement effectif lorsque le système avait 200 transactions par seconde, mais avec un temps de latence environ 20 fois plus élevé. [8]

Ce niveau de performance pour Hyperledger Fabric est confirmé par une autre étude comparant la performance de la version 0.6 et 1.0. Dans celle-ci, la version 1.0 permettrait, pour des lots de 1000 transactions, de 185 à 460 transactions par seconde dépendamment du type de requête effectué. L'étude note aussi que le temps d'exécution augmente significativement avec le nombre de transactions autant dans la version 0.6 que dans la version 1.0, mais note tout de même que les temps de latence de la version 1.0 sont meilleurs que ceux de la version 0.6 lorsque le nombre de transactions augmente. [9]

Une étude sur l'optimisation de la version 1.0 de Hyperledger Fabric permettrait de significativement améliorer le niveau de performance offert par celle-ci, soit d'environ 140 transactions par secondes jusqu'à 2250 transactions par secondes. De plus, les auteurs indiquent que les optimisations qu'ils ont effectuées ont été communiquées aux développeurs de Hyperledger Fabric et que ceux-ci seront implémentés à partir de la version 1.1 de Hyperledger Fabric. [10]

Finalement, une dernière étude comparative entre des bases de données NoSQL et Blockchain montre que la base de données Blockchain Herperledger Fabric offre un temps de réponse moyen d'environ 1400 ms, alors que la base de données NoSQL MongoDB approchait plutôt les 300 ms. De plus, le meilleur temps de réponse de MongoDB est réussi avec une utilisation CPU d'environ 35 %, alors que Hyperledger Fabric atteint les 75 % d'utilisation CPU. Cela pourrait indiquer que les technologies NoSQL peuvent offrir un meilleur temps de réponse et une meilleure efficacité énergétique que les technologies de base de données Blockchain. [11]

En bref, la littérature actuelle semble indiquer que, du point de vue de la performance, les bases de données NoSQL auraient un avantage significatif comparé aux bases de données Blockchain, mais laisse sous-entendre qu'il serait possible que

les technologies de Blockchain deviennent suffisamment optimisées pour approcher le niveau de performance des bases de données NoSQL.

III. EXPÉRIMENTATIONS

A. Environnement et Installation

Pour mettre en place un environnement uniforme de test, nous avons envisagé plusieurs cadriciels potentiels. Nous avons tout d'abord tenté l'expérimentation avec les instances proposées t2.micro de Amazon Web Services (AWS). Cela dit, nous n'avons pas continué avec cette option étant donné la complexité que cela causait avec les particularités du déploiement distribué de la base de données Cassandra. Nous avons ensuite tenté l'expérimentation simplement avec l'environnement Docker. Bien que les instructions proposées par Hyperledger et Cassandra pouvaient bien s'intégrer dans des Dockerfile et des fichiers de configuration docker-compose, certaines erreurs se sont produites lors du lancement de Hyperledger. En effet, l'instance principale n'arrivait pas à enrôler les sous-instances.

Nous avons donc sélectionné la dernière option, c'est-à-dire celle d'une machine virtuelle. Avec le logiciel VirtualBox de Oracle et d'une image « Ubuntu Server 22.10 », nous avons créé une instance locale pour effectuer nos tests de bases de données distribuées. Nous avons sélectionné une machine hôte conventionnelle pour laquelle nous avons réservé 8192 Mb de mémoire vive et 1 cœur de processeur (processeur virtuel). Nous discutons de la nature de ce choix et de ses impacts dans la section « Limitations et obstacles ».

Pour produire les différentes configurations et scripts de développement, nous avons regroupé les opérations utiles dans un dépôt distant (« repository ») [14]

1. *Déploiement de Hyperledger Fabric*

Sur la machine virtuelle décrite précédemment, nous avons procédé à l'installation et au déploiement de Hyperledger Fabric par le biais de commandes traitées directement en ligne de commande. Spécifiquement, ici, nous nous sommes servis de « Caliper Benchmarks », un outil développé par Hyperledger permettant de tester et d'étudier les performances de solutions blockchain diverses.

Après avoir installé les principales dépendances nécessaires pour la suite (nodejs 12 et npm, build-essential, docker, git et Python 2.7.18), nous avons procédé, dans l'ordre à l'exécution des étapes suivantes :

Étape	Description
1	Clonage du projet Hyperledger Fabric Caliper Benchmarks depuis Github
2	Checkout vers le commit d02cc8bb du 8 décembre 2020
	Le commit suivant dans le répertoire networks/fabric/config_solo_raft/ supprime les fichiers pour générer un réseau prédéfini de nœuds pour Hyperledger Fabric. Ainsi, nous avons accès au fichier `generate.sh` et aux fichiers de configuration d'un réseau fabric par défaut.
3	Installation du CLI de Caliper Benchmarks
4	Exécution d'un script de génération de la configuration de réseau de base, présent dans le dossier /networks/fabric/config_solo_raft/
5	Exécution du script de lancement du manager de Hyperledger Fabric avec Caliper (npx caliper launch manager) avec les configurations relatives au benchmark et à la configuration réseau, respectivement trouvés dans les fichiers : benchmarks/samples/fabric/marbles/config.yaml --caliper-networkconfig et networks/fabric/v1/v1.4.4/2org1peercouchdb_raft /fabric-go-tls-solo.yaml.

Tableau 1. *Étapes de déploiement de Hyperledger Fabric*

Il est à noter que de nombreuses erreurs ont été rencontrées dans le cadre du déploiement de Hyperledger Fabric avec Caliper, généralement lié à des problèmes de version des différents outils et cadriciels utilisés (commit de hyperledger, version de nodejs/npm et Python). La configuration présentée ici est donc le résultat de ces diverses expériences afin de faire réaliser le benchmark du cadriciel sur notre système.

2. *Déploiement de Cassandra*

Pour déployer une base de données distribuée avec Cassandra, on met en place quatre conteneurs basés sur l'image bitnami/cassandra (4.0). On assigne à chacune une adresse IP du sous-réseau 192.168.5.0/24. On redirige chacun des ports internes (9042 pour l'accès client et 7000 pour la communication inter instances) vers des ports uniques du sous-réseau.

Cette configuration à travers docker-compose nous permet de lancer les quatre instances simultanément et de coordonner les communications. On crée un « keyspace » [12] pour établir les caractéristiques de l'environnement de benchmark :

```
create keyspace ycsb WITH REPLICATION = {'class':
'SimpleStrategy', 'replication_factor':3}
```

Ainsi, la base de données distribuée Cassandra devrait contenir après toute transaction trois copies des données. [13]

On crée ensuite les colonnes de la table qui serviront aux tests. Dans ce cas précis, chaque ligne de la table "usertable" contiendra 10 colonnes et une clé primaire :

```
docker exec -it cassandra1 cqlsh 192.168.5.2 -u
cassandra -p cassandra -e "create table
ycsb.usertable (
        y_id varchar primary key,
        field0 varchar,
        field1 varchar,
        …
        field9 varchar);"
```

3. Tâches et lots de travail sélectionnés

Pour assurer une stratégie d'évaluation uniforme entre les deux bases de données distribuées, il a fallu sélectionner des tâches et des lots de travail similaires. Nous voulions tester plusieurs types de charges et ainsi pouvoir comparer les bases de données selon des charges variées de lecture et d'écriture. Nous avons sélectionné trois types de charges distincts.

Le premier consiste en une charge de 100 % en lecture et de 0 % en écriture. Celle-ci nous permettra de simuler une base de données largement remplie et désormais très peu active en écriture. Les transactions effectuées sont majoritairement, sinon totalement, à des fins de consultation par les utilisateurs.

Le second consiste en une charge égale entre la lecture et l'écriture (50 % chaque). Celle-ci nous permettra de simuler une base de données relativement active en écriture et relativement active en lecture aussi. Les transactions effectuées sont tout autant à des fins de modification et d'ajout qu'à des fins de consultations par les utilisateurs.

Le troisième consiste en une charge de 10 % en lecture et de 90 % en écriture. Celle-ci nous permettra de simuler une base de données très peu remplie et actuellement très active en écriture. Les transactions effectuées sont majoritairement, sinon presque totalement, à des fins de modifications et d'ajout par les utilisateurs.

Le tableau suivant recense les trois types de charges mentionnés précédemment :

Type de charge	**C**	**A**	**G**
Ratio Lecture	100 %	50 %	10 %
Ratio Écriture	0 %	50 %	90 %

Tableau 2. *Types de charges utilisées*

4. Mise en place de Hyperledger Caliper

Hyperledger Caliper permet de définir des fichiers de configurations pour varier les types de charges fournis au système lors des tests. Pour respecter les configurations C, A et G, nous avons produit les fichiers de configuration suivants :

Type de charge	**C**	**A**	**G**
Fichier de configuration	configc.yaml	configa.yaml	configg.yaml
Nombre d'opérations théorique[1]	1000	1000	1000
Écriture (init.js) – tx Number	0	500	900
Écriture (init.js) – tps	5	5	25
Lecture (query.js) – tx Duration	200	20	20
Lecture (query.js) – tps	5	25	5

Tableau 3. *Données de configuration pour Hyperledger Fabric*

Il ne reste ensuite qu'à Hyperledger Caliper à travers l'exécution de la commande mentionnée à l'étape #5 du déploiement d'Hyperledger Fabric. La sortie de la commande nous fournit directement les résultats recherchés.

5. Mise en place de Yahoo! Cloud Service Benchmark (YCSB)

YSCB nous permet aussi de définir des configurations de type de charge sous la forme de « workload » :

Type de charge	**C**	**A**	**G**
Fichier de configuration	workloadc	workloada	workloadg
Nombre d'opérations théorique[1]	1000	1000	1000
readproportion	1	0.5	0.1
updateproportion	0	0.5	0.9
scanproportion	0	0	0
insertproportion	0	0	0

Tableau 4*. Données de configuration pour YSCB*

*****Il est à noter que le nombre d'opérations théorique est celui spécifié dans la configuration, mais que pour des raisons expérimentales, celui-ci peut varier légèrement lors de l'exécution.

Les fichiers de configuration C et A étaient déjà existants dans le dépôt distant de l'outil de benchmark YSCB. Le fichier de configuration G a été produit à partir du modèle proposé par YSCB.

[1] Le nombre d'opérations théorique est celui spécifié dans la configuration, mais que pour des raisons expérimentales, celui-ci peut varier légèrement

Pour chacune des configurations de charge, nous avons procédé à trois itérations. Chaque itération était composée d'une phase de « Load » correspondant à un remplissage de la base de données, et à une phase « Run » correspondant aux opérations CRUD avec les ratios spécifiés dans la configuration [15].

Nous avons ensuite pris les moyennes des trois valeurs récupérées de la phase « Load » pour avoir une représentation valide des tests effectués.

A. Résultats

On présente ici les résultats des exécutions des bases de données Cassandra et Hyperledger Fabric. Toutes les mesures temporelles sont en secondes.

1. Observations et performances

D'une part, Hyperledger Caliper nous fournit des données sur les latences, les temps d'exécution et les taux de traitement de la base de données Hyperledger Fabric sous différents types de charge (selon le ratio d'accès écriture/lecture) :

Type de charge		**C**	**A**	**G**
Écriture	Latence maximale	0.91	1.38	1.46
	Latence minimale	0.86	0.52	0.43
	Latence moyenne	0.88	0.74	0.66
	Temps d'exécution	1.08	99.81	179.70
Lecture	Latence maximale	30.27	2.22	18.07
	Latence minimale	0.70	0.59	2.28
	Latence moyenne	15.39	1.08	13.67
	Temps d'exécution	34.16	101.35	32.71

Tableau 5. *Latences et temps d'exécution pour Hyperledger Fabric obtenus avec Hyperledger Caliper*

Type de charge		**C**	**A**	**G**
Écriture	Opérations	5	500	900
	Taux de traitement	5.4	5.0	5.0
	Erreurs	0	0	0
Lecture	Opérations	1005	505	105
	Taux de traitement	29.5	5.0	3.2
	Erreurs	0	0	0

Tableau 6. *Taux de traitement, opérations et erreurs pour Hyperledger Fabric obtenues avec Hyperledger Caliper*

D'autre part, YSCB nous fournit aussi des données sur les latences, les temps d'exécution et les taux de traitement de la base de données Cassandra sous différents types de charge (selon le ratio d'accès écriture/lecture) :

Type de charge		**C**	**A**	**G**
Écriture	Latence maximale	N/A	4.169343	4.109801
	Latence minimale		0.001600	0.001578
	Latence moyenne		0.021925	0.015050
Lecture	Latence maximale	8.173737	4.109183	0.067369
	Latence minimale	0.001764	0.001915	0.002950
	Latence moyenne	0.020216	0.022145	0.011873

Tableau 7. *Latences pour Cassandra obtenues avec YSCB*

Type de charge		**C**	**A**	**G**
Écriture	Opérations	N/A	510	911
	Erreurs		0	0
Lecture	Opérations	1000	490	89
	Erreurs	0	0	0
Temps d'exécution confondu		25.526	27.677	20.444
Taux de traitement confondu		42.69978	37.79710	53.61020

Tableau 8*. Nombres d'opérations, taux de traitement, erreurs et temps d'exécution pour Cassandra obtenus avec YSCB*

Une visualisation logarithmique des différentes latences des deux bases de données est disponible à la section « Annexes »

Avec les outils de profilage des ressources utilisées par les conteneurs internes à la machine virtuelle, nous avons aussi pu recenser l'évolution de la consommation des ressources au fil des exécutions. Afin de ne pas alourdir le document, ces graphiques se trouvent à la section « Annexes ».

2. Constats et analyse

Les latences des instances Cassandra sont relativement élevées, dans l'ordre de la seconde en moyenne. Dans le cas C, on remarque une latence moyenne de 15.39 lors des accès en lecture. Cette latence est non négligeable, mais est mitigée par le fait que l'on peut avoir des latences très courtes dans le meilleur cas (moins de 0.70 seconde) ou très longues dans le pire cas (plus de 34.16 secondes).

On remarque aussi que, dans le cas de type de charge G, Hyperledger Fabric a pris environ 180 secondes pour effectuer les 900 écritures. Le taux d'entrée de 25 transactions par seconde est diminué à 5 transactions traitées par seconde. Il y a donc probablement un goulot d'étranglement au niveau du taux de traitement de transactions dans ce cas précis. On remarque aussi le même phénomène en lecture (5 transactions en entrée pour 3.2 transactions traitées).

Les latences des instances de Cassandra sont relativement basses, dans l'ordre du centième de seconde en moyenne. Le pire cas se trouve aussi dans la latence maximale du cas C lors d'un accès en lecture de 8.17 secondes. On remarque cependant que les meilleurs cas peuvent atteindre l'ordre du millième de seconde.

Le taux de traitement est relativement élevé et constant dans les trois types de charges. Cassandra permet de compléter entre 37 et 53 transactions par seconde, et ce avec des temps d'exécution relativement bas. En effet, les trois types de charges ont pris moins de 30 secondes à s'exécuter correctement.

En ce qui concerne l'utilisation des ressources CPU, on remarque que les instances de Cassandra sont relativement inactives durant le début de la plupart des opérations de lectures et d'écritures. On remarque cependant un niveau d'activité élevé autour de 40 % de chacune des instances vers la fin de l'opération. Cela démontre une répartition correcte de la charge entre les instances. Les instances de Hyperledger Fabric sont, pour leur part, actives à raison de sursauts périodiques à 50 % d'activité. La fin de l'opération suscite ici aussi des variations d'intensité notables.

Finalement, relativement à la consommation de mémoire vive (RAM) pour les instances de Cassandra, on remarque que le niveau des instances ne varie que très peu du début à la fin (entre 15.75 % et 16.25 %). Cela est dû au fait que chacune des instances de Cassandra alloue directement la quantité de mémoire auquel elle a droit, même si cet espace n'est pas nécessairement utilisé. De leur côté, les instances de Hyperledger Fabric ne consomment que très peu de mémoire vive. On remarque une augmentation périodique, mais celle-ci ne cumule jamais au-delà de 3 % de la mémoire vive disponible. Il faut toutefois prendre ce résultat avec un certain recul. Il est possible que les instances de Hyperledger Fabric génèrent elles-mêmes des sous-conteneurs pour effectuer le traitement des tâches de lecture et d'écriture.

IV. DISCUSSIONS

A. Comparaisons et différences

Cassandra et Hyperledger Fabric sont tous les deux des bases de données conçues pour être utilisées par de larges entreprises ou institutions ayant besoin de stocker de très larges quantités de données et maintenir une qualité de service optimale malgré un nombre élevé de transactions. Il apparaît intéressant de comparer les deux technologies étudiées dans le cadre de ce rapport.

Si les deux technologies n'ont pas été développées dans le même objectif, elles possèdent néanmoins deux cas d'utilisation communs : pouvoir insérer et récupérer de l'information stockée dans un système apparaissant comme centralisé pour l'utilisateur ou les autres systèmes connexes qui souhaitent l'utiliser.

En ce qui concerne le taux de latence et le temps d'exécution moyen des requêtes, Cassandra semble, une fois de plus, se démarquer. Les quatre instances offrent une latence moyenne de l'ordre du dixième de secondes (0.1s) tandis que pour les transactions réalisées avec Hyperledger Fabric, le temps de latence est généralement de l'ordre de la seconde, voire de la dizaine de secondes dans les pires cas. La différence est particulièrement flagrante pour les charges de travail qui contiennent un nombre élevé d'écritures, où on observe 0.011873 seconde en moyenne pour Cassandra contre 13.67 secondes pour Hyperledger Fabric. La figure suivante permet de visualiser la disparité entre les deux technologies du point de vue du temps d'exécution (en secondes).

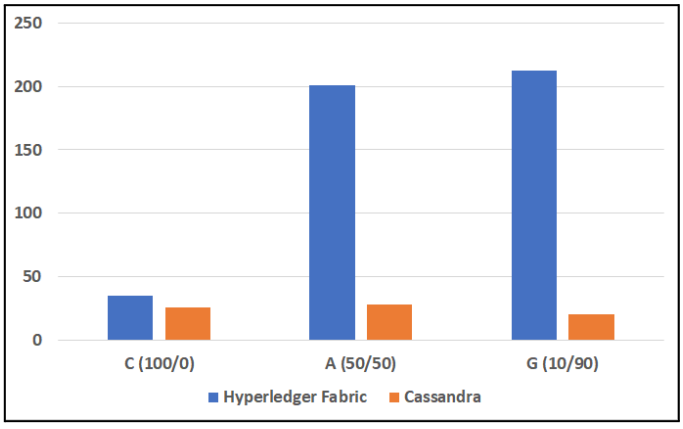

***Fig 3**. Comparaison des temps d'exécution de Cassandra et d'Hyperledger Fabric sous trois types de charges*

La tendance est la même lorsque nous comparons le taux de traitement des transactions pour Cassandra et Hyperledger Fabric. D'après les résultats obtenus à la section précédente pour la charge de travail G, le taux de traitement confondu moyen de Cassandra atteint 53.61 transactions par secondes

tandis qu'on reste à 8.2 en moyenne pour Hyperledger Fabric, ce qui rend Cassandra presque 7 fois plus efficace. La figure suivante montre un comparatif du taux de traitements (nombre de transactions par seconde) pour les deux technologies, selon les 3 charges de travail étudiées.

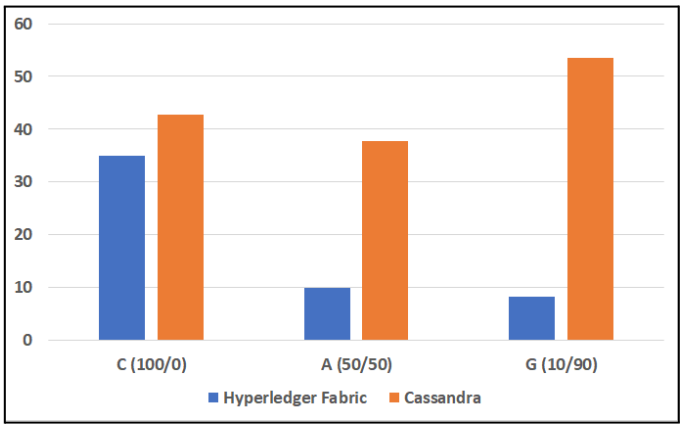

***Fig 4**. Comparaison des taux de traitement de Cassandra et d'Hyperledger Fabric sous trois types de charges*

Dans les deux cas, aucune erreur n'a été constatée dans le cadre de nos tests, quel que soit le type de charge. Les deux technologies peuvent donc être considérées comme équivalentes à ce niveau, dès lors que l'on prend en compte l'échantillon limité utilisé.

Ainsi, à travers cette étude, nous pouvons déterminer que Cassandra est en moyenne beaucoup plus performante que Hyperledger Fabric, particulièrement lorsque l'on réalise un nombre élevé d'écritures. Cela peut s'expliquer par le fait que Cassandra a été développée spécifiquement pour supporter de très larges échanges de données à travers plusieurs nœuds (mettant donc de l'avant la performance) tandis que Hyperledger Fabric a été développé pour privilégier la sécurité et surtout la cohérence des données par le biais du système blockchain avec un mécanisme basé sur le consensus.

Chacune de ces deux technologies peut donc s'avérer pertinente selon le cas d'utilisation de chaque entreprise. Il est ainsi important de souligner que cette différence ne sera réellement perceptible que dans les cas où la minimisation de la latence et l'optimisation du taux de traitement sont critiques au fonctionnement d'un système.

B. Limitations et obstacles

Comme mentionné précédemment, il aurait été intéressant de pouvoir faire fonctionner les tests des deux bases de données avec un environnement constitué purement de fichiers de configuration Docker. Malheureusement, ce ne fut pas le cas. Bien que la solution admise pour ce travail ait été celle de la machine virtuelle fonctionnant avec un environnement sous-jacent Docker, il serait intéressant de se pencher sur les raisons pour lesquelles les membres de l'organisation (« workers ») n'arrivaient pas à communiquer entre eux lors du lancement d'une tâche de test. La documentation à ce sujet est très éparse et la liste de causes potentielles est très garnie. Il faudrait donc s'attarder au cœur du problème et comprendre pourquoi, dans la situation actuelle, le déploiement d'Hyperldeger avec des fichiers de configuration Docker n'a pas permis aux instances de communiquer entre elles et a produit les erreurs d'enrôlement.

Une autre limitation fut les limites transactionnelles de la base de données Hyperledger. Puisque le comportement des instances d'Hyperldger est très dépendant des contraintes de la machine virtuelle, certaines combinaisons de charges de lecture et d'écriture génèrent des erreurs homologues à celles-ci.

```
GET_QUERY_RESULT failed: transaction
ID:0eeb5d2925b20a1548f423805eaf4d6e03291458c97c82
d1617eba562d32fe4c: error reading response body:
net/http: request canceled (Client.Timeout
exceeded while reading body)
```

Ainsi, il y a parfois des cas de figure qui ne permettent pas aux instances d'Hyperledger d'accepter le taux d'arrivée des requêtes de lecture ou d'écriture. Les requêtes sont ainsi abandonnées et conduisent à des échecs. Ce constat est d'autant plus marqué par le nombre de cœurs de processeurs attribués à la machine virtuelle.

En effet, notre expérimentation se base sur une machine virtuelle à un seul cœur. Avec la même machine physique, nous avons voulu augmenter d'un seul cœur à quatre. Par contre, comme mentionné précédemment, ce changement n'a pas été nécessairement bénéfique pour les charges sur les instances d'Hyperledger.

On note en vert les variations bénéfiques de l'utilisation de quatre cœurs au lieu d'un seul et en rouge les variations défavorables.

Type de charge		**C**	**A**	**G**
Écriture	Latence maximale	0.92	1.55	2.43
	Latence minimale	0.85	0.56	0.59
	Latence moyenne	0.90	0.69	0.94
	Temps d'exécution	1.038	99.828	179.757
Lecture	Latence maximale	8.14	2.35	34.67
	Latence minimale	0.49	0.64	14.37
	Latence	2.57	1.06	26.42

	moyenne			
	Temps d'exécution	22.217	101.049	54.774

Tableau 9. *Latences et temps d'exécution pour Hyperledger Fabric obtenus avec Hyperledger Caliper pour l'exécution avec 4 cœurs virtuels*

Type de charge		**C**	**A**	**G**
Écriture	Opérations	5	500	900
	Taux de traitement	5.4	5.0	5.0
	Erreurs	0	0	0
Lecture	Opérations	1005	505	105
	Taux de traitement	45.3	5.0	1.9
	Erreurs	0	0	90

Tableau 10. *Taux de traitement, opérations et erreurs pour Hyperledger Fabric obtenues avec Hyperledger Caliper pour l'exécution avec 4 cœurs virtuels*

On remarque ainsi que quatre cœurs virtuels améliorent largement les performances avec le type de charge C les performances des instances Hyperledger. En effet, le taux de traitement (Throughput) est largement supérieur (45.3) à celui à un seul cœur (29.5). Il en va de même pour le temps d'exécution largement inférieur (22.217) à celui à un seul cœur (34.167).

Par contre, on remarque que, dans le cas des types de charges A et G, l'expérimentation avec quatre cœurs a significativement réduit les performances. La plupart des latences sont plus élevées et le système a même généré 90 erreurs de lecture dans le type de charge G.

On peut donc présupposer que plusieurs cœurs seraient bénéfiques à un type de charge fort en lecture (C), mais défavorables à ceux forts en lecture (A et G).

Il serait ainsi intéressant de répéter l'expérimentation de la variation du nombre de cœurs sur différentes machines physiques pour confirmer ou infirmer que la puissance des processeurs physiques influence la performance des tests de charge dans la machine virtuelle hébergée.

V. CONCLUSION

En résumé, nous avons trouvé dans la littérature que Cassandra devrait pouvoir supporter jusqu'à 3200 opérations par seconde sur un déploiement unique [5] et jusqu'à 12000 opérations par seconde dans un déploiement à 3 nœuds [6]. De son côté Hyperledger Fabric pourrait supporter jusqu'à 300 transactions par seconde [7]. On constate donc un rendement plus élevé chez la base de données NoSQL Cassandra, mais cela est dû à la prise en compte des enjeux de sécurité de la base de données Blockchain Hyperledger Fabric.

Nous observons également ce genre de performance avec notre expérimentation. Les temps d'exécution de Cassandra sont largement inférieurs à ceux de Hyperledger Fabric pour des charges nécessitant une quantité même quasi négligeable d'écritures. Le graphique suivant démontre bien que le taux de traitement confondu est largement supérieur chez les instances de Cassandra (bleu) :

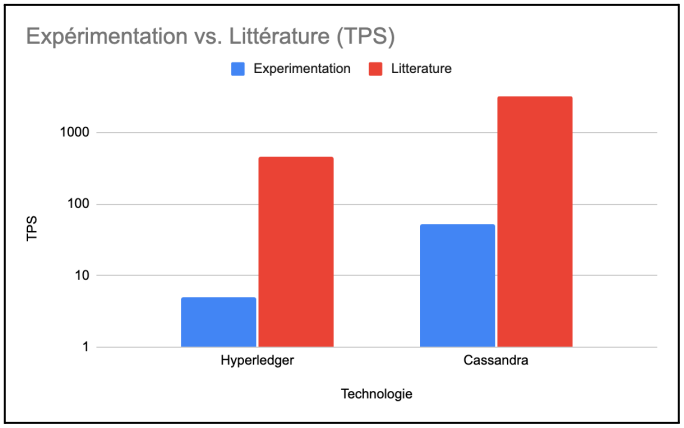

Fig 5*. Comparaison des taux de traitement obtenus expérimentalement à ceux de la littérature*

Cependant, pour une même base de données, nous remarquons un écart considérable entre le nombre de transactions que la base de données devrait effectuer dans le cadre de notre étude, par rapport au nombre de transactions que nous avons obtenues. Nous supposons que cet écart entre nos résultats et ceux de la littérature sont dus en partie à notre utilisation d'une machine virtuelle pour contenir les déploiements de nos bases de données.

Dans une étude future, il serait intéressant d'effectuer le même déploiement, directement sur une machine physique, sans l'utilisation de la machine virtuelle, afin de déterminer si cette dernière cause un goulot d'étranglement.

VI. REFERENCES

[1] Elli Androulaki et al., Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains. (2018). [En ligne]. Disponible : https://arxiv.org/abs/1801.10228v1

[2] IBM Research Editorial Staff, Behind the Architecture of Hyperledger Fabric, (2018). [En ligne]. Disponible : https://www.ibm.com/blogs/research/2018/02/architecture-hyperledger-fabric/

[3] Hyperledger Foundation, HL_Greenhouse_Current. (2022). [En ligne]. Disponible : https://www.hyperledger.org/use/attachment/hl_greenhouse_current

[4] Apache Foundation, What is Apache Cassandra?. (2023). [En ligne]. Disponible : https://cassandra.apache.org/_/cassandra-basics.html

[5] John Klein et al., Performance Evaluation of NoSQL Databases: A Case Study. (2015). [En ligne]. Disponible : https://dl.acm.org/doi/abs/10.1145/2694730.2694731

[6] Brian F. Cooper et al., Benchmarking cloud serving systems with YCSB. (2010). [En ligne]. Disponible: https://dl.acm.org/doi/epdf/10.1145/1807128.1807152

[7] Imre Kocsis et al., Towards Performance Modeling of Hyperledger Fabric. [En ligne]. Disponible: http://webspn.hit.bme.hu/~telek/cikkek/kocs17a.pdf

[8] Murat Kuzlu et al., Performance Analysis of a Hyperledger Fabric Blockchain Framework: Throughput, Latency and Scalability (2019). [En ligne]. Disponible: https://ieeexplore.ieee.org/abstract/document/8946222

[9] Qassim Nasir et al., Performance Analysis of Hyperledger Fabric Platforms (2018). [En ligne]. Disponible : https://www.hindawi.com/journals/scn/2018/3976093/

[10] Parth Thakkar et al., Performance Benchmarking and Optimizing Hyperledger Fabric Blockchain Platform (2018). [En ligne]. Disponible : https://ieeexplore.ieee.org/abstract/document/8526892

[11] Mohammadreza Rasolroveicy et al., Performance Evaluation of Distributed Ledger Technologies for IoT data registry : A Comparative Study. (2020). [En ligne]. Disponible: https://ieeexplore.ieee.org/abstract/document/9210358

[12] Intellipaat. Cassandra keyspace. (2023). [En ligne]. Disponible : https://intellipaat.com/blog/cassandra-keyspace/

[13] DataStax. CREATE KEYSPACE. (2022). [En ligne]. Disponible : : https://docs.datastax.com/en/cql-oss/3.3/cql/cql_reference/cqlCreateKeyspace.html

[14] Xavier Dupuis, LOG8430 Benchmark NoSQL and Blockchain. (2023). [En ligne]. Disponible : https://github.com/XavierDupuis/LOG8430-Benchmark-NoSQL-and-Blockchain

[15] Benchant, The Ultimate YCSB Benchmark Guide (2021). (2021). [En ligne]. Disponible : https://benchant.com/blog/ycsb

VII. ANNEXES

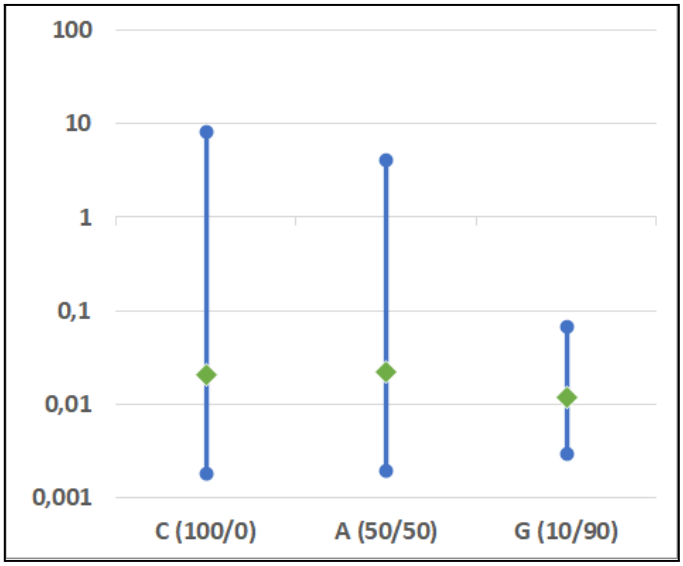

***Fig 6**. Latences minimales, moyennes et maximales des lectures des instances de Cassandra selon la charge exécutée*

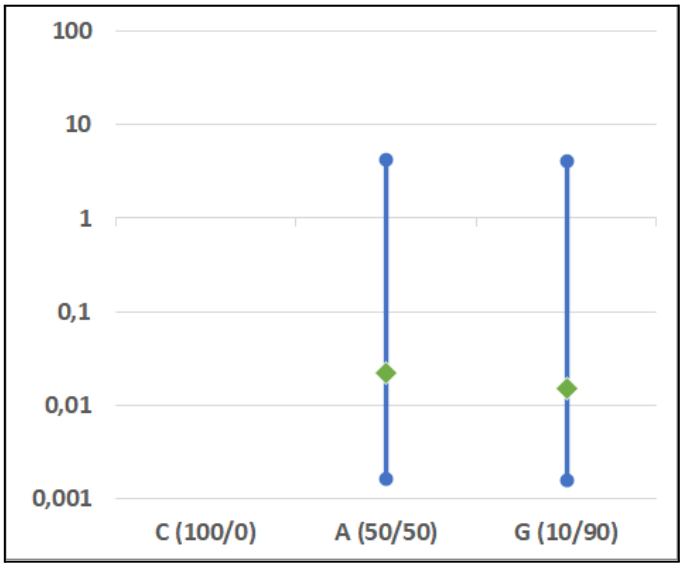

***Fig 7**. Latences minimales, moyennes et maximales des écritures des instances de Cassandra selon la charge exécutée*

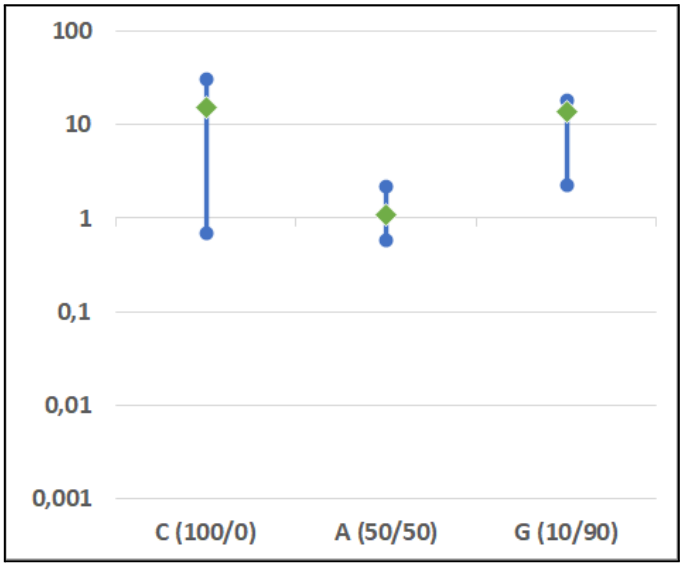

***Fig 8.** Latences minimales, moyennes et maximales des lectures des instances de Hyperledger Fabric selon la charge exécutée*

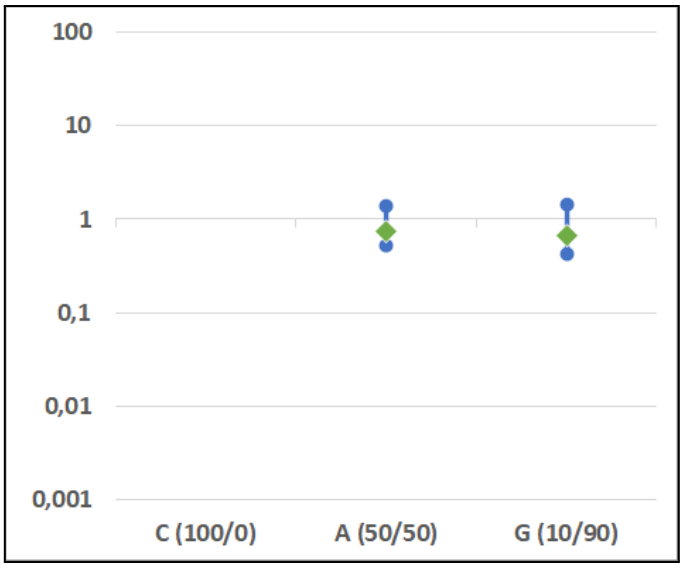

***Fig 9**. Latences minimales, moyennes et maximales des écritures des instances de Hyperledger Fabric selon la charge exécutée*

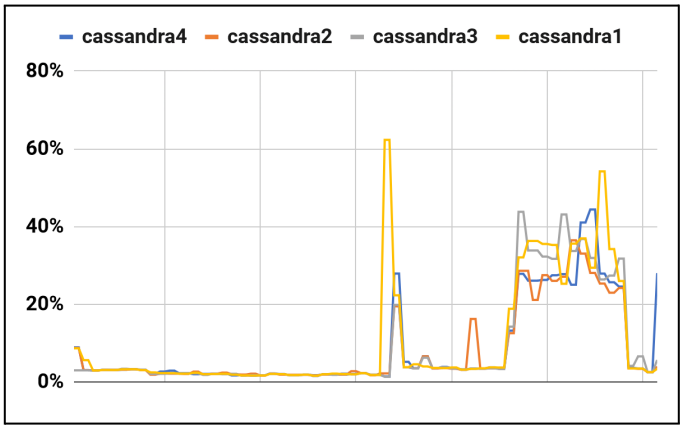

***Fig 10**. Consommation CPU des instances de Cassandra lors l'exécution de la charge C*

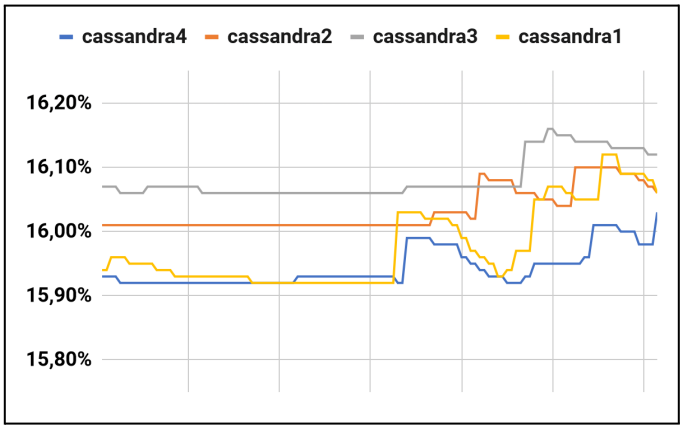

***Fig 13**. Consommation RAM des instances de Cassandra lors l'exécution de la charge C*

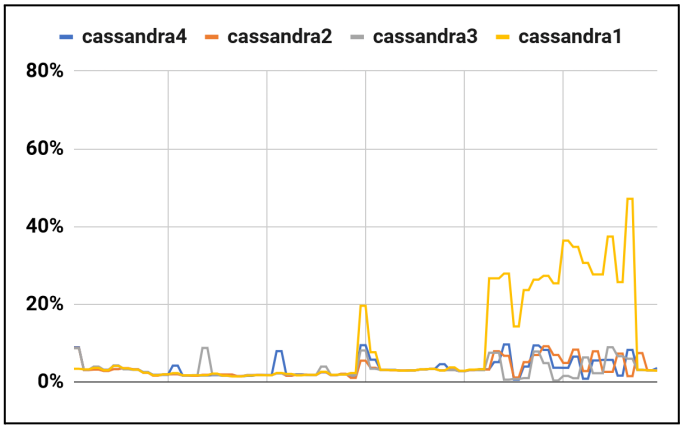

***Fig 11**. Consommation CPU des instances de Cassandra lors l'exécution de la charge A*

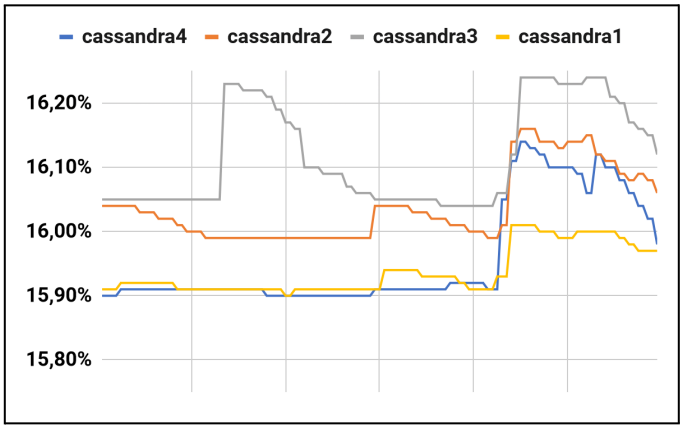

***Fig 14**. Consommation RAM des instances de Cassandra lors l'exécution de la charge A*

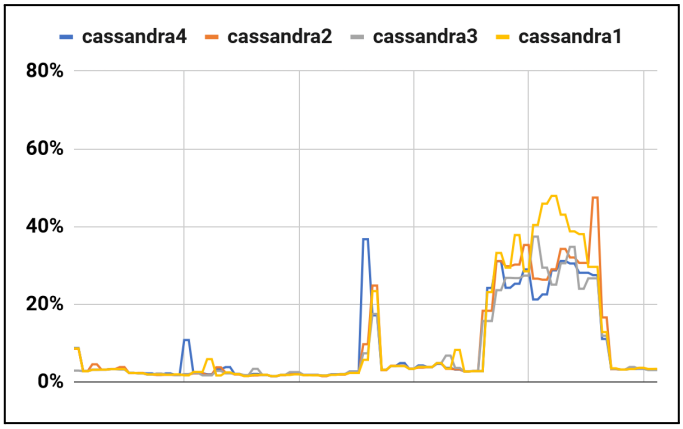

***Fig 12**. Consommation CPU des instances de Cassandra lors l'exécution de la charge G*

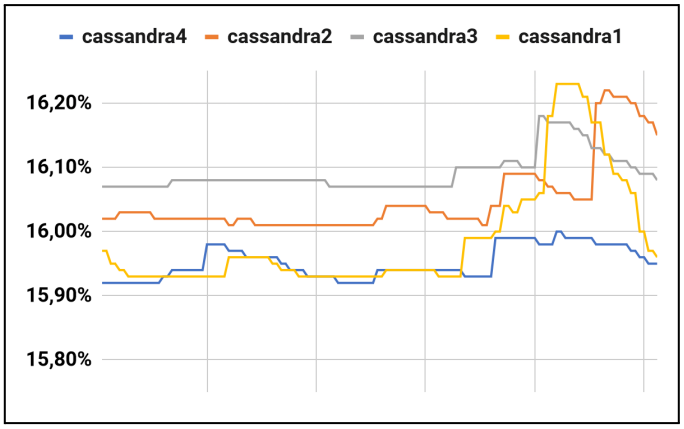

***Fig 15**. Consommation RAM des instances de Cassandra lors l'exécution de la charge G*

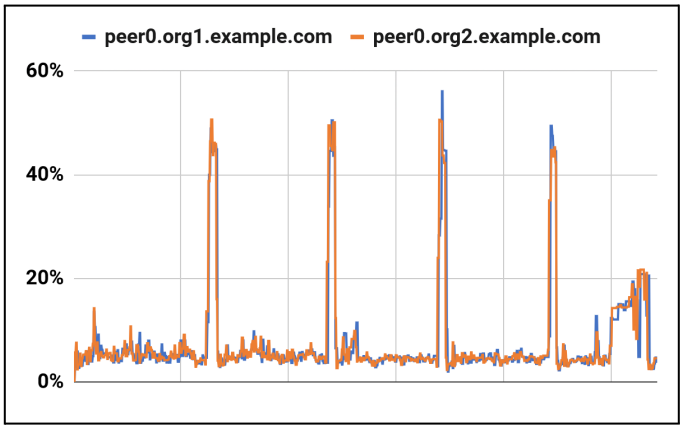

***Fig 16**. Consommation CPU des instances de Hyperledger Fabric lors l'exécution de la charge C*

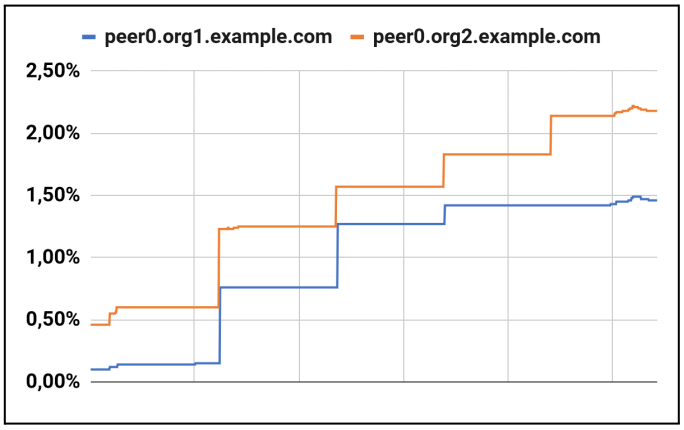

***Fig 19**. Consommation RAM des instances de Hyperledger Fabric lors l'exécution de la charge C*

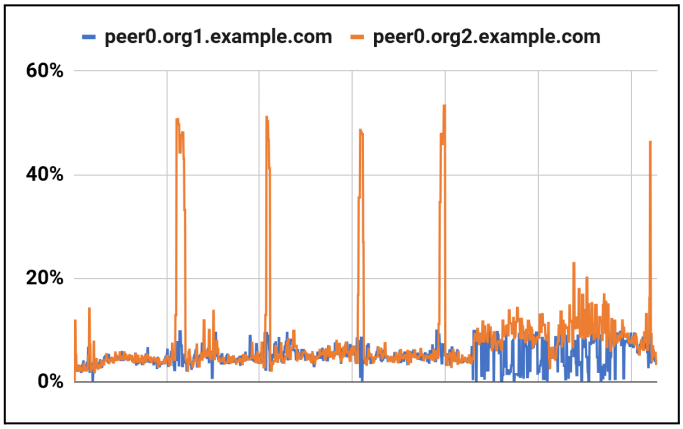

***Fig 17**. Consommation CPU des instances de Hyperledger Fabric lors l'exécution de la charge A*

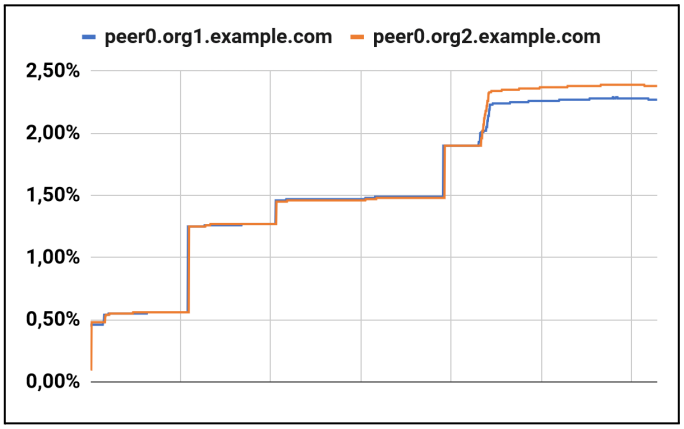

***Fig 20**. Consommation RAM des instances de Hyperledger Fabric lors l'exécution de la charge A*

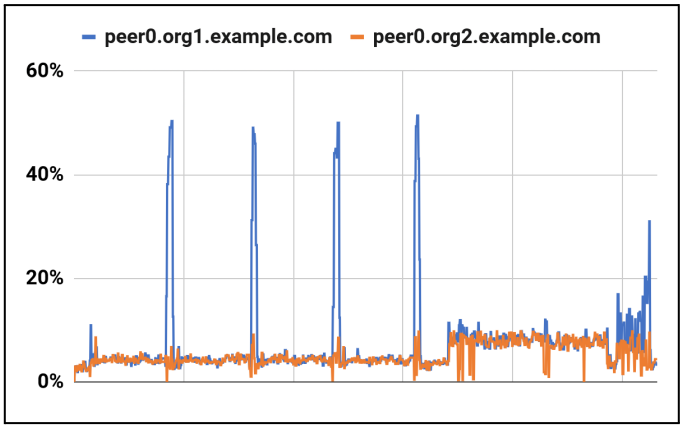

***Fig 18**. Consommation CPU des instances de Hyperledger Fabric lors l'exécution de la charge G*

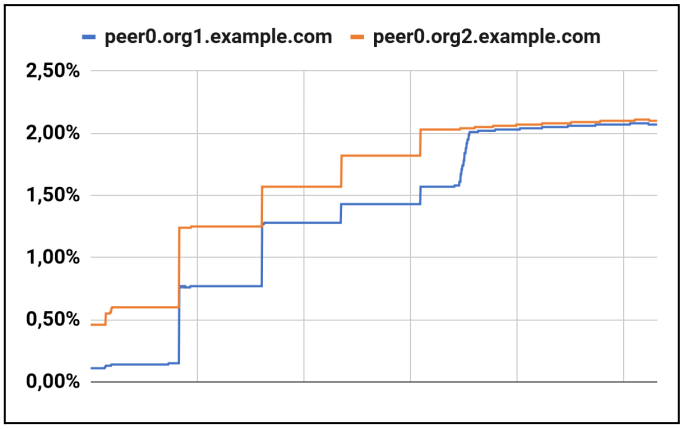

***Fig 21.** Consommation RAM des instances de Hyperledger Fabric lors l'exécution de la charge G*