

Rapport de Comparaison des Performances entre MySQL et MongoDB

Brelot Julien



Partenaire stratégique
Institut Mines-Télécom

Table des matières

1	Introduction	5
2	Contexte et Objectifs	6
I	Systèmes Comparés	
I.1	MongoDB	
I.2	MySQL	
II	Étude des aspects de MongoDB	
II.1	Comparaison des Performances des Requêtes	
II.2	Performances avec et sans Indexation	
II.3	RéPLICATION et Fragmentation	
3	Structure des Tests	11
I	Architecture MongoDB	
I.1	Architecture avec serveur unique : Standalone	
I.2	Architecture avec réplications : Replica Set	
I.3	Architecture Fragmentée : shards	
II	Architecture MySQL	
II.1	Architecture avec serveur unique : Standalone	
II.2	Architecture Distribuée : Cluster	
III	Description des Tests	
III.1	Contexte	
III.2	Principe des tests	
III.3	Paramètres des tests	
III.4	Structure des Données	
4	Résultats et Analyses pour MongoDB	17
I	MongoDB Standalone	
I.1	Performances des Opérations uniques	
I.2	Performances des Opérations avec données multiples . .	
I.3	Indexation	
II	RéPLICATION	
II.1	Performances des Opérations uniques	
II.2	Performances des Opérations avec données multiples . .	
II.3	Indexation	

- II.4 Observations Globales sur la RéPLICATION
- III Sharding
 - III.1 Performances des Opérations uniques
 - III.2 Performances des Opérations avec données multiples
 - III.3 Indexation
 - III.4 Observations Globales sur la fragmentation

5 Résultats et Analyses pour MySQL

- I MySQL Standalone
 - I.1 Performances des Opérations uniques
 - I.2 Performances des Opérations avec données multiples
 - I.3 Indexation
 - I.4 Observations Globales sur MySQL Standalone
- II MySQL Cluster
 - II.1 Performances des Opérations uniques
 - II.2 Performances des Opérations avec données multiples
 - II.3 Indexation
 - II.4 Observations Globales sur MySQL Cluster

6 Analyse des Résultats

- I Performances des Requêtes
- II Impact de la RéPLICATION et du Sharding
- III Critique des résultats
- IV Comparaisons et Bilan

7 Conclusion

A Annexe

- I Déploiement du Code
 - I.1 Tests de performance de MongoDB
 - I.2 Tests de performance de MySQL

Table des figures

3.1	Architecture Standalone.
3.2	Architecture Replica Set.
3.3	Architecture Sharded Cluster de MongoDB.
3.4	Architecture MySQL Cluster.
4.1	Distribution des performances des opérations CRUD uniques.
4.2	Performances des opérations CRUD uniques selon la quantité de données dans la base.
4.3	Distribution des performances des opérations CRUD multiples.
4.4	Performances des opérations CRUD multiples selon la quantité de données dans la base.
4.5	Distribution des performances des opérations CRUD uniques.
4.6	Performances des opérations CRUD uniques selon la quantité de données dans la base.
4.7	Distribution des performances des opérations CRUD multiples.
4.8	Performances des opérations CRUD multiples selon la quantité de données dans la base.
4.9	Distribution des performances des opérations CRUD uniques.
4.10	Performances des opérations CRUD uniques selon la quantité de données dans la base.
4.11	Distribution des performances des opérations CRUD multiples.
4.12	Performances des opérations CRUD multiples selon la quantité de données dans la base.
4.13	Distribution des performances des opérations CRUD uniques.
4.14	Performances des opérations CRUD uniques selon la quantité de données dans la base.
4.15	Distribution des performances des opérations CRUD multiples.
4.16	Performances des opérations CRUD multiples selon la quantité de données dans la base.
4.17	Distribution des performances des opérations CRUD uniques.
4.18	Performances des opérations CRUD uniques selon la quantité de données dans la base.
4.19	Distribution des performances des opérations CRUD multiples.

- 4.20 Performances des opérations CRUD multiples selon la quantité de données dans la base.
 - 4.21 Distribution des performances des opérations CRUD uniques.
 - 4.22 Performances des opérations CRUD uniques selon la quantité de données dans la base.
 - 4.23 Distribution des performances des opérations CRUD multiples.
 - 4.24 Performances des opérations CRUD multiples selon la quantité de données dans la base.
-
- 5.1 Distribution des performances des opérations CRUD uniques.
 - 5.2 Performances des opérations CRUD uniques selon la quantité de données dans la base.
 - 5.3 Distribution des performances des opérations CRUD multiples.
 - 5.4 Performances des opérations CRUD multiples selon la quantité de données dans la base.
 - 5.5 Distribution des performances des opérations CRUD uniques.
 - 5.6 Performances des opérations CRUD uniques selon la quantité de données dans la base.
 - 5.7 Distribution des performances des opérations CRUD multiples.
 - 5.8 Performances des opérations CRUD multiples selon la quantité de données dans la base.
 - 5.9 Distribution des performances des opérations CRUD uniques.
 - 5.10 Performances des opérations CRUD uniques selon la quantité de données dans la base.
 - 5.11 Distribution des performances des opérations CRUD multiples.
 - 5.12 Performances des opérations CRUD multiples selon la quantité de données dans la base.
 - 5.13 Distribution des performances des opérations CRUD uniques.
 - 5.14 Performances des opérations CRUD uniques selon la quantité de données dans la base.
 - 5.15 Distribution des performances des opérations CRUD multiples.
 - 5.16 Performances des opérations CRUD multiples selon la quantité de données dans la base.

Chapitre 1

Introduction

Ce rapport présente une étude comparative des performances entre MySQL et MongoDB, deux systèmes de gestion de bases de données (SGBD) très utilisés, en mettant l'accent sur les opérations CRUD (Create, Read, Update, Delete).

L'objectif est de tester différentes configurations et scénarios d'utilisation pour analyser l'impact des index, de la réPLICATION, et de la fragmentation sur les performances.

La structure des tests, ainsi que les résultats obtenus, sont décrits en détail.

Chapitre 2

Contexte et Objectifs

I

Systèmes Comparés

I.1

MongoDB

MongoDB est un SGBD NoSQL orienté documents. Contrairement aux bases relationnelles, MongoDB stocke les données sous forme de documents JSON ou BSON, offrant une grande flexibilité structurelle. Ses caractéristiques principales incluent :

- **Stockage orienté documents** : Les données sont regroupées en collections, chaque document étant un objet JSON avec une structure flexible.
- **Requêtes riches** : MongoDB prend en charge des requêtes complexes incluant des opérations d'agrégation et des jointures.
- **Scalabilité horizontale** : Avec la fragmentation (sharding), les données peuvent être réparties sur plusieurs nœuds.
- **RéPLICATION** : MongoDB utilise des Replica Sets pour garantir la haute disponibilité et la tolérance aux pannes.
- **Indexation avancée** : MongoDB offre un support pour les index secondaires et composites afin d'optimiser les performances des requêtes.

I.2

MySQL

MySQL est un SGBD relationnel open-source largement utilisé dans les applications web et les systèmes d'information. Ses caractéristiques principales incluent :

- **Modèle relationnel** : Les données sont stockées dans des tables avec des relations définies par des clés étrangères.
- **Transactions ACID** : MySQL garantit l'atomicité, la cohérence, l'isolation et la durabilité des transactions.
- **Indexation** : Prise en charge d'index pour optimiser les performances des requêtes.
- **RéPLICATION** : MySQL utilise un modèle maître-esclave pour répliquer les données entre plusieurs serveurs.
- **Clusterisation** : MySQL Cluster permet de distribuer les données sur plusieurs nœuds

II

Étude des aspects de MongoDB

II.1

Comparaison des Performances des Requêtes

MongoDB se distingue des bases relationnelles comme MySQL par sa capacité à exécuter des requêtes sur des données semi-structurées et non structurées. La comparaison des performances des requêtes révèle :

- MongoDB excelle dans les scénarios où les données sont semi-structurées ou nécessitent une mise à jour fréquente de leur structure.
- MySQL, en revanche, est plus performant pour les requêtes transactionnelles complexes grâce à son moteur ACID robuste.
- Les schémas flexibles de MongoDB permettent une évolutivité plus facile pour les applications en croissance.
- La mise à l'échelle horizontale de MongoDB via la fragmentation (sharding) est un avantage pour les applications nécessitant une haute disponibilité et une distribution des données.
- Tolérance aux pannes : MongoDB offre une haute disponibilité des données grâce à la réPLICATION automatique et aux élections de nœuds primaires.
- Les performances de lecture/écriture peuvent être optimisées en configurant les politiques de réPLICATION et de fragmentation.

II.2

Performances avec et sans Indexation

L'indexation est un facteur clé des performances :

- **MongoDB sans indexation** : Les requêtes nécessitent un balayage complet des documents, ce qui ralentit considérablement les performances.
- **MongoDB avec indexation** : Les performances des lectures augmentent considérablement grâce à la réduction du nombre de documents scannés.
- **MySQL avec indexation** : Étant donné sa nature relationnelle, MySQL exploite efficacement les index pour optimiser les lectures et écritures.

II.3

RéPLICATION ET FRAGMENTATION

La réPLICATION et la FRAGMENTATION influencent directement les performances et la tolérance aux pannes.

RéPLICATION dans MongoDB :

- **RéPLICATION primaire-secondaire** : Permet une haute disponibilité des données. Les écritures se font uniquement sur le nœud primaire, tandis que les lectures peuvent être réparties sur les nœuds secondaires.
- **Élections automatiques** : En cas de panne du nœud primaire, un nouveau primaire est élu parmi les secondaires, garantissant une continuité de service.
- **Impact sur les performances** : La latence d'écriture augmente avec un write concERNé élevé, car l'écriture doit être confirmée par plusieurs nœuds.

FRAGMENTATION dans MongoDB :

- **Distribution des données** : Les documents sont répartis entre plusieurs shards, permettant de gérer efficacement de grandes volumes.
- **Impact sur les performances** : La fragmentation améliore les temps de réponse pour les lectures et écritures parallèles, mais peut augmenter la complexité des requêtes agrégées.

Chapitre 3

Structure des Tests

I

Architecture MongoDB

Pour les tests de performance, nous avons utilisé trois architectures MongoDB différentes : Standalone, Replica Set, et Sharded Cluster. Pour MySQL, nous avons testé deux configurations : Standalone et Cluster. Ceci afin d'estimer l'impact du choix de l'architecture sur les performances des opérations CRUD.

I.1

Architecture avec serveur unique : Standalone

Un seul serveur MongoDB.

Standalone MySQL/MongoDB Server

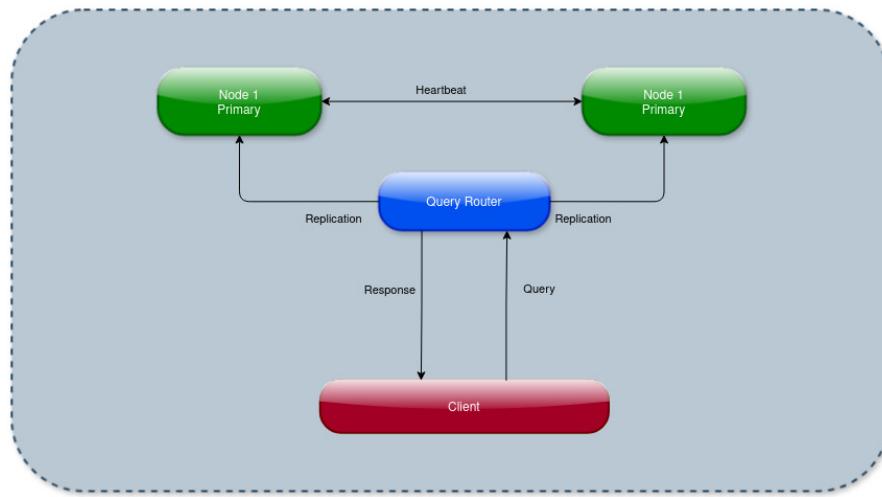
Query /Response

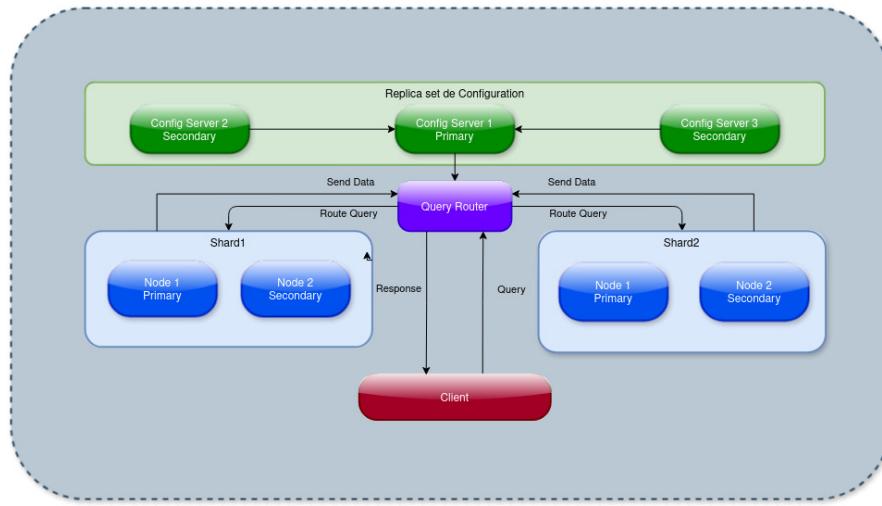
Client

I.2

Architecture avec réPLICATIONS : Replica Set

Un Replica Set avec un nœud primaire et deux nœuds secondaires.





I.3

Architecture Fragmentée : shards

Un cluster avec trois Serveurs de configuration, un Query router, et deux shards constitués chacun de deux nœuds. On aurait pu mettre 3 noeuds par shard, mais pour des raisons de ressources, on a opté pour 2.

II

Architecture MySQL

II.1

Architecture avec serveur unique : Standalone

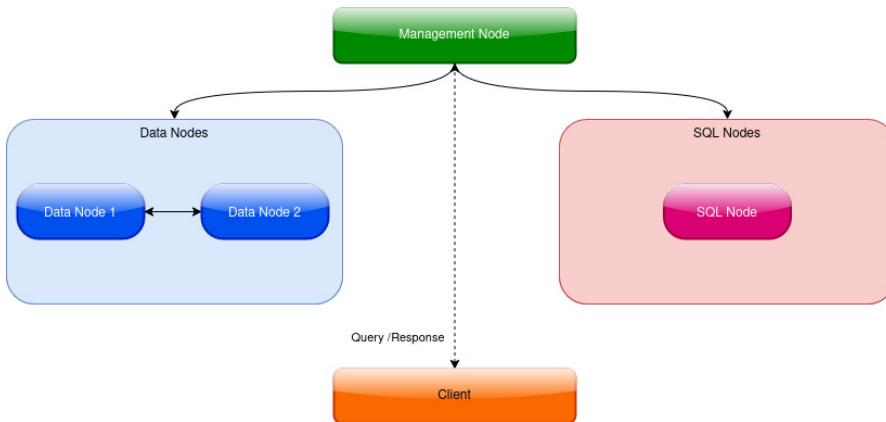
Un serveur MySQL simple. Voir la figure 3.1.

II.2

Architecture Distribuée : Cluster

Architecture MySQL-NDB avec :

- Un gestionnaire (manager).
- Deux nœuds de données.
- Un noeud SQL.



III Description des Tests

III.1 Contexte

Avant l'exécution des tests, il est nécessaire de générer les données de test.

Cela est possible grâce au script Python `generate_data.py` qui génère des données aléatoires pour les tests de performance.

Ce script repose sur l'utilisation du module `Faker` pour générer des données aléatoires pour les tests.

Une graine a été employée afin d'avoir des résultats reproductibles.

Pour MongoDB et MySQL respectivement, Les tests de performance sont réalisés en utilisant des scripts Python (`mongodb.py`, `mysql.py`) et les modules clients `pymongo`, `pymysql`.

Ces scripts effectuent des opérations et mesurent les temps d'exécution des opérations CRUD sur les bases de données.

Notons que le module `pymongo` intègre un sous module pour monitorer les temps d'exécution des requêtes pour les tests, reposant sur les retours de MongoDB.

En revanche, pour MySQL, les temps d'exécution sont mesurés en utilisant le module en local en mesurant l'intervalle de temps entre l'envoi et la réception des requêtes.

Ainsi, les tests dépendent également en partie de la latence réseau, néanmoins, le docker compose `docker-compose.yml` permet de déployer des serveurs en local et de s'affranchir de cette limite.

Enfin, le module `matplotlib` est utilisé pour générer des graphiques à partir des résultats des tests.

III.2

Principe des tests

Il y a fondamentalement 2 types de tests :

- Tests globaux : N Opérations CRUD successives sur l'ensemble des données.
- Tests avec variation de quantité de données initiales : Opérations CRUD données avec variation du nombre de données initiales.

Les tests globaux visent à mesurer les statistiques globales des opérations successives de N insertions, N mises à jour, N lectures, puis N suppression de données.

Les tests avec variation de quantité de données initiales insèrent en premier lieu, un nombre x_n de données initiales.

Ensuite, on mesure le temps d'exécution d'une opération d'insertion, puis de mise à jour, de lecture et de suppression de données.

On répète ces opérations n fois pour des quantités de données initiales x_n différentes entre 0 et N .

Dès lors, on s'intéresse à la différence de performance entre une opération CRUD avec des données uniques et des données multiples.

Par exemple, on regarde la différence entre 10 insertions de 10 données distinctes une par une, et l'insertion de 10 données en une seule fois.

Ainsi, ces tests effectueront $\frac{N}{n_donnees_par_operation}$ opérations CRUD.

Finalement, on voudrait mesurer l'impact de l'indexation sur les performances des opérations CRUD.

On fera donc au total 8 tests par infrastructure de base de données, soit 24 tests au total.

En effet, on réplique les 2 types de tests pour des opérations uniques et des opérations multiples, et cela avec et sans indexation.

III.3 Paramètres des tests**III.4 Structure des Données**

Afin de tenter de mesurer les performances des bases de données, sans favoriser l'une ou l'autre, les données générées sont les mêmes pour les deux bases de données.

Elles sont génériques et construites pour éviter d'exploiter des fonctionnalités spécifiques à l'une ou l'autre base de données.

Elles intègrent différents types de données, tels que des chaînes de caractères, des dates, des nombres entiers et des nombres flottants.

Ainsi, La table/collection "test" utilisée pour les tests, regroupe des données avec la structure suivante :

- **id** : Identifiant unique (UUID ou auto-incrément pour MySQL, ObjectId pour MongoDB).
- **title** : Titre du livre (String).
- **author** : Auteur du livre (String).
- **published_date** : Date de publication (Date).
- **genre** : Genre du livre (String).
- **price** : Prix du livre (Float).
- **copies_sold** : Nombre d'exemplaires vendus (Integer).
- **ran** : Champ aléatoire pour les tests entre 0 et num_records_per_many – 1 (Integer).

Ce qui donne pour MySQL :

```
CREATE TABLE test (
    id INT ,
    title VARCHAR(255),
    author VARCHAR(255),
    published_date DATE,
    genre VARCHAR(255),
    price FLOAT,
    copies_sold INT,
    ran INT
);
```

Chapitre 4

Résultats et Analyses pour MongoDB

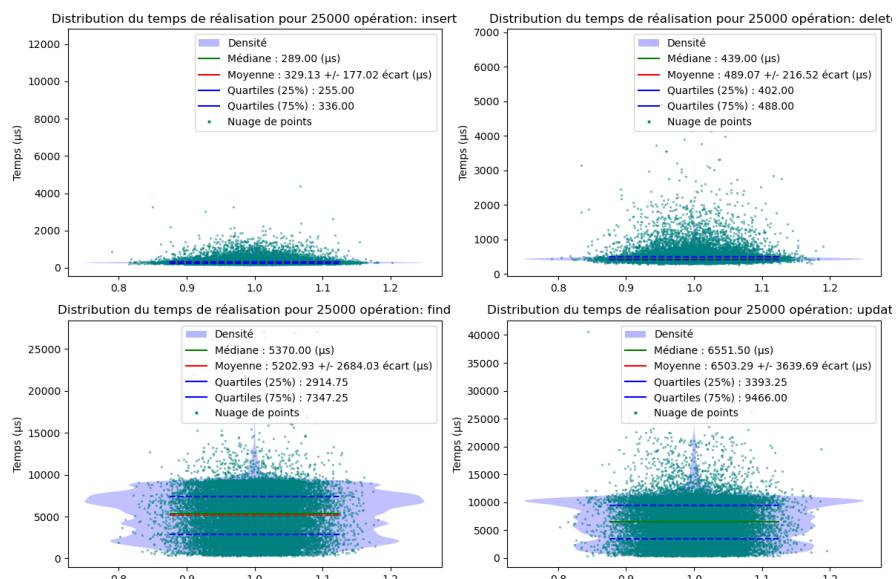
I

MongoDB Standalone

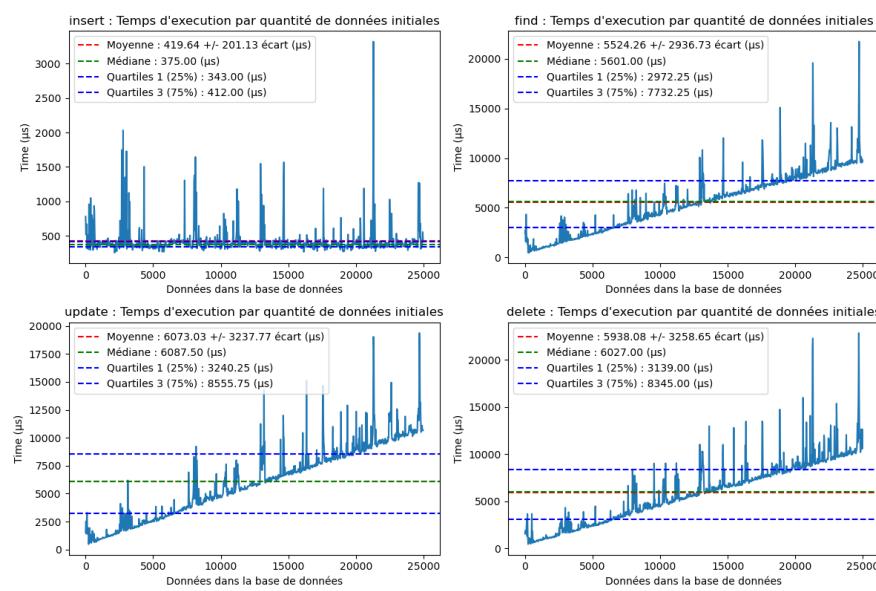
I.1

Performances des Opérations uniques

Distribution des performances des opérations CRUD uniques



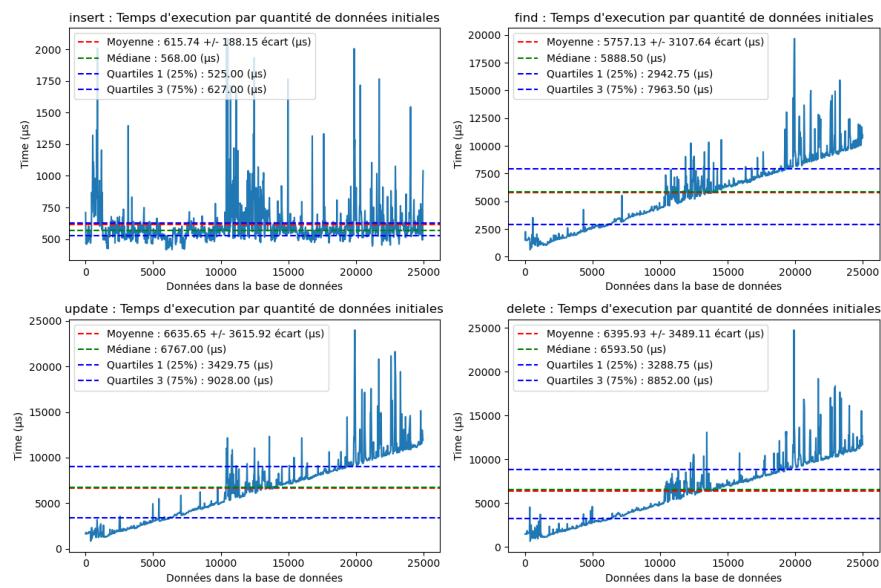
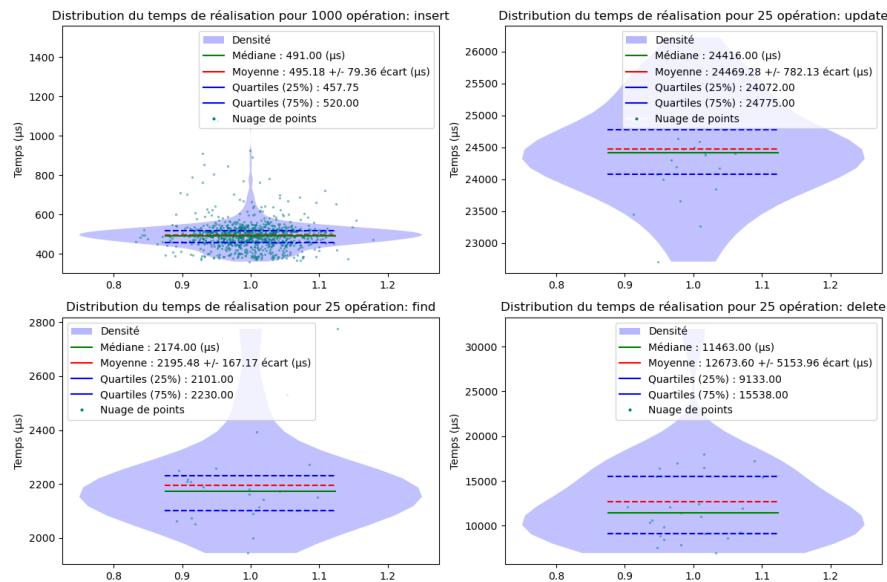
Performances des Opérations uniques selon la quantité de données dans la base



Les résultats montrent que l'insertion des données se fait en temps quasiment constant, tandis que les autres opérations évoluent linéairement avec la quantité de données. Cela reflète une augmentation des coûts de recherche dans les collections non indexées.

I.2

Performances des Opérations avec données multiples



Les résultats sont semblables et assez proches numériquement de ceux des opérations uniques.

On remarque néanmoins que les temps d'execution sont plus longs pour les opérations multiples, ce qui est attendu.

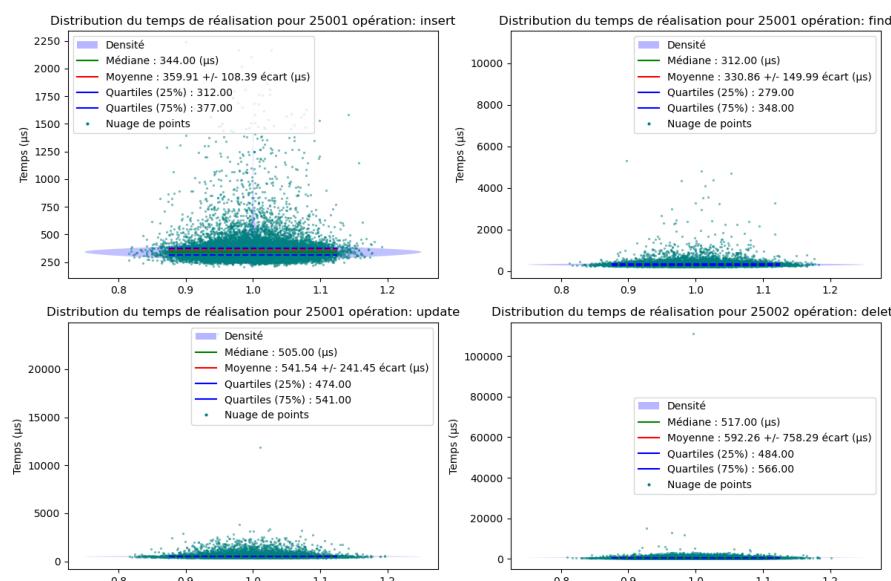
La différence n'est certainement pas assez notable car trop peu de données sont opérer en simultanées (25 données à la fois) pour que cela ait un impact significatif.

néanmoins, on remarque que les temps d'execution augmentent plus ainsi que le variance pour la modification de données.

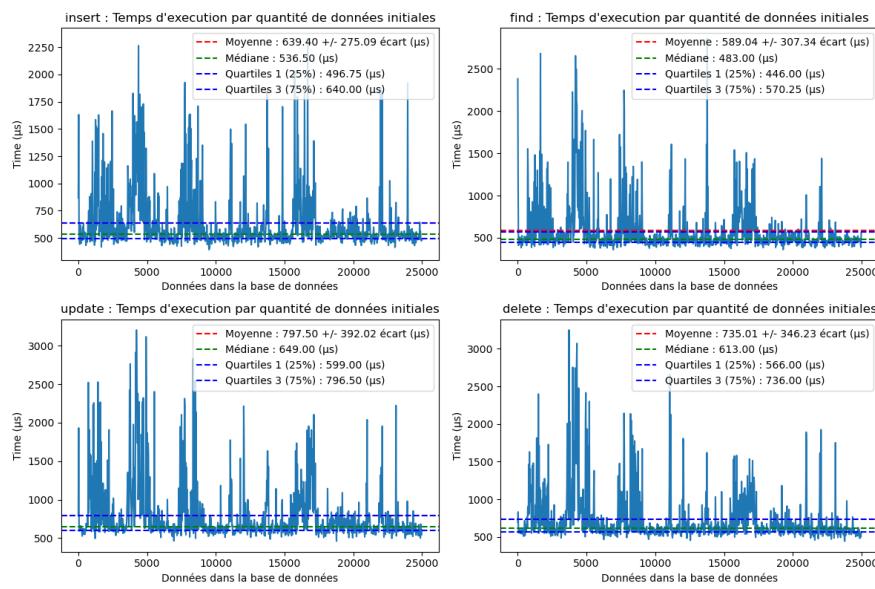
Cependant, rapporté au nombre de données sur lequel on opère en une fois et au temps proche de l'opération unique,
on peut en conclure que les opérations multiples sont plus intéressantes que les opérations uniques dans le cas sans indexation.

I.3 Indexation

Performances des Opérations uniques



CHAPITRE 4. RÉSULTATS ET ANALYSES POUR MONGODB



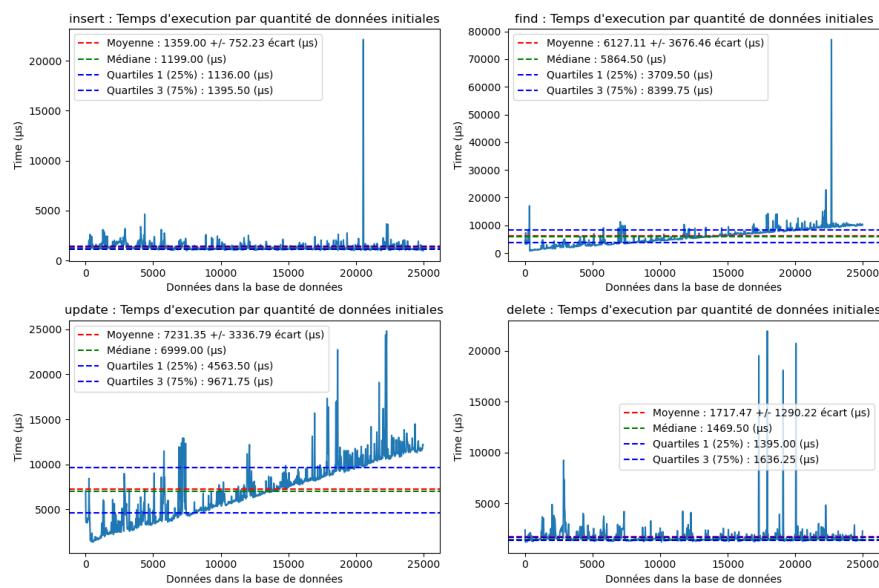
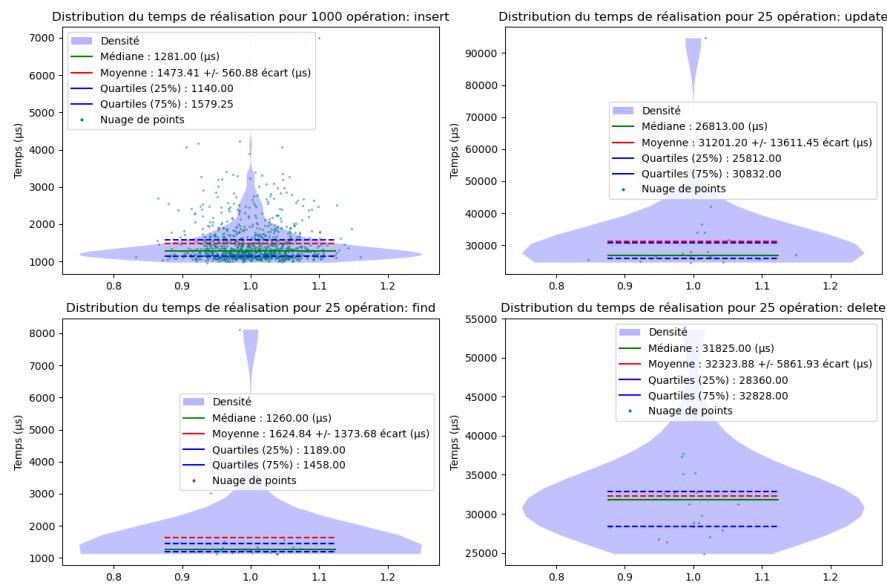
On observe ici une nette amélioration des performances pour les opérations de lecture et de mise à jour, et suppression (près de 10 fois plus rapide en moyenne).

En ce qui concerne l'insertion, les performances sont proches mais légèrement plus lentes.

L'évolution n'est plus linéaire, ce qui montre l'efficacité de l'indexation, les opérations sont faites presques en temps constant en moyenne.

On observe simplement des pics de variations, due à l'indexation. Néanmoins l'écart type reste comparable au cas sans indexation.

Performances des Opérations avec données multiples



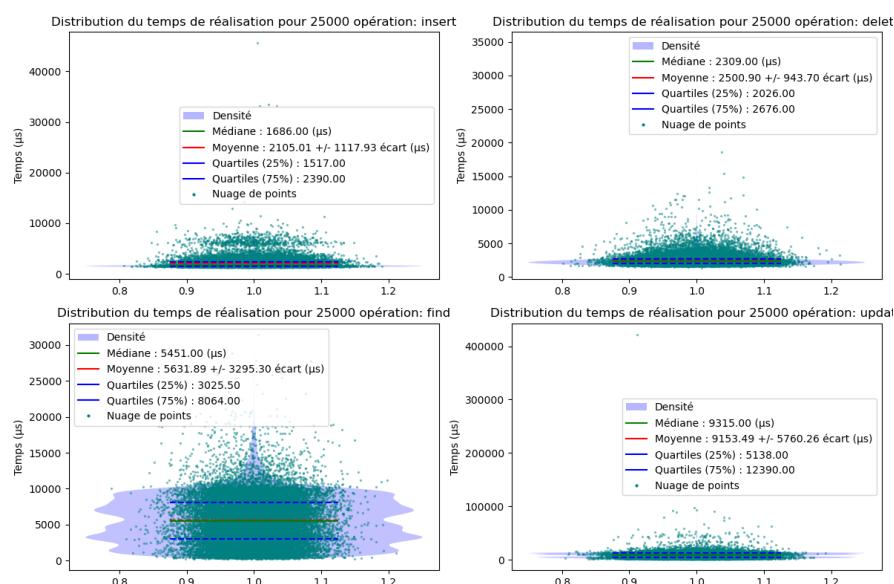
Ici, les observations se tiennent pour l'insertion et la suppression. En revanche, on observe une évolution linéaire légèrement coefficienté qui semble s'amortir pour la modification et la lecture. Les performances affichées semblent bien moins bonnes que pour les opérations uniques, mais rapporté au nombre de données sur lequel on opère (25) en une fois, on peut en conclure que les opérations multiples reste plus intéressantes que les opérations uniques dans le cas avec indexation. Néanmoins, il faudrait certainement trouver un point d'équilibre (le nombre de données sur lequel on opère en une fois) pour optimiser les performances.

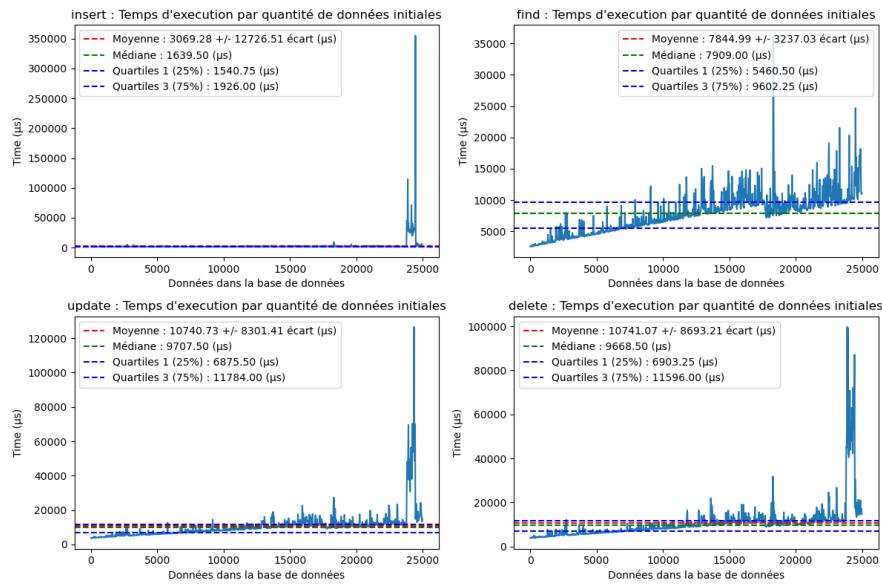
II

RéPLICATION

II.1

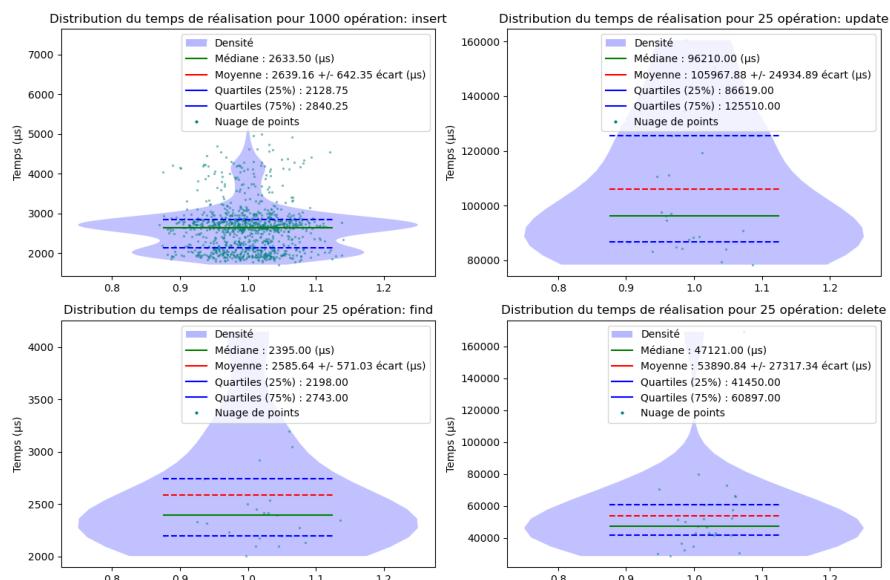
Performances des Opérations uniques

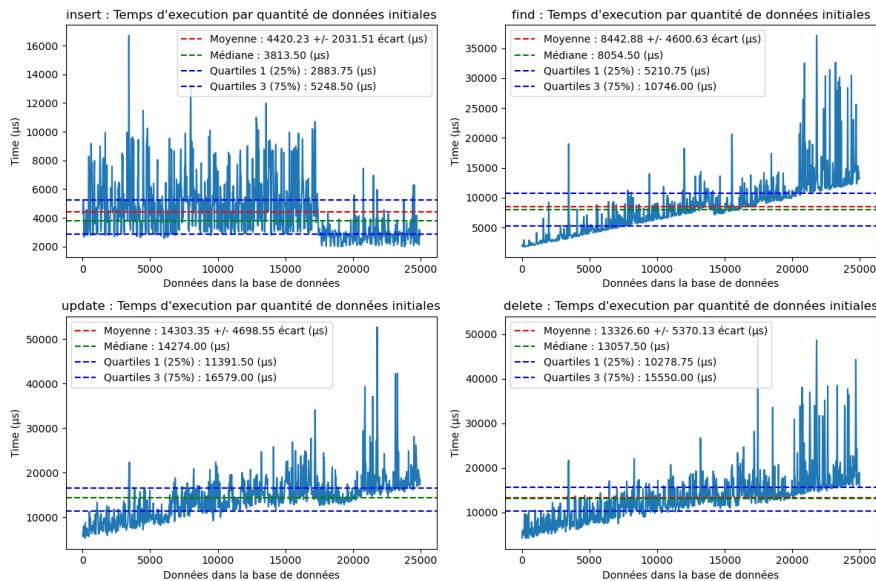




II.2

Performances des Opérations avec données multiples

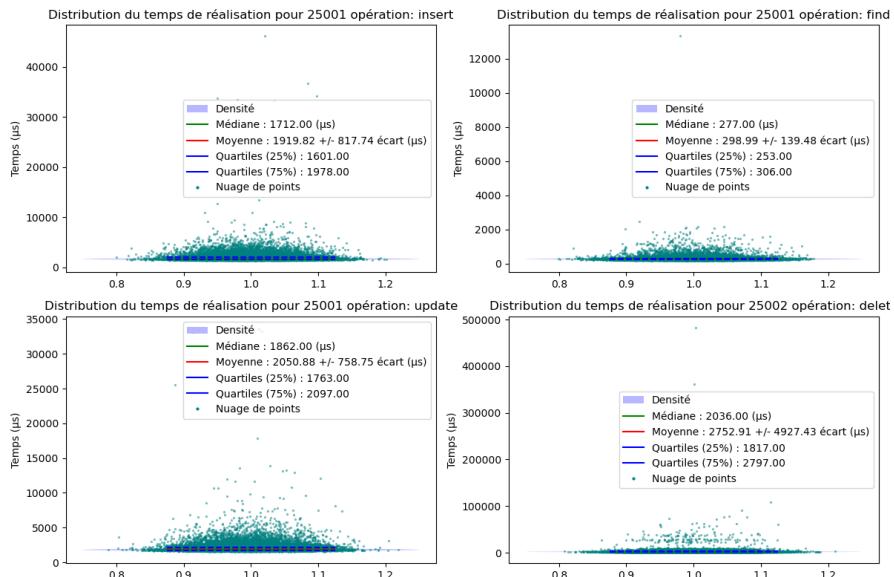




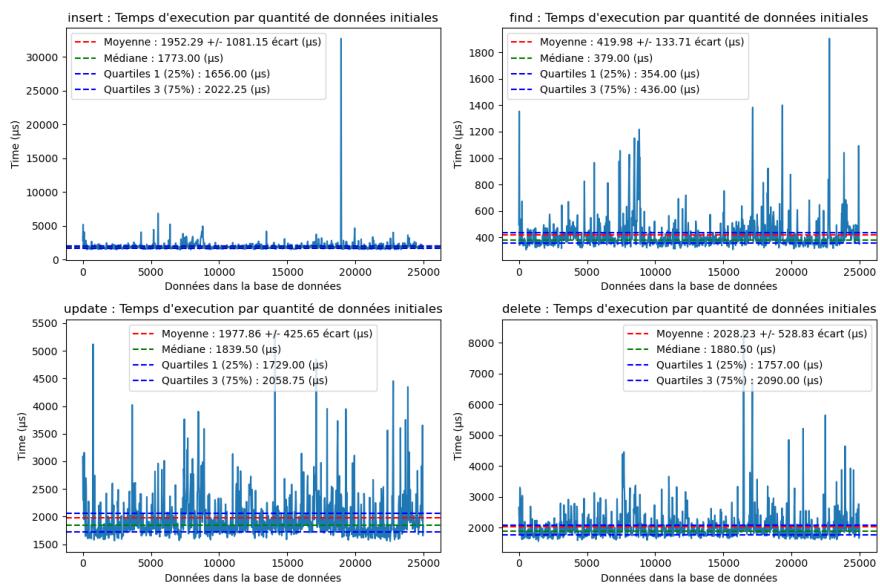
II.3

Indexation

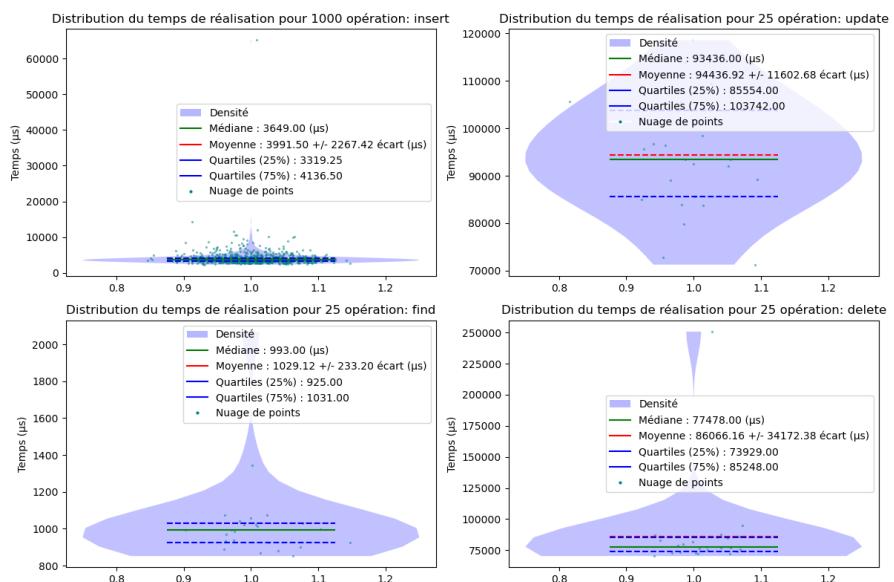
Performances des Opérations uniques

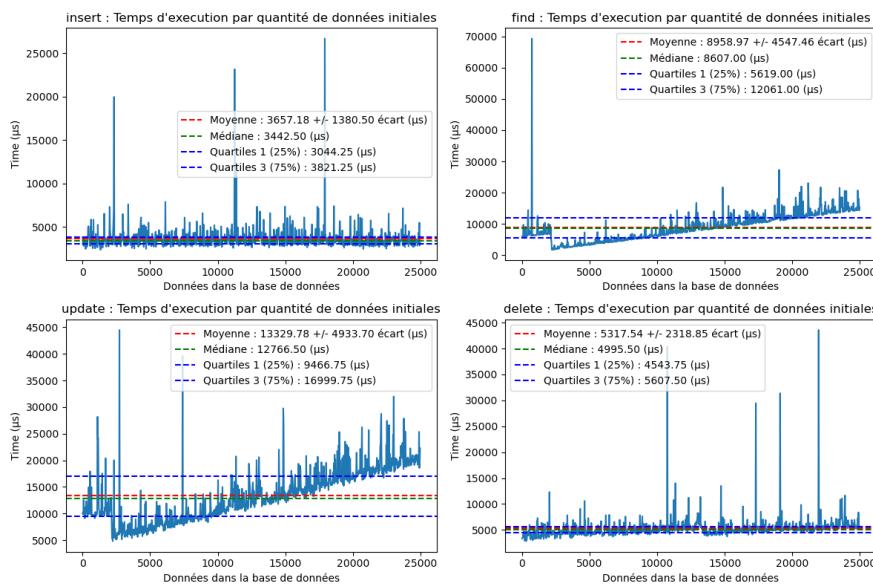


CHAPITRE 4. RÉSULTATS ET ANALYSES POUR MONGODB



Performances des Opérations avec données multiples





II.4

Observations Globales sur la RéPLICATION

Les résultats montrent que la réPLICATION a un impact sur les performances des opérations CRUD.

Les opérations de lecture et de mise à jour sont plus lentes en moyenne, tandis que l'insertion et la suppression sont plus rapides.

Cela est dû au fait que les opérations de lecture et de mise à jour doivent être confirmées par plusieurs nœuds, ce qui augmente la latence.

En revanche, l'insertion et la suppression peuvent être effectuées sur un seul nœud, ce qui réduit la latence.

L'indexation améliore les performances des opérations de lecture et de mise à jour, mais a un impact négligeable sur l'insertion et la suppression.

Les opérations multiples sont plus lentes que les opérations uniques, mais restent plus efficaces pour traiter de grandes quantités de données.

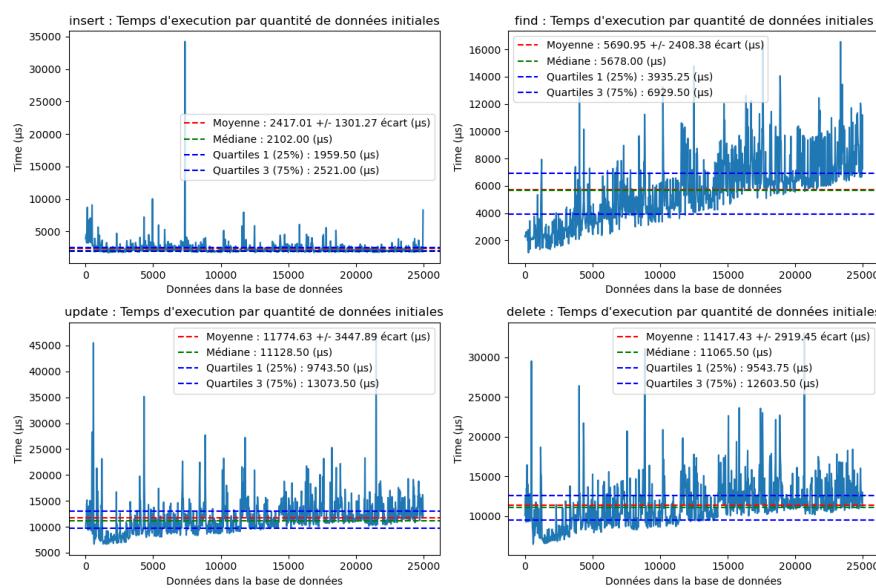
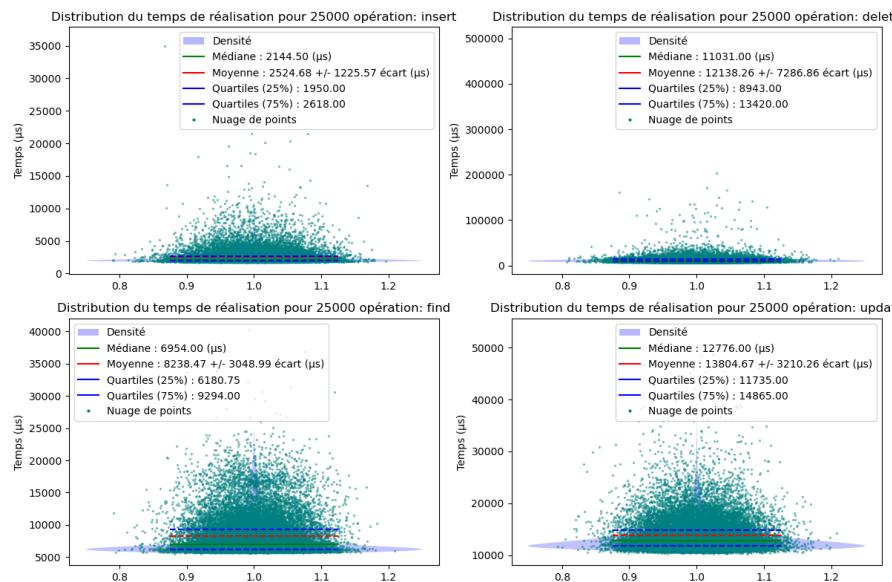
Enfin, les performances des opérations CRUD sont plus stables avec l'indexation, ce qui réduit la variance des temps d'exécution.

III

Sharding

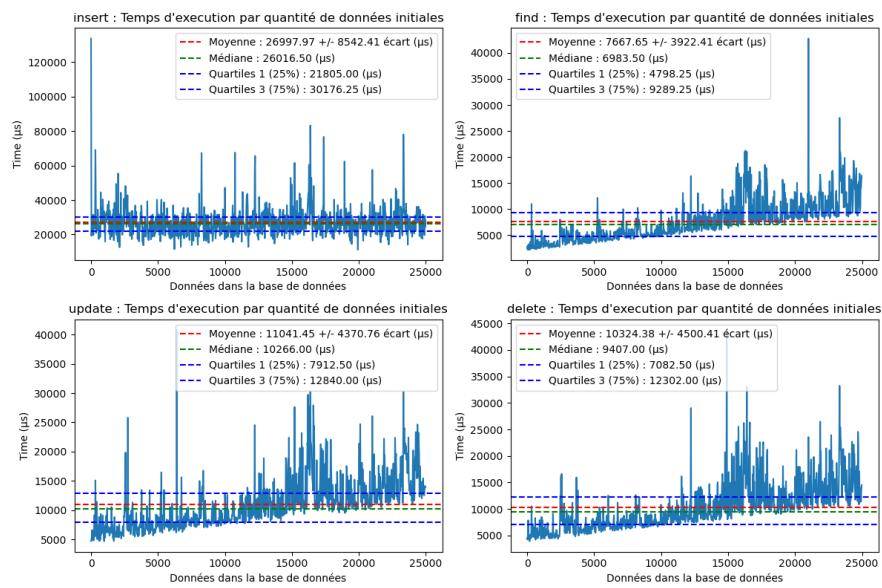
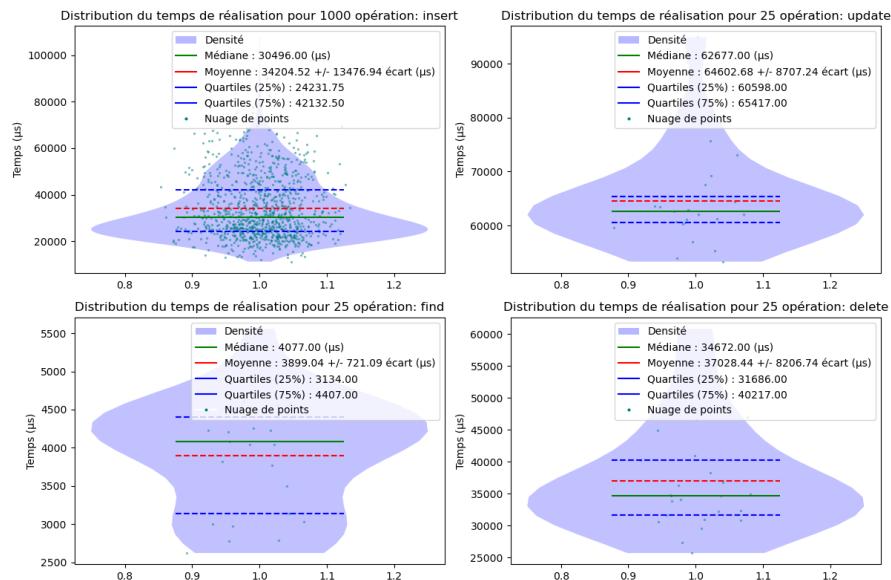
III.1

Performances des Opérations uniques



III.2

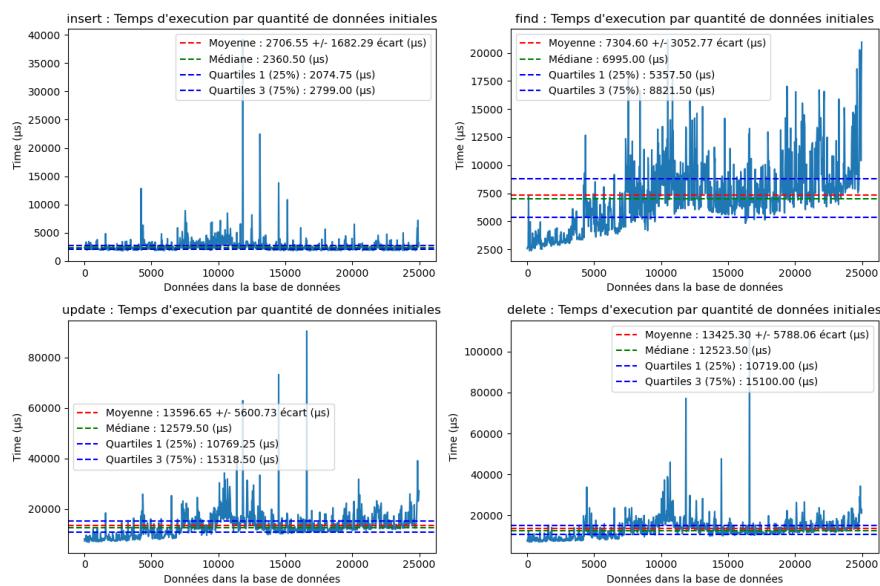
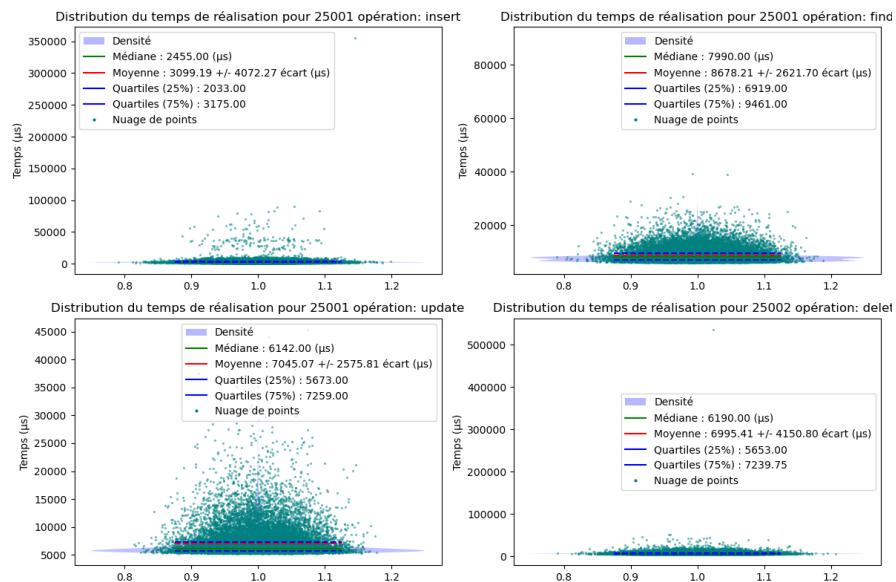
Performances des Opérations avec données multiples



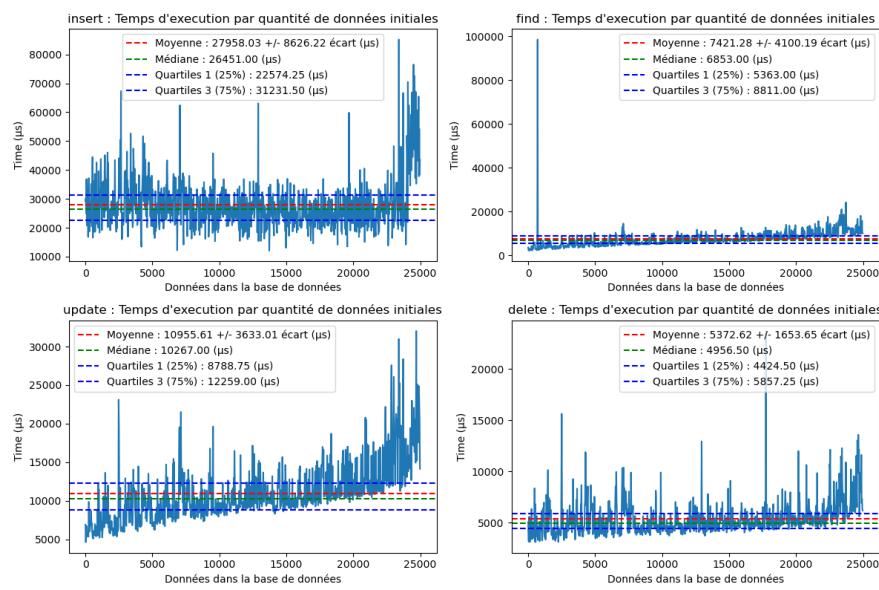
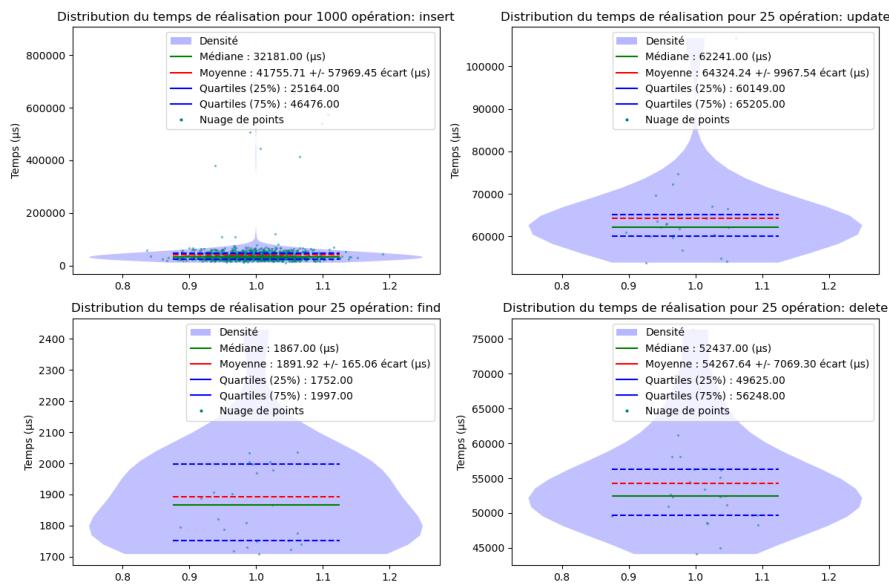
III.3

Indexation

Performances des Opérations uniques



Performances des Opérations avec données multiples



III.4

Observations Globales sur la fragmentation

Les résultats montrent que la fragmentation améliore les performances des opérations CRUD par rapport à la réPLICATION.

Les opérations sont plus rapides ou quasiment aussi performantes et surtout plus stables. Cela est dû au fait que les opérations de lecture et de mise à jour peuvent être parallélisées sur plusieurs nœuds, tandis que l'insertion et la suppression nécessitent une coordination entre les nœuds.

Entre la version indexée et non indexée, on n'observe pas d'amélioration significative des performances.

Cela peut être du au fait que le shard se fait déjà sur un champ indexé (id) et que les opérations sont déjà optimisées et les index peu utiles, voir même contre-productifs.

Il faudrait certainement avoir des operations plus adaptées et un schéma d'indexation plus optimisée pour voir apparaître des différences significatives.

De plus, comparativement au mode standalone, les performances sont moindre contrairement aux attentes.

Cela peut être dû au fait que la configuration n'est pas optimisée, que les requêtes sont envoyées au noeud le plus proche et non au noeud primaires ce qui engendre des coûts de latence supplémentaires.

Néanmoins, le sharding reste intéressant, avec un configuration optimisée, pour des applications nécessitant une haute disponibilité et une distribution des données.

Chapitre 5

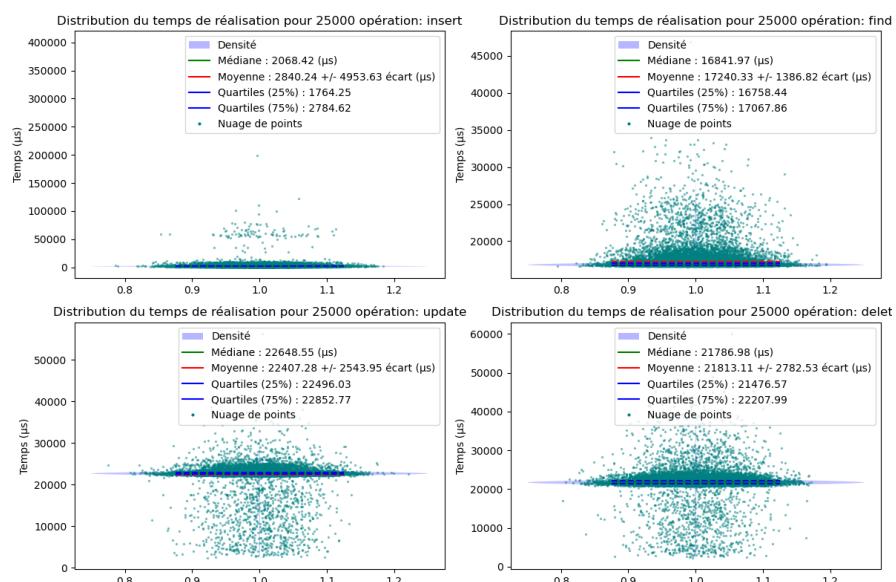
Résultats et Analyses pour MySQL

I

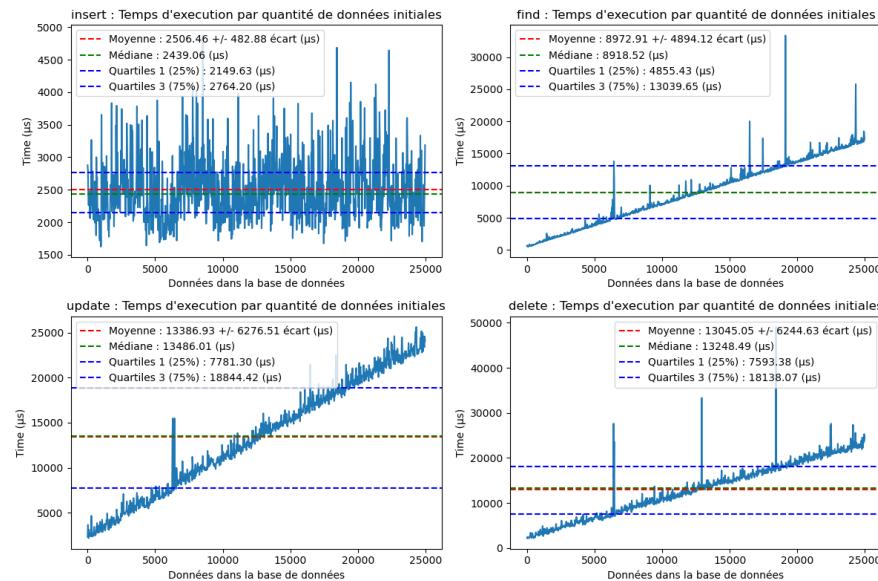
MySQL Standalone

I.1

Performances des Opérations uniques

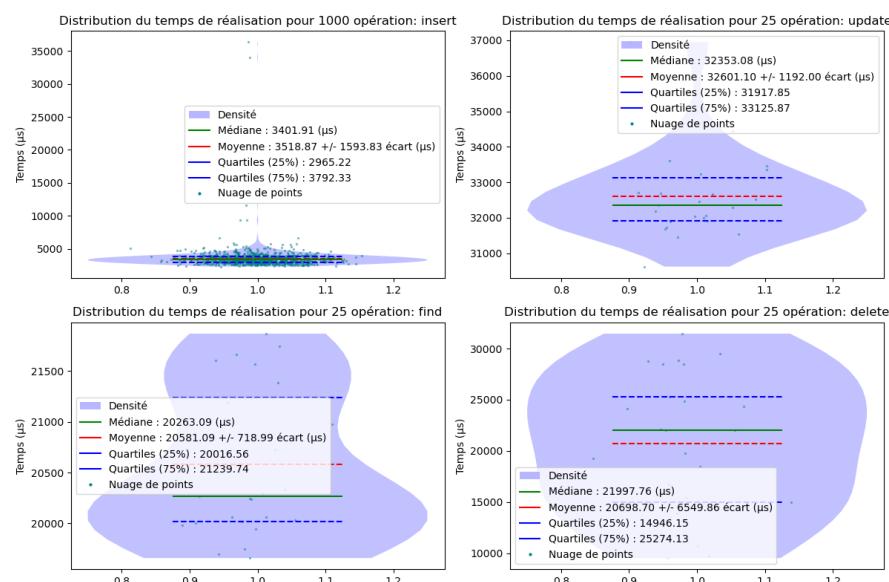


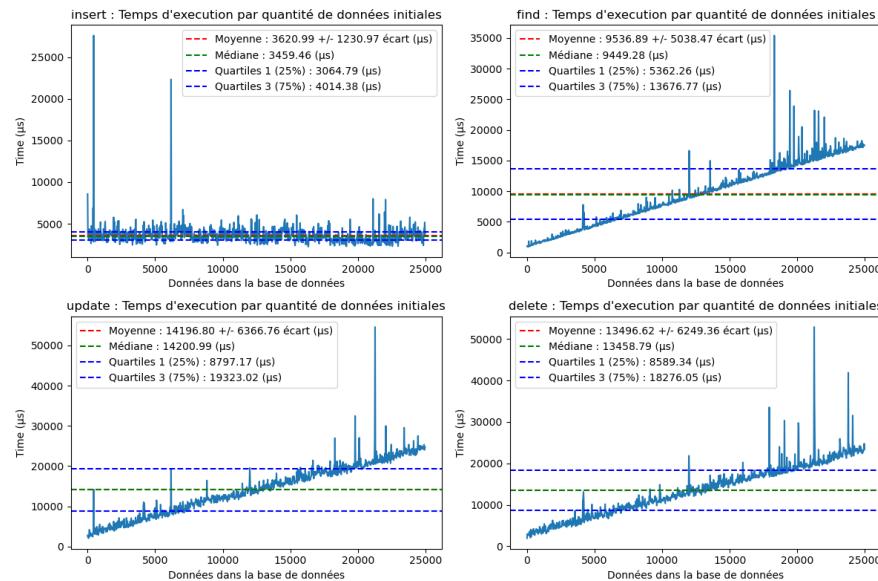
CHAPITRE 5. RÉSULTATS ET ANALYSES POUR MYSQL



I.2

Performances des Opérations avec données multiples

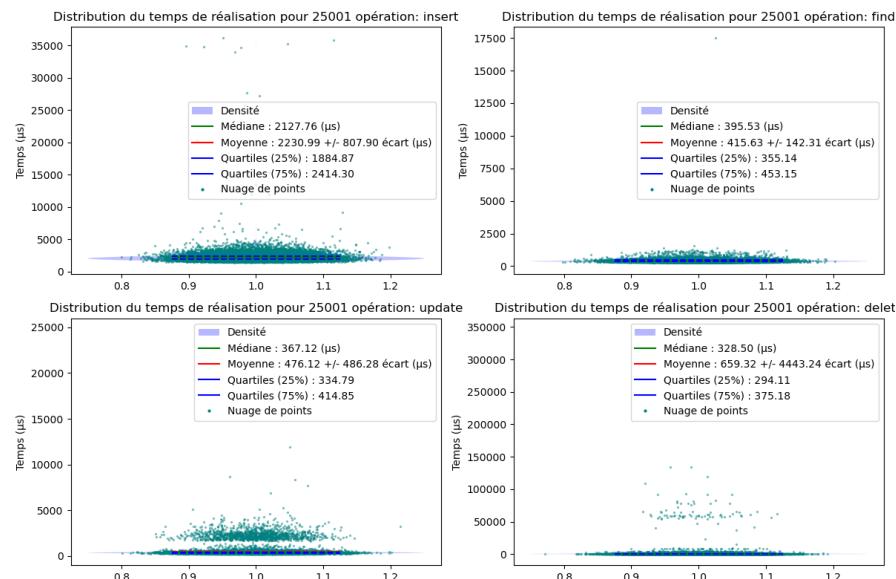




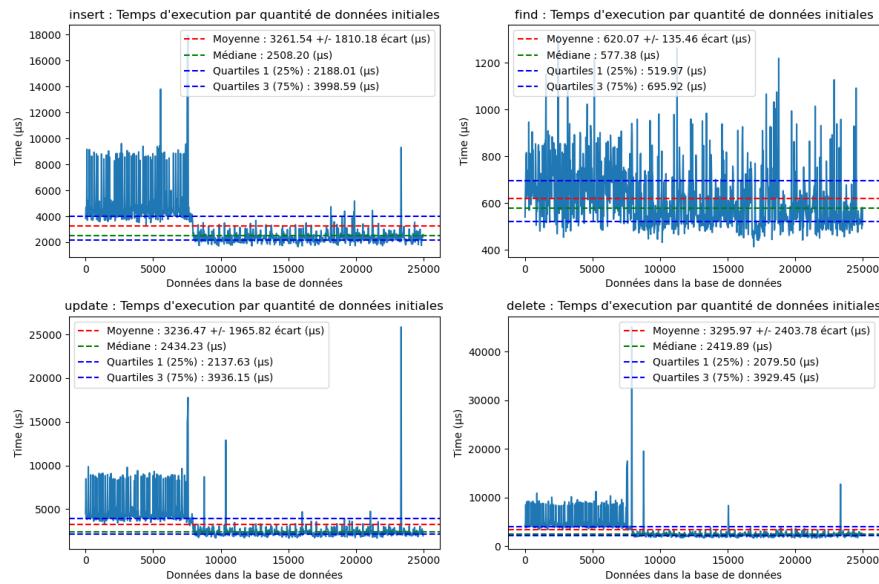
I.3

Indexation

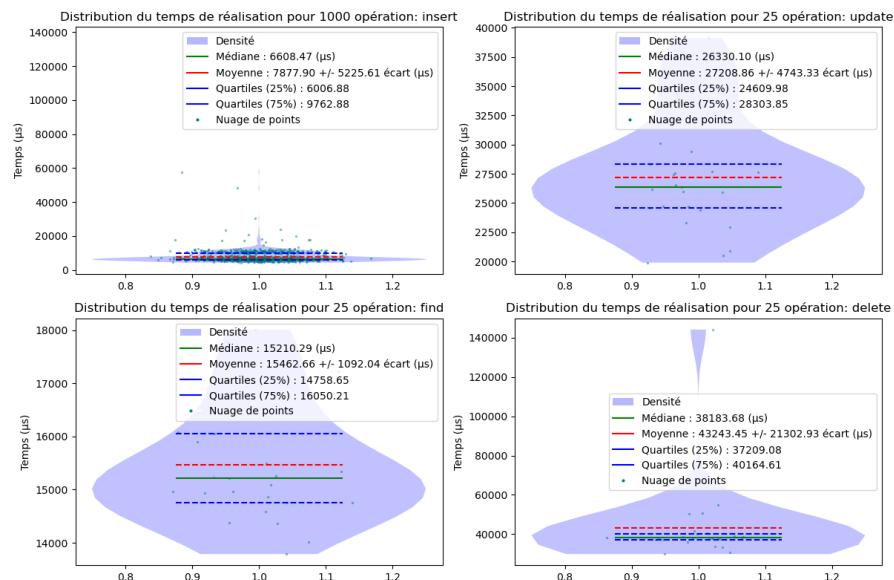
Performances des Opérations uniques

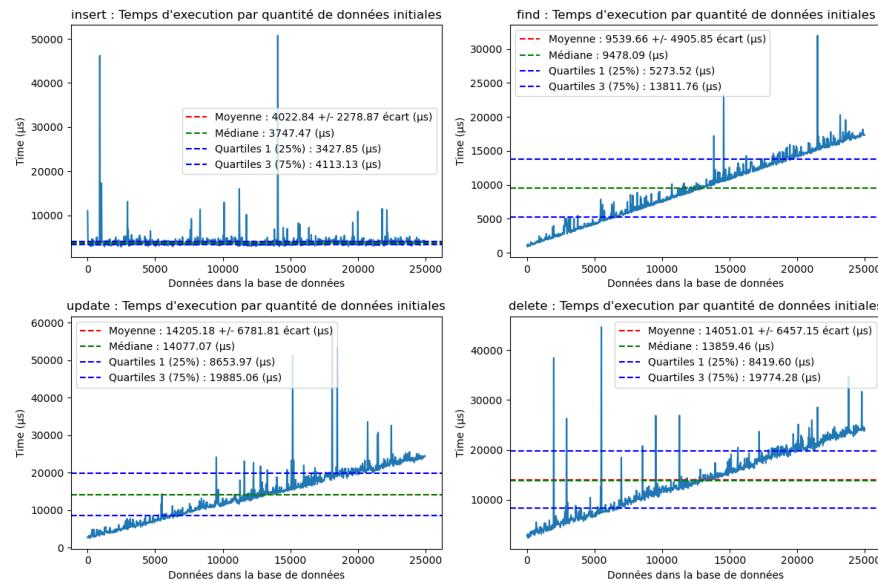


CHAPITRE 5. RÉSULTATS ET ANALYSES POUR MYSQL



Performances des Opérations avec données multiples





I.4

Observations Globales sur MySQL Standalone

Les résultats montrent que MySQL est moins performant sans indexation pour les opérations uniques que MongoDB.

Les observations selon la quantité de données dans la base sont similaires à celles de MongoDB, avec néanmoins un coefficient plus important dans le cas de la modification. Avec indexation, on observe pour les opérations à l'exception de la recherche, une variance importante, avec un temps d'exécution long, jusqu'à un certain point où les opérations deviennent constantes en moyennes et avec des performances similaires à MongoDB.

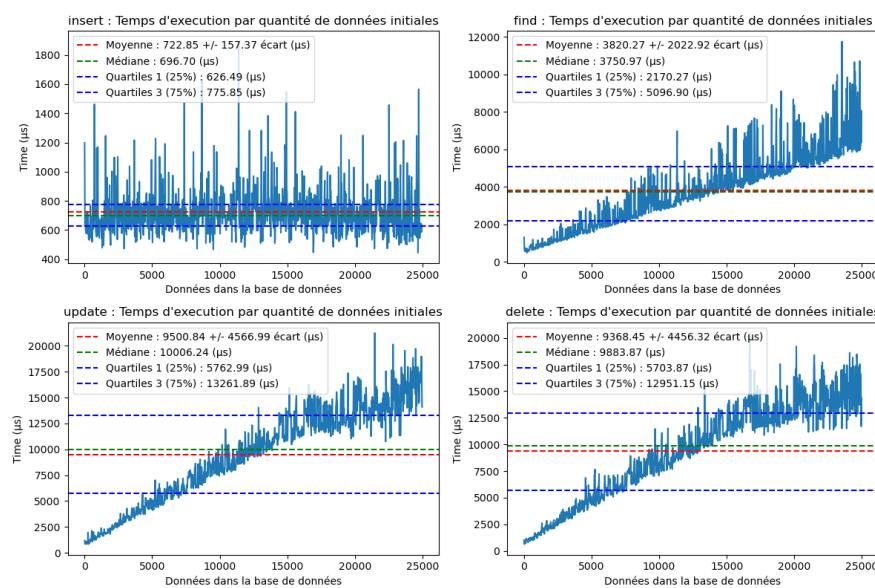
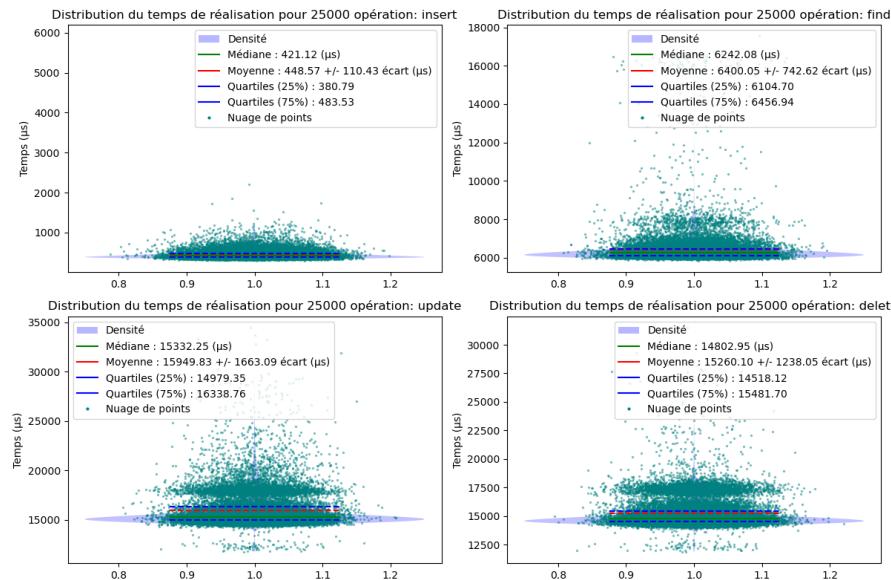
Néanmoins, des imprécisions peuvent être dues à la façon dont les temps d'exécution sont mesurés.

II

MySQL Cluster

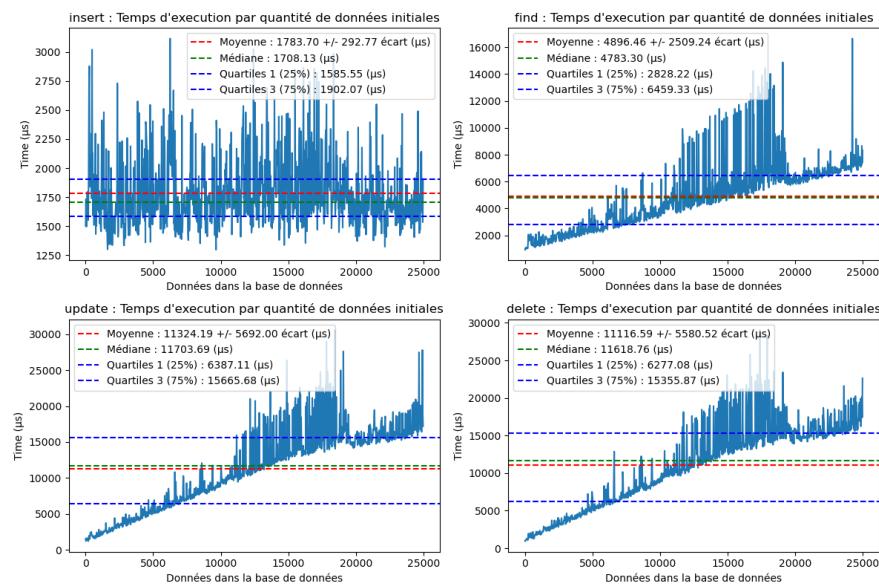
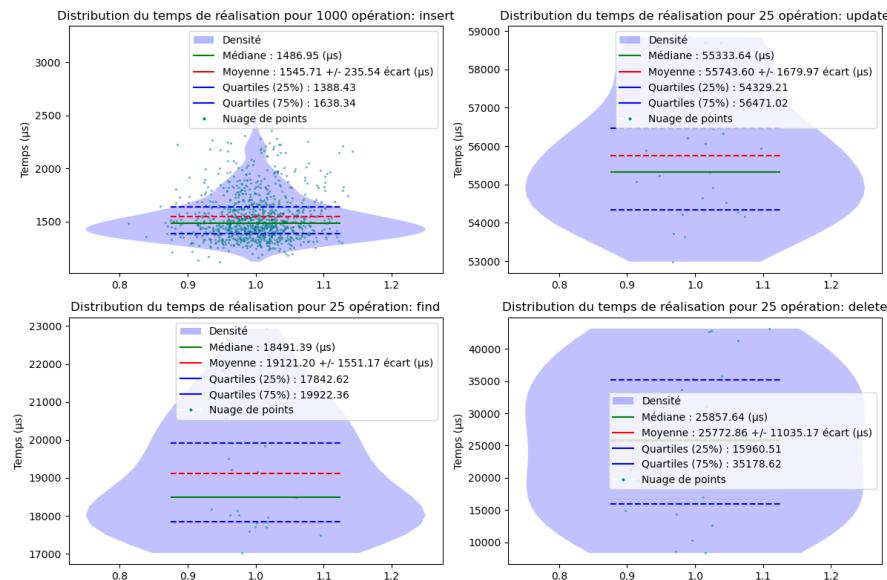
II.1

Performances des Opérations uniques



II.2

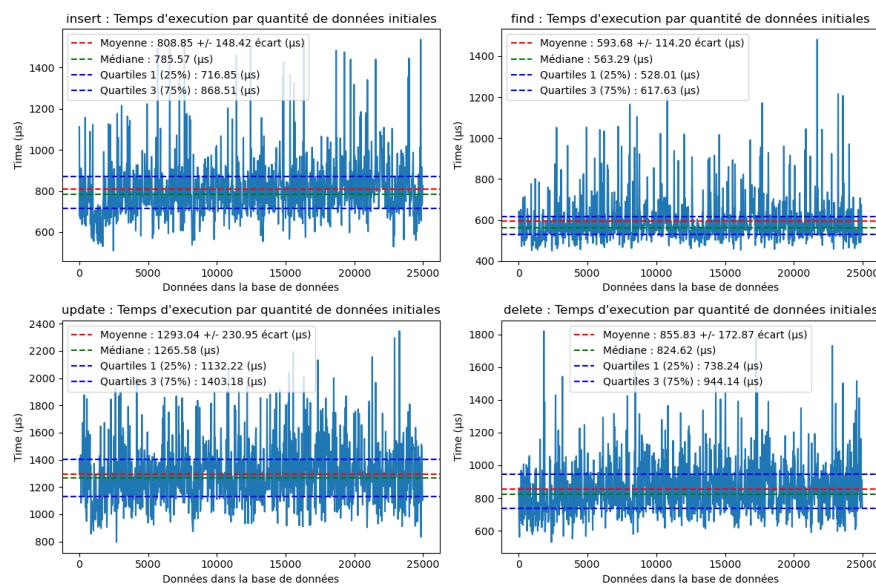
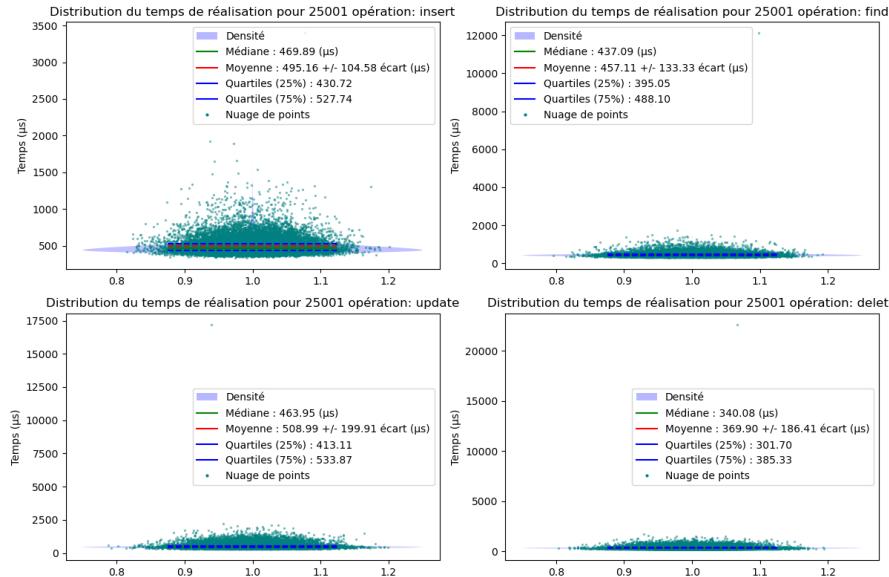
Performances des Opérations avec données multiples



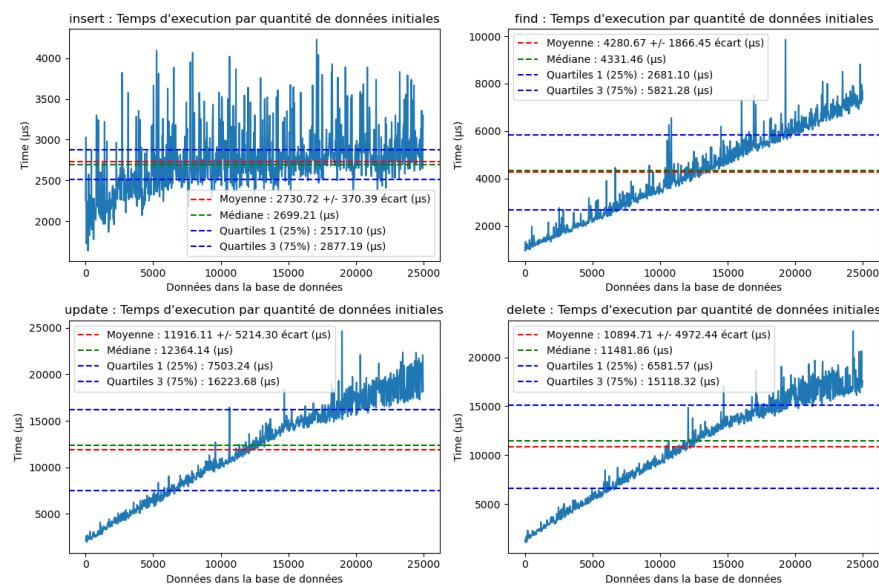
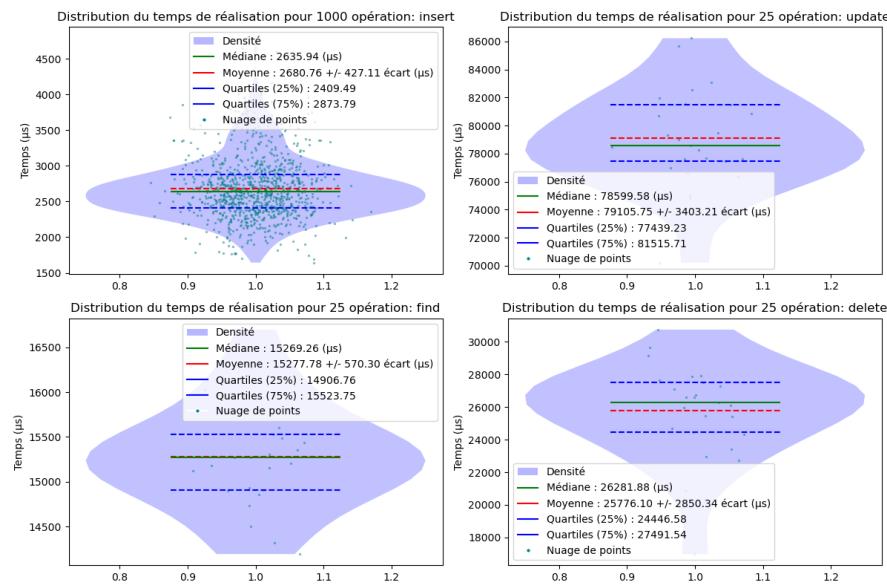
II.3

Indexation

Performances des Opérations uniques



Performances des Opérations avec données multiples



II.4

Observations Globales sur MySQL Cluster

Les résultats montrent que MySQL Cluster est plus performant que MySQL Standalone pour les opérations uniques.

Les performances sont comparables à celles de MongoDB, avec des temps d'exécution plus rapides pour les opérations de lecture et de mise à jour.

Les opérations multiples sont plus lentes que les opérations uniques, mais restent plus efficaces pour traiter de grandes quantités de données.

L'indexation améliore les performances des opérations de lecture et de mise à jour, mais a un impact négligeable sur l'insertion et la suppression.

Les performances des opérations CRUD sont plus stables avec l'indexation, ce qui réduit la variance des temps d'exécution.

Enfin, les performances de MySQL Cluster sont comparables à celles de MongoDB en mode standalone.

Chapitre 6

Analyse des Résultats

I

Performances des Requêtes

- MongoDB excelle pour les lectures indexées et les mises à jour fréquentes de la structure des données.
- MySQL est plus performant pour les requêtes transactionnelles complexes et les lectures indexées.
- L'indexation est un facteur clé pour améliorer les performances des requêtes.
- Les performances de MongoDB augmentent grandement avec l'indexation.
- Paradoxalement, l'indexation n'améliore pas significativement les performances de MySQL.
- Les opérations multiples sont plus efficaces pour traiter de grandes quantités de données.
- L'indexation permet de stabiliser et d'améliorer le temps d'exécution des opérations CRUD.

II

Impact de la RéPLICATION et du Sharding

- La réPLICATION et le sharding améliorent la tolérance aux pannes et la scalabilité.
- La réPLICATION augmente la latence d'écriture, mais garantit une haute disponibilité des données.
- Le sharding améliore les performances des lectures et écritures parallèles par rapport à la réPLICATION.
- MySQL Cluster offre des fonctionnalités similaires à MongoDB, mais avec une configuration plus complexe.
- MongoDB est plus adapté pour les applications nécessitant une scalabilité horizontale et une haute disponibilité.

III

Critique des résultats

- Les tests ont été réalisés sur des configurations de base, sans optimisations spécifiques.
- Les performances peuvent varier en fonction de la charge, de la volumétrie des données, et de la configuration matérielle.
- Les résultats sont indicatifs et doivent être confirmés par des tests plus approfondis.
- Les performances des SGBD dépendent fortement de la nature des requêtes et de la structure des données.
- Les tests ont été réalisés sur des scénarios simples, mais les performances peuvent varier dans des cas d'utilisation réels.

IV

Comparaisons et Bilan

CHAPITRE 6. ANALYSE DES RÉSULTATS

Chapitre 7

Conclusion

Cette étude met en évidence les avantages et inconvénients des architectures NoSQL et relationnelles dans divers scénarios.

MongoDB offre une grande flexibilité et scalabilité, toutefois, la réPLICATION et le sharding peuvent impacter les performances des opérations CRUD. Néanmoins, le sharding, s'il est possible est une très bonne solution.

Avec une meilleure configuration, on pourrait s'attendre à des performances similaire au mode standalone et on a tout de même obtenu des meilleures performances que dans le mode répliqué.

Toutefois MySQL, reste performant pour des cas nécessitant des transactions complexes et ne peut montrer l'étendue de ses performances dans un cas aussi simple ne nécessitant pas de modèle relationnel.

De plus, les performances du cluster MySQL s'est avéré, contre toute attente, plus performant que MySQL Standalone, et aussi performante que MongoDB Standalone.

Annexe A

Annexe

I

Déploiement du Code

Le projet nécessite l'installation de :

- python3
- pip3
- docker
- docker compose

Ces outils permettent d'installer les dépendances des codes python et de réaliser les tests de performance de MongoDB et MySQL. Afin de reproduire les tests de performance, il faut d'abord cloner le dépôt git suivant :

```
git clone https://github.com/BJCode-git/Projet-TDLE.git -b main &&
cd Projet-TDLE
```

Il faut ensuite générer les données de test en utilisant les commandes suivantes :

```
pip3 install -r requirements/data_generation-requirements.txt &&
python3 data\generation.py
```

On déploie alors des conteneurs docker pour réaliser des tests de performance sur une base de données mongodb standalone, une base de données mongodb répliquée et une base de données mongodb fragmentée. Pour MySQL, on réalise des tests de performance sur une base de données standalone et une base de données fragmentée.

Il est possible d'utiliser le script **start.sh** pour déployer automatiquement les conteneurs, lancer tous les test puis arrêter les conteneurs. Cette approche permet de réaliser les tests de performance de manière automatique et permet de consommer moins de ressources mémoire et CPU.

```
chmod +x start.sh &&
./start.sh
```

Le script **start.sh** réalise basiquement les opérations suivantes : - Nettoyage de l'environnement - Crée les dossiers logs et plots pour les logs et les graphiques - Installation des dépendances python - Génération des données de test - Test de performance de MongoDB en standalone - Test de performance de MongoDB en réPLICATION - Test de performance de MongoDB en sharding - Test de performance de MySQL en standalone - Test de performance de MySQL en sharding

Installation des dépendances python :

```
pip3 install -r requirements/generate_data-requirements.txt
pip3 install -r requirements/mongo-requirements.txt
pip3 install -r requirements/mysql-requirements.txt
```

Génération des données :

```
python3 generate_data.py &&
docker compose down -v --remove-orphans
```

I.1

Tests de performance de MongoDB

Test en standalone :

```
## Démarrage et attente de la disponibilité du serveur
docker compose up mongo-standalone -d --wait &&
python3 mongodb.py --standalone &&
docker compose down -v --remove-orphans
```

Test avec replica :

```
docker compose up mongo-replica-initiate &&
python3 mongodb.py --replica           &&
docker compose down -v --remove-orphans
```

Test avec sharding :

```
docker compose up mongo-sharded-cluster &&
python3 mongodb.py --sharded           &&
docker compose down -v --remove-orphans
```

I.2

Tests de performance de MySQL

Test en standalone :

```
# Démarrage et attente de la disponibilité du serveur
docker compose up --wait -d mysql-standalone &&
python3 mysql.py --standalone           &&
docker compose down -v --remove-orphans
```

Test avec sharding :

```
docker compose up mysql-sharded-initiate &&
python3 mysql.py --sharded           &&
docker compose down -v --remove-orphans
```

Il est également possible de séparer les test de performance de MongoDB et MySQL. Après génération des données, on peut lancer toutes les infrastructures MongoDB et lancer les tests de performance de MongoDB en utilisant les commandes suivantes :

```
docker compose up start_mongo &&
python3 mongodb.py
```

On peut ensuite stopper les conteneurs docker en utilisant la commande suivante :

ANNEXE A. ANNEXE

```
docker compose down -v --remove-orphans
```

On peut en faire de même avec MySQL :

```
docker compose up start_mysql &&  
python3 mysql.py
```

Finalement, Pour nettoyer les données générées, on peut utiliser le script **clear.sh** :

```
./clear.sh
```

Note : Il est possible de configurer les paramètres des tests et le déploiement en modifiant les variables disponibles dans le fichier .env