Université sorbonne Pari Nord

IUT de Villetaneuse

J-B Clément - Université, 93430 Villetaneuse

Étude Comparative de Redis et MongoDB pour l'Optimisation des Données

Réalisé par : Hiba OUMSID

Enseignant : Jean-Christophe Cerin

Sommaire

1. Contexte et Problématique

Ce rapport est rédigé pour le compte de HTL, une entreprise spécialisée dans le développement de solutions de gestion de données pour les services de transport aérien. Dans le cadre de l'évolution de sa plateforme, HTL souhaite optimiser la gestion des données critiques, telles que les informations sur les vols, les profils des pilotes, les réservations, ainsi que les détails concernant les villes de départ et d'arrivée.

Cette optimisation est essentielle pour garantir des performances élevées, un accès rapide aux données, et une gestion efficace des informations pour ses utilisateurs.

Pour répondre à ces besoins, deux technologies de bases de données NoSQL sont envisagées : Redis et MongoDB. Redis, réputé pour sa rapidité d'accès en mémoire, pourrait être utilisé pour les fonctionnalités nécessitant un accès instantané, comme la mise à jour en temps réel des statuts de vols ou des informations de réservation. MongoDB, quant à lui, se distingue par sa flexibilité dans la gestion de données complexes et structurées, ce qui la rend idéale pour la gestion des profils de pilotes, des détails de vols, et des réservations.

L'objectif de ce rapport est de comparer les capacités de Redis et MongoDB afin de déterminer laquelle de ces solutions répond le mieux aux besoins spécifiques de la plateforme HTL. En nous basant sur des critères tels que la rapidité d'accès, la gestion de la charge et l'aptitude de chaque technologie à traiter les données critiques du secteur aérien, cette étude inclura une série de tests sur la lecture, l'insertion, la jointure et la mise à jour des données pour évaluer leurs performances respectives dans ces domaines.

Les conclusions de ce rapport fourniront à HTL une recommandation claire sur la technologie à privilégier et offriront des pistes d'optimisation pour répondre aux exigences futures de la plateforme

2. Matériel et Méthodes

2.1 Environnement de Développement

- Redis : Pour mener à bien cette étude, nous avons utilisé WSL (Windows Subsystem for Linux) sur notre ordinateur Windows afin d'installer et de configurer Redis. Pour ce faire, les commandes suivantes ont été exécutées dans le terminal WSL :

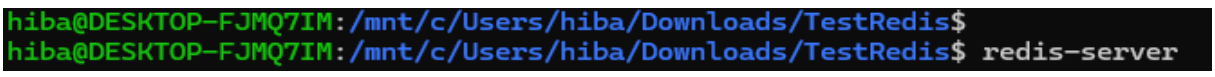


Figure 1 : CMD pour démarrer redis

- MongoDB Atlas : Pour travailler avec MongoDB, nous avons utilisé MongoDB Atlas, un service de base de données cloud. La création d'un compte implique les étapes suivantes :

- Inscription sur le site de MongoDB Atlas.
- Création d'un nouveau cluster, nécessitant la sélection d'un fournisseur cloud, d'une région et d'un type de cluster.

- Configuration de l'accès réseau en ajoutant l'adresse IP de la machine locale ou en utilisant 0.0.0.0/0 pour permettre l'accès depuis n'importe quelle adresse IP.
- Copie de la chaîne de connexion (connection string), qui contient l'ID du cluster et les informations nécessaires pour se connecter à la base de données à partir de l'application Python. Exemple : mongodb+srv://hibaoumsid:<motDePasse>@hiba.o9sls.mongodb.net/?retryWrites=true&w=majority&appName=hiba.
- **N.B.** : Il n'est pas nécessaire d'activer manuellement le serveur pour MongoDB Atlas, car le serveur est déjà en ligne.

2.2 Langage et Bibliothèques

Concernant le langage, nous avons utilisé Python pour développer les scripts d'importation et de test de performance, avec les bibliothèques nécessaires installées via la commande :
pip install <library_name>

2.3 Outils Communs

- Les bibliothèques utilisées dans les deux technologies incluent :
- os : Interagit avec le système d'exploitation pour gérer les fichiers.
- time : Mesure le temps d'exécution des opérations.
- memory_profiler : Évalue l'utilisation de la mémoire par le code.
- json : Manipule et convertit les données en format JSON pour faciliter leur importa-tion.

2.4 Outils Spécifiques à Redis

Pour Redis, les bibliothèques spécifiques incluent :

- Redis : Permet de se connecter à la base de données et d'effectuer des opérations sur les données.

2.5 Outils Spécifiques à MongoDB

Pour MongoDB, les bibliothèques spécifiques incluent :

- MongoClient : Permet de se connecter à MongoDB et d'effectuer des opérations sur les données.

2.6 Les méthodes

a. Dénormalisation des Données

Pour simuler un scénario de chargement de données, plusieurs fichiers texte contenant des informations sur les pilotes, les réservations et les avions ont été utilisés. Ces données ont ensuite été dénormalisées et assemblées sous la forme d'un unique fichier JSON, garantissant que les informations soient correctement formatées pour l'importation dans les bases de données.

- **jointure_tests.py :**
  - **jointure_sur_attribut(data1, data2, attribut, valeur_cible=None) :** Teste les jointures sur un attribut donné et retourne une liste des jointures trouvées.
  - **sauvegarder_jointures(jointures, nom_fichier) :** Sauvegarde les résultats de la jointure dans un fichier JSON.
  - **compter_elements_json(nom_fichier) :** Compte le nombre d'éléments dans le fichier JSON.
  - **taille_fichier_json(nom_fichier)** : Obtient la taille du fichier JSON.

Comme l'explique la figure 2, de la ligne 48 à la ligne 68, le code réalise les opérations suivantes :

Il charge les données à partir du fichier JSON data/data.json, puis les divise en deux dictionnaires distincts, data_part1 et data_part2.

Ensuite, une jointure est effectuée sur ces deux ensembles de données en utilisant l'attribut "VilleD", afin de sélectionner les entrées correspondant à la valeur "Marseille".

Si des correspondances sont trouvées, elles sont sauvegardées dans un nouveau fichier JSON nommé data/jointure_result.json, après quoi le code compte le nombre d'éléments dans ce fichier et calcule sa taille.

```
47  # Exemple d'exécution
48  if __name__ == "__main__":
49      # Charger les données depuis le fichier data.json
50      with open("data/data.json", 'r') as f:
51          data = json.load(f)
52
53      # Diviser les données pour le test
54      items = list(data.items())
55      data_part1 = dict(items[:len(items) // 2])
56      data_part2 = dict(items[len(items) // 2:])
57
58      # Effectuer la jointure sur l'attribut "VilleD"
59      jointures = jointure_sur_attribut(data_part1, data_part2, attribut="VilleD", valeur_cible="Marseille")
60
61      # Sauvegarder le résultat de la jointure dans un nouveau fichier JSON si des jointures ont été trouvées
62      if jointures:
63          nom_fichier_jointure = "data/jointure_result.json"
64          sauvegarder_jointures(jointures, nom_fichier_jointure)
65
66          # Compter le nombre d'éléments et obtenir la taille du fichier JSON
67          compter_elements_json(nom_fichier_jointure)
68          taille_fichier_json(nom_fichier_jointure)
69
```

Figure 2 : jointure_tests.py

- **json_construction.py :**
  - **construire_json_final(dir_path) :** Construit un fichier JSON à partir de plusieurs fichiers texte en utilisant une correspondance de colonnes définie dans un dictionnaire, et intègre les données des avions, pilotes et réservations.
  - **sauvegarder_json(json_data, file_path) :** Sauvegarde les données JSON dans un fichier spécifié.

b. Insertion des Données

. Insertion des données dans Redis

Plusieurs fichiers Python ont été créés pour structurer le code et organiser les différentes fonctionnalités liées à Redis :

- **redis_insertion.py :**
  - **inserer_dans_redis(file_path) :** Charge les données JSON depuis un fichier spécifié et les insère dans Redis, en utilisant une clé formatée pour chaque vol (par exemple, vol:V101). La fonction s'assure de ne stocker que les vols à partir de V101.

Comme illustré dans la figure 3, de la L1 à L20 : ce code connecte Redis, charge les données depuis data/data.json, et stocke en base Redis uniquement les vols numérotés à partir de V101, chaque vol étant sauvegardé sous une clé unique et affichant un message de confirmation.

scripts > redis_insertion.py > ...

```python
import json
import redis

# Connexion à Redis
r = redis.StrictRedis(host='localhost', port=6379, db=0, decode_responses=True)

# Charger et insérer les données dans Redis
def inserer_dans_redis(file_path):
    with open(file_path, 'r') as f:
        json_data = json.load(f)

    for vol in json_data:
        if int(vol[1:]) >= 101:  # Condition pour les vols à partir de V101
            key = f"vol:{vol}"
            r.set(key, json.dumps(json_data[vol]))  # Stocker chaque vol dans Redis
            print(f"Vol {vol} stocké dans Redis sous la clé {key}")

# Exécution principale pour insérer les données dans Redis
if __name__ == "__main__":
    inserer_dans_redis("data/data.json")

```

Figure 3 : redis_insertion.py

- **performance-tests.py :**
  - **test_construction_json() :** Mesure la performance de la construction du JSON et de son stockage dans un fichier.

Comme l'explique la figure 4 de la L9 à L23, le code mesure la performance de la construction d'un fichier JSON en suivant le temps écoulé et la mémoire utilisée avant et après l'opération, puis affiche ces informations pour évaluer l'efficacité.

```python
# Mesurer la performance de la construction JSON
def test_construction_json():
    start_time = time.time()
    mem_before = memory_profiler.memory_usage()[0]

    # Construire le JSON
    json_data = json_construction.construire_json_final("data/")
    json_construction.sauvegarder_json(json_data, "data/data.json")

    end_time = time.time()
    mem_after = memory_profiler.memory_usage()[0]

    print(f"Construction JSON terminée en {end_time - start_time:.2f} secondes")
    print(f"Utilisation de la mémoire : {mem_after - mem_before:.2f} MiB")
```

Figure 4 : performance-tests.py

test_jointure() : Mesure la performance de la jointure entre deux ensembles de données à partir du fichier JSON.

- **test_insertion_redis()** : Mesure la performance de l'insertion des données dans Redis.

- **updateTest.py :**

Ce fichier évalue la rapidité des mises à jour dans Redis. Une connexion à Redis est établie pour permettre l'insertion et la mise à jour des données. Un ensemble de données de vol est inséré dans Redis, puis plusieurs mises à jour du statut du vol sont effectuées. Le temps nécessaire pour réaliser ces mises à jour est mesuré, ce qui permet d'évaluer la performance de Redis dans ce contexte.

Comme l'explique la figure 5, de la ligne 1 à la ligne 19, le code réalise les actions suivantes :

1. Il importe les modules nécessaires (time et redis) et se connecte à une instance Redis locale.
2. Une entrée de vol (vol:AF123) est insérée dans Redis avec des informations de départ, d'arrivée et de statut initial.
3. Le code mesure le temps nécessaire pour mettre à jour le statut de vol trois fois dans Redis, en changeant le statut à chaque itération (par exemple, "en retard", "embarquement", "annulé") et en lisant les données après chaque mise à jour. Enfin, il affiche le temps total pris pour ces mises à jour.

scripts > updateTest.py > ...

```python
import time
import redis

# Connexion à Redis
redis_conn = redis.StrictRedis(host='localhost', port=6379, db=0)

# Données de vol pour le test
vol_data = {"vol_id": "AF123", "depart": "Paris", "arrivee": "Londres", "statut": "à l'heure"}

# Insertion initiale des données dans Redis
redis_conn.hmset(f"vol:{vol_data['vol_id']}", vol_data)

# Mesure de la vitesse de mise à jour pour Redis
start_time = time.time()
for statut in ["en retard", "embarquement", "annulé"]:
    redis_conn.hset(f"vol:{vol_data['vol_id']}", "statut", statut)
    # Lecture des données après chaque mise à jour
    redis_conn.hgetall(f"vol:{vol_data['vol_id']}")
print("Redis Update time:", time.time() - start_time)

```

Figure 5 : updateTest.py

- Insertion des données dans MongoDB

Voici la présentation des fichiers Python avec les fonctions de MongoDB :

- **mongodb_insertion.py :**

- **inserer_json_mongo(fichier_json, nom_collection) :** Charge les données JSON depuis un fichier spécifié et les insère dans MongoDB. La fonction gère à la fois les listes d'objets JSON et les objets JSON uniques.

- **supprimer_donnees_mongo(nom_collection) :** Supprime tous les documents d'une collection spécifiée. Comme l'illustre la figure 6 de la L45 à L48

```
45  def supprimer_donnees_mongo(nom_collection):
46      collection = db[nom_collection]
47      result = collection.delete_many({})
48      print(f"{result.deleted_count} documents supprimés de la collection '{nom_collection}'.")
49
50  supprimer_donnees_mongo("lesVols")
51
```

Figure 6: mongodb_insertion.py

- **updateTest.py :**

  - Ce fichier évalue la rapidité des mises à jour dans MongoDB. Une connexion à MongoDB est établie pour permettre l'insertion et la mise à jour des données. Un ensemble de données de vol est inséré dans MongoDB, puis plusieurs mises à jour du statut du vol sont effectuées. Le temps nécessaire pour réaliser ces mises à jour est mesuré, ce qui permet d'évaluer la performance de MongoDB dans ce contexte.

Comme l'explique la figure 7, de la ligne 1 à la ligne 22, le code mesure le temps nécessaire pour mettre à jour plusieurs fois le statut d'un vol dans MongoDB, après l'avoir inséré ou mis à jour initialement, puis affiche le temps total pris pour ces mises à jour.

```
scripts > updateTest.py > ...
1   import time
2   from pymongo import MongoClient
3
4   # Connexion à MongoDB
5   mongo_client = MongoClient("mongodb+srv://hibaoumsid:hiba@hiba.o9sls.mongodb.net/?retryWrites=true&w=majority&ap
6   mongo_db = mongo_client["TestMongo"]
7   mongo_collection = mongo_db["lesVols"]
8
9   # Données de vol pour le test
10  vol_data = {"vol_id": "AF123", "depart": "Paris", "arrivee": "Londres", "statut": "à l'heure"}
11
12  # Insertion initiale des données dans MongoDB
13  mongo_collection.update_one({"vol_id": vol_data["vol_id"]}, {"$set": vol_data}, upsert=True)
14
15  # Mesure de la vitesse de mise à jour pour MongoDB
16  start_time = time.time()
17  for statut in ["en retard", "embarquement", "annulé"]:
18      mongo_collection.update_one({"vol_id": vol_data["vol_id"]}, {"$set": {"statut": statut}})
19      # Lecture des données après chaque mise à jour
20      mongo_collection.find_one({"vol_id": vol_data["vol_id"]})
21  print("MongoDB Update time:", time.time() - start_time)
22
```

Figure 7 : updateTest.py

- **testlecture.py :**

  - **lire_donnees_mongo(nom_collection) :** Se connecte à MongoDB, puis lit et affiche tous les documents d'une collection donnée. Cette fonction mesure également le temps total de lecture des données et le nombre de documents lus, offrant une évaluation de la performance de la lecture des données.

Comme l'explique la figure 8, de la ligne 1 à la ligne 29, le code se connecte à une base de données MongoDB distante, puis récupère toutes les données d'une collection spécifique. Il mesure le temps de lecture de la collection et affiche le nombre de documents ainsi que le contenu de chaque document, tout en indiquant le temps nécessaire pour cette opération.

```python
import time
from pymongo import MongoClient

# Connexion à MongoDB
client = MongoClient("mongodb+srv://hibaoumsid:hiba@hiba.o9sls.mongodb.net/?retryWrites=true&w=majority&appName
db = client['TestMongo']  # Remplace par le nom de ta base de données

def lire_donnees_mongo(nom_collection):
    collection = db[nom_collection]

    # Mesurer le temps de lecture
    start_time = time.time()

    # Récupérer toutes les données de la collection
    documents = list(collection.find())

    end_time = time.time()

    # Afficher le nombre de documents
    print(f"Nombre de documents dans la collection '{nom_collection}': {len(documents)}")
    print(f"Temps de lecture des données : {end_time - start_time:.4f} secondes.")

    # Affichage des données
    for doc in documents:
        print(doc)

# Exécution principale (si nécessaire)
if __name__ == "__main__":
    lire_donnees_mongo("lesVols")
```

Figure 8 : testlecture.py

3. Résultats

Dans cette section, nous présentons les résultats des preuves de concept effectuées avec Redis et MongoDB, en mettant l'accent sur les performances de stockage et de récupération des données, ainsi que sur l'impact de la dénormalisation

3.1 Performance de la Dénormalisation des Données

La figure 9 représente un extrait représentatif du fichier JSON généré, illustrant la structure et la disposition des données après dénormalisation.


```
        "V141": {
            "VilleD": "Amsterdam",
            "VilleA": "Marseille",
            "DateD": "1/04/07",
            "HD time": "23:20",
            "DateA": "2/04/07",
            "HA time": "1:10",
            "NumPil": "5050",
            "NumAv": "101",
            "avion": {
                "NomAv": "Boeing 707",
                "CapAv": "350",
                "VilleAv": "Beyrouth"
            },
            "pilote": {
                "NomPil": "Leblanc",
                "NaisPil": "1960",
                "VillePil": "Marseille"
            },
            "reservations": {
                "1008": {
                    "NumVol": "V141",
                    "Classe": "Touriste",
                    "NbPlaces": "2"
                },
                "1011": {
                    "NumVol": "V141",
                    "Classe": "Economique",
                    "NbPlaces": "8"
                }
            }
        },
        "V142": {
            "VilleD": "Amsterdam",
```


Figure 9 : fichier data.json

3.2 Test d'Insertion des Données

- **Redis** : Insertion terminée en **0.23 secondes** avec une utilisation de mémoire de **0.36 MiB**.

```
Vol V927 stocké dans Redis sous la clé vol:V927
Vol V928 stocké dans Redis sous la clé vol:V928
Vol V929 stocké dans Redis sous la clé vol:V929
Insertion Redis terminée en 0.23 secondes
Utilisation de la mémoire : 0.36 MiB
```

Figure 10 : Résultat de redis_insertion.py

- **MongoDB** : Temps d'insertion de **0.0627 secondes** avec une utilisation de mémoire de 39.2 MiB comme le montre la figure 11

```
hiba@DESKTOP-FJMQ7IM:/mnt/c/Users/hiba/Downloads/TestMongo$ /usr/bin/python /mnt/c/Users/hiba/Downlo
ads/TestMongo/scripts/mongodb_insertion.py
2 documents supprimés de la collection 'lesVols'.
Données insérées avec succès dans la collection lesVols.
Temps d'insertion : 0.0627 secondes.
Filename: /mnt/c/Users/hiba/Downloads/TestMongo/scripts/mongodb_insertion.py

Line #    Mem usage    Increment  Occurrences   Line Contents
=============================================================
    10     39.2 MiB     39.2 MiB           1   @profile
    11                                         def inserer_json_mongo(fichier_json, nom_collection):
    12                                             # Charger les données depuis le fichier JSON
    13     39.2 MiB      0.0 MiB           1       try:
```

Figure 11 : Résultat de MongoDB Test d'insertion.py

3.3 Test de Lecture des Données

- **MongoDB** : Temps de lecture des données de **0.3310 secondes** comme le montre la figure 12

```
hiba@DESKTOP-FJMQ7IM:/mnt/c/Users/hiba/Downloads/TestMongo$ /usr/bin/python /mnt/c/Users/hiba/Downlo
ads/TestMongo/scripts/testlecture.py
Nombre de documents dans la collection 'lesVols': 1
Temps de lecture des données : 0.3310 secondes.
```

Figure 12 : Résultat de MongoDB Test de lecture.py

- **Redis** : Temps de lecture des données de **0.0543 secondes** comme le montre la figure 13

```
hiba@DESKTOP-FJMQ7IM:/mnt/c/Users/hiba/Downloads/TestRedis$ /usr/bin/python /mnt/c/Users/hiba/
Downloads/TestRedis/scripts/lecture.py
Clé 'vol:AF123' est de type 'hash', donc ignorée.
Nombre de clés lues : 200
Temps de lecture des données dans Redis : 0.0543 secondes.
```

Figure 13 : Résultat de Redis Test de lecture.py

3.4 Test de Mise à Jour des Données

- **Redis** : Temps de mise à jour : **0.00195 secondes** comme le montre la figure 14.

```
hiba@DESKTOP-FJMQ7IM:/mnt/c/Users/hiba/Downloads/TestRedis$ /usr/bin/python /mnt/c/Users/hiba/
Downloads/TestRedis/scripts/updateTest.py
/mnt/c/Users/hiba/Downloads/TestRedis/scripts/updateTest.py:11: DeprecationWarning: Redis.hmse
t() is deprecated. Use Redis.hset() instead.
  redis_conn.hmset(f"vol:{vol_data['vol_id']}", vol_data)
Redis Update time: 0.001955747604370117
```

Figure 14 : Résultat de Redis Test mis-a-jour.py

- **MongoDB** : Temps de mise à jour : **0.086 secondes** comme le montre la figure 15

```
hiba@DESKTOP-FJMQ7IM:/mnt/c/Users/hiba/Downloads/TestMongo$ /usr/bin/python /mnt/c/Users/hiba/Downlo
ads/TestMongo/scripts/updateTest.py
MongoDB Update time: 0.08698368072509766
hiba@DESKTOP-FJMQ7IM:/mnt/c/Users/hiba/Downloads/TestMongo$
```

Figure 15 : Résultat de MongoDB Test mis-a-jour.py

3.5 Test de la construction du Json

Le processus de construction du JSON a été achevé en **0.12 secondes**, avec une utilisation de la mémoire négligeable de **0.00 MiB** comme le montre la figure 16

```
Lecture du fichier : AVIONS.txt
Lecture du fichier : CLIENTS.txt
Lecture du fichier : DEFCLASSES.txt
Lecture du fichier : PILOTES.txt
Lecture du fichier : Plain Text.txt
Lecture du fichier : RESERVATIONS.txt
Lecture du fichier : VOLS.txt
Données sauvegardées avec succès dans data/data.json
Construction JSON terminée en 0.12 secondes
Utilisation de la mémoire : 0.00 MiB
hiba@DESKTOP-FJMQ7IM:/mnt/c/Users/hiba/Downloads/TestRedis$
```

Figure 16 : Résultat de performance_test.py

3.6 Test de la performance de la jointure

La jointure entre les ensembles de données a également été effectuée en **0.12 secondes**, avec une utilisation de la mémoire légèrement plus élevée, à **0.32 MiB**. Comme le montre la figure 17

```
Jointure trouvée entre V523 et V690 sur VilleD avec la valeur 'Marseille'
Jointure trouvée entre V523 et V691 sur VilleD avec la valeur 'Marseille'
Jointure trouvée entre V523 et V692 sur VilleD avec la valeur 'Marseille'
Jointure trouvée entre V523 et V693 sur VilleD avec la valeur 'Marseille'
Jointure trouvée entre V523 et V694 sur VilleD avec la valeur 'Marseille'
Jointure terminée en 0.16 secondes
Utilisation de la mémoire : 0.32 MiB
```

Figure 17 : Résultat de de jointure_tests.py

4. Discussion

4.1 Performance de la Construction et de la Jointure des Données

Le processus de construction du JSON s'est révélé très rapide, avec un temps de 0.12 secondes et une utilisation de mémoire quasi nulle (0.00 MiB). Cette faible utilisation de mémoire montre l'efficacité de cette étape, en particulier lors de la préparation initiale des données pour une base de données.

Pour le test de jointure, bien que la durée d'exécution reste identique à celle de la construction (0.12 secondes), l'utilisation de la mémoire a légèrement augmenté à 0.32 MiB. Cette augmentation s'explique par la complexité supplémentaire de manipuler plusieurs ensembles de données et d'établir des correspondances entre eux. Bien que ces opérations de jointure soient plus exigeantes, la consommation de mémoire reste faible, indiquant une bonne gestion des ressources pour la préparation des données.

4.2 Comparaison des Performances d'Insertion

Les résultats d'insertion montrent des écarts marqués entre Redis et MongoDB. Redis réalise l'insertion en 0.23 secondes avec une utilisation de mémoire de 0.36 MiB, tandis que MongoDB est plus rapide avec un temps d'insertion de 0.0627 secondes, mais consomme significativement plus de mémoire (39.2 MiB). Cette différence peut s'expliquer par la nature des deux bases de données : Redis, en tant que base de données en mémoire, est conçu pour gérer efficacement de petites quantités de données rapidement accessibles, tandis que MongoDB, une base de données NoSQL orientée document, gère des structures de données plus lourdes, ce qui peut justifier la consommation mémoire plus élevée.

4.3 Comparaison des Performances de Lecture

En matière de lecture, Redis se démarque nettement avec un temps de 0.0543 secondes pour lire 200 clés, contre 0.3310 secondes pour MongoDB. Cela confirme que Redis, conçu pour des accès rapides en mémoire, excelle dans les opérations de lecture par rapport à MongoDB, qui effectue des lectures depuis le disque ou à partir de caches partiels. Ces résultats renforcent le choix de Redis pour les applications nécessitant des accès très rapides à des données en mémoire.

4.4 Comparaison des Performances de Mise à Jour

Les tests de mise à jour montrent également un contraste entre Redis et MongoDB. Redis effectue les mises à jour en 0.00195 secondes, tandis que MongoDB prend 0.086 secondes. Cette différence est cohérente avec les caractéristiques des deux technologies : Redis met en cache les données en mémoire, ce qui permet des mises à jour instantanées.

MongoDB, bien que performant, est limité par le processus de mise à jour des documents stockés sur disque ou dans des caches, ce qui explique le temps légèrement plus élevé.

4.5 Avantages et Inconvénients de MongoDB et Redis

Ce tableau résume les principaux avantages et inconvénients de MongoDB et Redis de manière succincte.

Critères	MongoDB	Redis
Avantages	- Structure flexible (JSON) - Support des requêtes complexes	- Accès ultra-rapide - Richesse des structures de données
Inconvénients	- Temps d'accès plus long - Consommation mémoire élevée	- Données non persistantes par défaut - Limité pour les requêtes complexes, Limité pour les requêtes complexes

5. Conclusion

Pour répondre aux besoins de la plateforme, deux options sont envisageables : MongoDB seul, ou une solution hybride associant MongoDB et Redis. MongoDB, utilisé en tant que solution unique, permet de stocker et gérer toutes les données critiques de manière structurée et évolutive. Il offre une performance satisfaisante pour les opérations d'insertion et de lecture, tout en garantissant un stockage durable et une flexibilité optimale, notamment grâce à MongoDB Atlas, qui assure une infrastructure cloud fiable et sécurisée.

La solution hybride, quant à elle, combine MongoDB pour le stockage des données durables avec Redis comme cache en mémoire pour les données consultées fréquemment. Cette approche maximise la performance de la plateforme en permettant un accès instantané aux informations critiques sans compromettre la durabilité des données. En résumé, MongoDB seul conviendra si la simplicité de gestion est privilégiée. Cependant, si la réactivité est primordiale pour le client, la solution hybride MongoDB-Redis est recommandée pour garantir des temps de réponse optimisés lors des opérations fréquentes.

6. Annexes

Figure 1 : CMD pour démarrer redis

```
hiba@DESKTOP-FJMQ7IM:/mnt/c/Users/hiba/Downloads/TestRedis$
hiba@DESKTOP-FJMQ7IM:/mnt/c/Users/hiba/Downloads/TestRedis$ redis-server
```

Figure 2 : jointure_tes

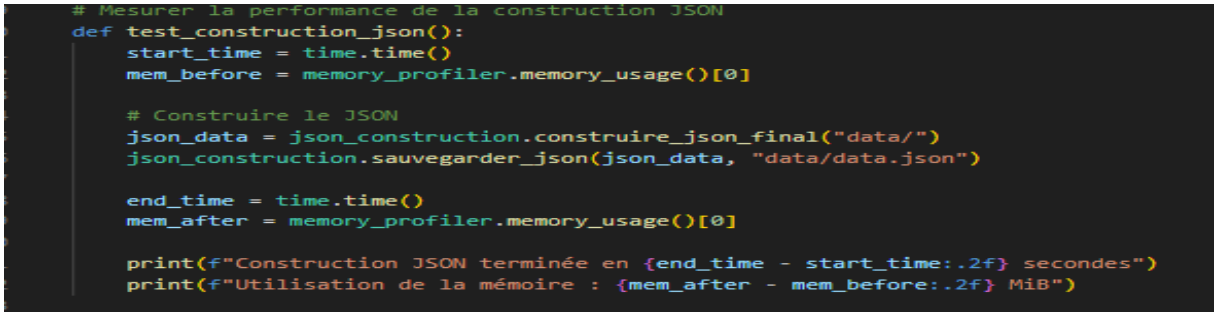
```
# Exemple d'exécution
if __name__ == "__main__":
    # Charger les données depuis le fichier data.json
    with open("data/data.json", 'r') as f:
        data = json.load(f)

    # Diviser les données pour le test
    items = list(data.items())
    data_part1 = dict(items[:len(items) // 2])
    data_part2 = dict(items[len(items) // 2:])

    # Effectuer la jointure sur l'attribut "VilleD"
    jointures = jointure_sur_attribut(data_part1, data_part2, attribut="VilleD", valeur_cible="Marseille")

    # Sauvegarder le résultat de la jointure dans un nouveau fichier JSON si des jointures ont été trouvées
    if jointures:
        nom_fichier_jointure = "data/jointure_result.json"
        sauvegarder_jointures(jointures, nom_fichier_jointure)

        # Compter le nombre d'éléments et obtenir la taille du fichier JSON
        compter_elements_json(nom_fichier_jointure)
        taille_fichier_json(nom_fichier_jointure)
```

Figure 3 : redis_insertion.py

scripts > redis_insertion.py > ...

```
import json
import redis

# Connexion à Redis
r = redis.StrictRedis(host='localhost', port=6379, db=0, decode_responses=True)

# Charger et insérer les données dans Redis
def inserer_dans_redis(file_path):
    with open(file_path, 'r') as f:
        json_data = json.load(f)

    for vol in json_data:
        if int(vol[1:]) >= 101:  # Condition pour les vols à partir de V101
            key = f"vol:{vol}"
            r.set(key, json.dumps(json_data[vol]))  # Stocker chaque vol dans Redis
            print(f"Vol {vol} stocké dans Redis sous la clé {key}")

# Exécution principale pour insérer les données dans Redis
if __name__ == "__main__":
    inserer_dans_redis("data/data.json")
```

Figure 4: performance-tests.py



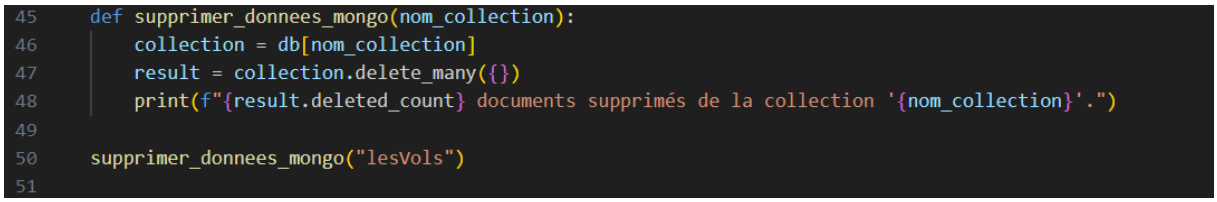
```python
# Mesurer la performance de la construction JSON
def test_construction_json():
    start_time = time.time()
    mem_before = memory_profiler.memory_usage()[0]

    # Construire le JSON
    json_data = json_construction.construire_json_final("data/")
    json_construction.sauvegarder_json(json_data, "data/data.json")

    end_time = time.time()
    mem_after = memory_profiler.memory_usage()[0]

    print(f"Construction JSON terminée en {end_time - start_time:.2f} secondes")
    print(f"Utilisation de la mémoire : {mem_after - mem_before:.2f} MiB")
```

Figure 5: updateTest.py

scripts > updateTest.py > ...

```python
import time
import redis

# Connexion à Redis
redis_conn = redis.StrictRedis(host='localhost', port=6379, db=0)

# Données de vol pour le test
vol_data = {"vol_id": "AF123", "depart": "Paris", "arrivee": "Londres", "statut": "à l'heure"}

# Insertion initiale des données dans Redis
redis_conn.hmset(f"vol:{vol_data['vol_id']}", vol_data)

# Mesure de la vitesse de mise à jour pour Redis
start_time = time.time()
for statut in ["en retard", "embarquement", "annulé"]:
    redis_conn.hset(f"vol:{vol_data['vol_id']}", "statut", statut)
    # Lecture des données après chaque mise à jour
    redis_conn.hgetall(f"vol:{vol_data['vol_id']}")
print("Redis Update time:", time.time() - start_time)

```

Figure 6 : mongodb_insertion.py



```python
def supprimer_donnees_mongo(nom_collection):
    collection = db[nom_collection]
    result = collection.delete_many({})
    print(f"{result.deleted_count} documents supprimés de la collection '{nom_collection}'.")

supprimer_donnees_mongo("lesVols")

```

Figure 7 : updateTest.py

```python
import time
from pymongo import MongoClient

# Connexion à MongoDB
mongo_client = MongoClient("mongodb+srv://hibaoumsid:hiba@hiba.o9sls.mongodb.net/?retryWrites=true&w=majority&a
mongo_db = mongo_client["TestMongo"]
mongo_collection = mongo_db["lesVols"]

# Données de vol pour le test
vol_data = {"vol_id": "AF123", "depart": "Paris", "arrivee": "Londres", "statut": "à l'heure"}

# Insertion initiale des données dans MongoDB
mongo_collection.update_one({"vol_id": vol_data["vol_id"]}, {"$set": vol_data}, upsert=True)

# Mesure de la vitesse de mise à jour pour MongoDB
start_time = time.time()
for statut in ["en retard", "embarquement", "annulé"]:
    mongo_collection.update_one({"vol_id": vol_data["vol_id"]}, {"$set": {"statut": statut}})
    # Lecture des données après chaque mise à jour
    mongo_collection.find_one({"vol_id": vol_data["vol_id"]})
print("MongoDB Update time:", time.time() - start_time)
```

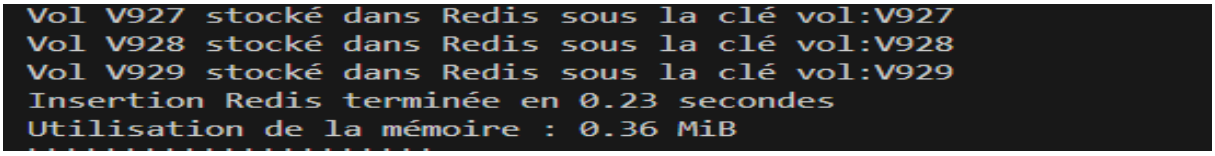
Figure 8 : testlecture.py

```python
import time
from pymongo import MongoClient

# Connexion à MongoDB
client = MongoClient("mongodb+srv://hibaoumsid:hiba@hiba.o9sls.mongodb.net/?retryWrites=true&w=majority&appName
db = client['TestMongo']  # Remplace par le nom de ta base de données

def lire_donnees_mongo(nom_collection):
    collection = db[nom_collection]

    # Mesurer le temps de lecture
    start_time = time.time()

    # Récupérer toutes les données de la collection
    documents = list(collection.find())

    end_time = time.time()

    # Afficher le nombre de documents
    print(f"Nombre de documents dans la collection '{nom_collection}': {len(documents)}")
    print(f"Temps de lecture des données : {end_time - start_time:.4f} secondes.")

    # Affichage des données
    for doc in documents:
        print(doc)

# Exécution principale (si nécessaire)
if __name__ == "__main__":
    lire_donnees_mongo("lesVols")
```

Figure 9 : fichier data.json

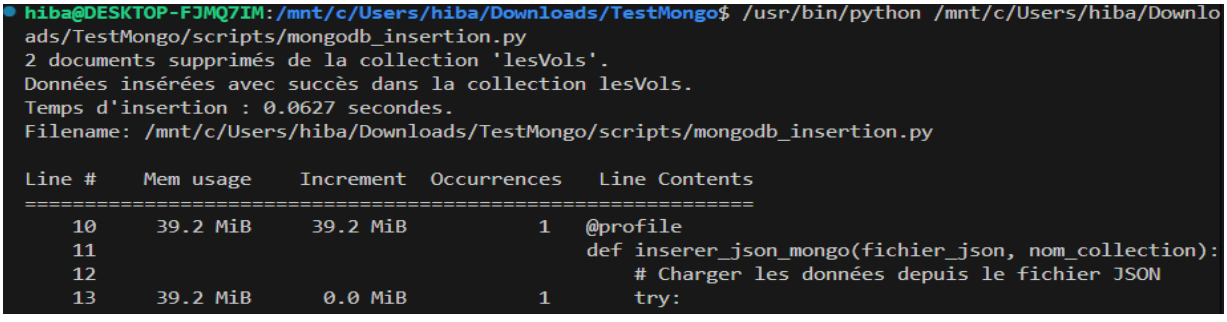
```
        "V141": {
            "VilleD": "Amsterdam",
            "VilleA": "Marseille",
            "DateD": "1/04/07",
            "HD time": "23:20",
            "DateA": "2/04/07",
            "HA time": "1:10",
            "NumPil": "5050",
            "NumAv": "101",
            "avion": {
                "NomAv": "Boeing 707",
                "CapAv": "350",
                "VilleAv": "Beyrouth"
            },
            "pilote": {
                "NomPil": "Leblanc",
                "NaisPil": "1960",
                "VillePil": "Marseille"
            },
            "reservations": {
                "1008": {
                    "NumVol": "V141",
                    "Classe": "Touriste",
                    "NbPlaces": "2"
                },
                "1011": {
                    "NumVol": "V141",
                    "Classe": "Economique",
                    "NbPlaces": "8"
                }
            }
        },
        "V142": {
            "VilleD": "Amsterdam",
```

Figure 10 : Résultat de redis_insertion.py



```
Vol V927 stocké dans Redis sous la clé vol:V927
Vol V928 stocké dans Redis sous la clé vol:V928
Vol V929 stocké dans Redis sous la clé vol:V929
Insertion Redis terminée en 0.23 secondes
Utilisation de la mémoire : 0.36 MiB
```

Figure 11 : Résultat de MongoDB Test d'insertion.py



```
hiba@DESKTOP-FJMQ7IM:/mnt/c/Users/hiba/Downloads/TestMongo$ /usr/bin/python /mnt/c/Users/hiba/Downlo
ads/TestMongo/scripts/mongodb_insertion.py
2 documents supprimés de la collection 'lesVols'.
Données insérées avec succès dans la collection lesVols.
Temps d'insertion : 0.0627 secondes.
Filename: /mnt/c/Users/hiba/Downloads/TestMongo/scripts/mongodb_insertion.py

Line #    Mem usage    Increment  Occurrences   Line Contents
=============================================================
    10     39.2 MiB     39.2 MiB           1   @profile
    11                                         def inserer_json_mongo(fichier_json, nom_collection):
    12                                             # Charger les données depuis le fichier JSON
    13     39.2 MiB      0.0 MiB           1       try:
```

Figure 12 : Résultat de MongoDB Test de lecture.py

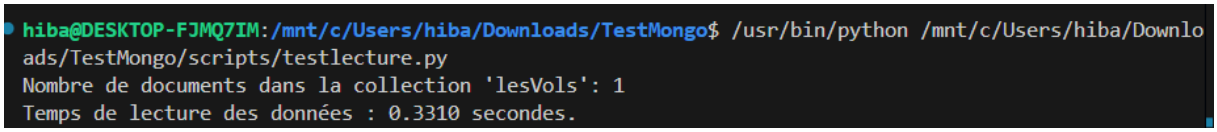


Figure 13 : Résultat de Redis Test de lecture.py

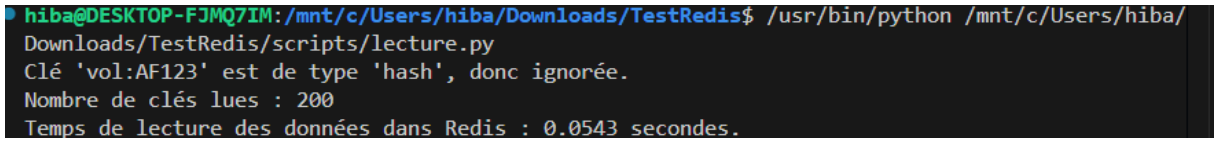


Figure 14 : Résultat de Redis Test mis-a-jour.py

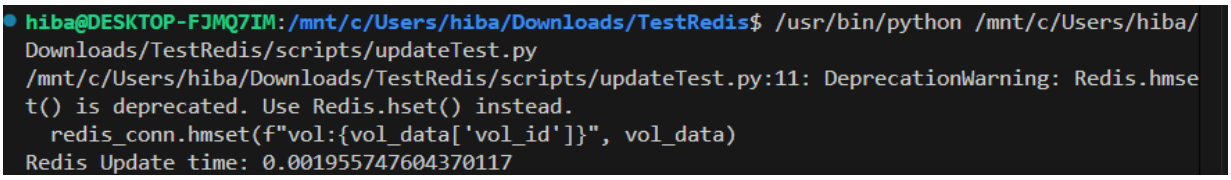


Figure 15 : Résultat de MongoDB Test mis-a-jour.py

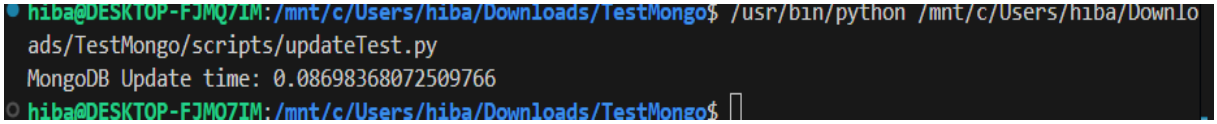


Figure 16 : Résultat de performance_test.py

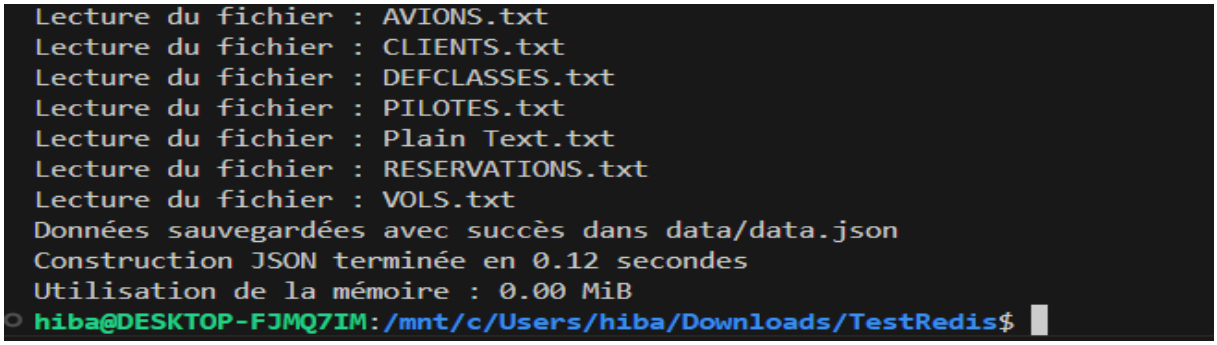


Figure 17 : Résultat de de jointure_tests.py

```
Jointure trouvée entre V523 et V690 sur VilleD avec la valeur 'Marseille'
Jointure trouvée entre V523 et V691 sur VilleD avec la valeur 'Marseille'
Jointure trouvée entre V523 et V692 sur VilleD avec la valeur 'Marseille'
Jointure trouvée entre V523 et V693 sur VilleD avec la valeur 'Marseille'
Jointure trouvée entre V523 et V694 sur VilleD avec la valeur 'Marseille'
Jointure terminée en 0.16 secondes
Utilisation de la mémoire : 0.32 MiB
```