



Projet :

Mesure de l'impact sur les performances de la mise en cache et du sharding sur des application web

Membres du groupe :

Arona Dia DIALLO

Papa Alioune Dramé KANDJI

El Hadji Abdoulaye NDIAYE

INTRODUCTION	4
I- Organisation et Diagramme de Gantt	4
1. Organisation et affectation des taches	4
2. Diagramme de Gantt	5
II- Paramètres de l'environnement matériel de test	5
III- Paramètres de Configuration	6
1. Scénario 0 (initial)	6
2. Scénario 1 (initial + cache)	7
3. Scénario 2 (initial + base distribuée)	8
4. Scénario 3 (initial + cache+base distribuée)	9
IV- Graphe d'évolution	9
1. Scénario 0	9
2. Scénario 1	10
3. Scénario 2	11
4. Scénario 3	12
V- Analyse statistique des données	13
1. Scénario 0	13
a) Nombre de requêtes par rapport au temps	13
b) Taille des paquets par rapport au temps	13
2. Scénario 1	14
a) Nombres de requêtes par rapport au temps	14
b) Taille des paquets par rapport au temps	14
3. Scénario 2	14
a) Nombres de requêtes par rapport au temps	14
b) Taille des paquets par rapport au temps	15
4. Scénario 3	15
a) Nombres de requêtes par rapport au temps	15
b) Taille des paquets par rapport au temps	15
Conclusion	16

<u>Figure 1 :Graphe d'évolution du nombre de requêtes par rapport au temps</u>	<u>9</u>
<u>Figure 2 : Graphe d'évolution de la taille des paquets par rapport au temps</u>	<u>10</u>
<u>Figure 3:Graphe d'évolution du nombre de requêtes par rapport au temps</u>	<u>10</u>
<u>Figure 4:Graphe d'évolution de la taille des paquets par rapport au temps</u>	<u>11</u>
<u>Figure 5:Graphe d'évolution du nombres de requêtes par rapport au temps</u>	<u>11</u>
<u>Figure 6:Graphe d'évolution de la taille des paquets par rapport au temps</u>	<u>12</u>
<u>Figure 7:Graphe d'évolution du nombre de requêtes par rapport au temps</u>	<u>12</u>
<u>Figure 8:Graphe d'évolution de la taille des paquets par rapport au temps</u>	<u>13</u>

INTRODUCTION

Avec la montée du nombre d'utilisateurs sur son site web, une entreprise utilisant WordPress se doit de garantir une expérience utilisateur fluide et rapide. Pour cela, il est crucial d'investir dans une infrastructure adaptée afin d'améliorer les performances du site. Dans cette optique, l'entreprise a décidé d'organiser une étude comparative entre différents scénarios, dans le but de mesurer l'impact de son investissement sur l'amélioration des performances du site. Les scénarios étudiés vont de l'installation initiale avec une architecture AMP classique à l'ajout de serveurs cache REDIS et de bases de données distribuées. L'objectif de cette étude comparative est de déterminer quelle infrastructure sera la plus rentable pour l'entreprise en termes de performances et de coûts. Cela permettra à l'entreprise de prendre des décisions éclairées quant à l'investissement à réaliser pour garantir des performances optimales à ses utilisateurs.

I- Organisation et Diagramme de Gantt

1. Organisation et affectation des taches

Dans le cadre de l'organisation de ce projet, chaque membre de l'équipe a été affecté à une partie spécifique de l'architecture pour maximiser l'efficacité et l'expertise. Papa Alioune Dramé KANDJI est responsable de la partie liée aux bases de données et utilise Docker pour virtualiser les différentes bases de données nécessaires. Cela permet de créer des environnements de développement et de test identiques à ceux de la production, ce qui facilite le développement et le déploiement.

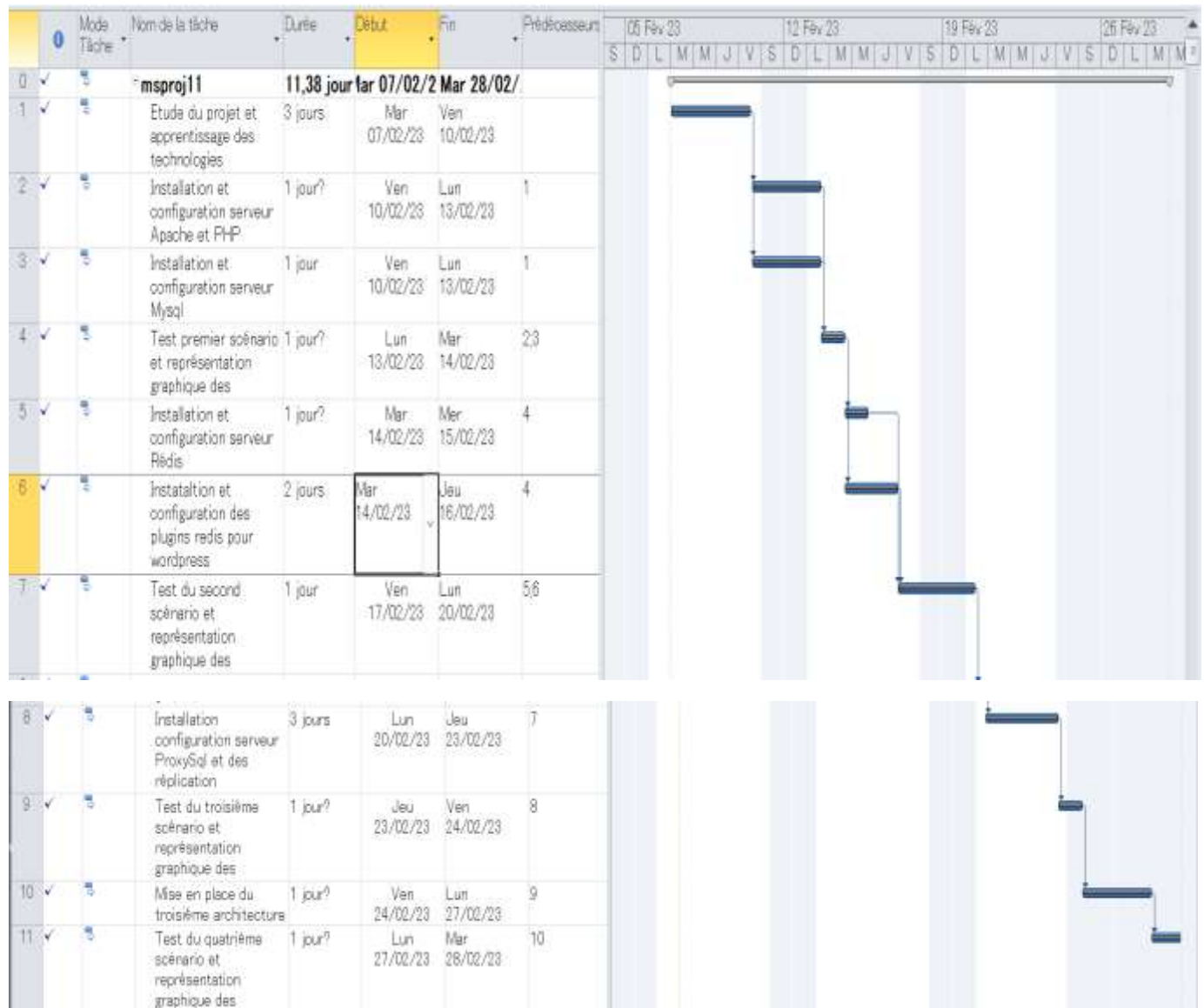
Arona Dia DIALLO est en charge du serveur AMP et de l'installation des plugins nécessaires au bon fonctionnement de l'application. Il est également responsable de la configuration et de la maintenance du serveur pour assurer des performances optimales. Il s'assure que les plugins sont compatibles avec les autres éléments de l'architecture et que tout est configuré correctement pour garantir une expérience utilisateur fluide.

El Hadji Abdoulaye NDIAYE est responsable de Redis et de l'écriture de scripts permettant d'exécuter des commandes WRK. Il travaille à l'amélioration des performances en utilisant Redis pour stocker des données fréquemment utilisées et en écrivant des scripts pour mesurer les performances et l'impact des changements apportés à l'architecture.

En travaillant de manière collaborative, chaque membre de l'équipe apporte son expertise et ses compétences pour atteindre l'objectif commun d'améliorer les performances du site WordPress et de garantir une expérience utilisateur fluide et rapide. Cette approche devrait permettre de

maximiser l'efficacité et de minimiser les coûts, tout en garantissant une performance optimale de l'architecture mise en place.

2. Diagramme de Gantt



II- Paramètres de l'environnement matériel de test

L'environnement de test utilisé pour ce projet était composé de 3 machines physiques A, B et C.

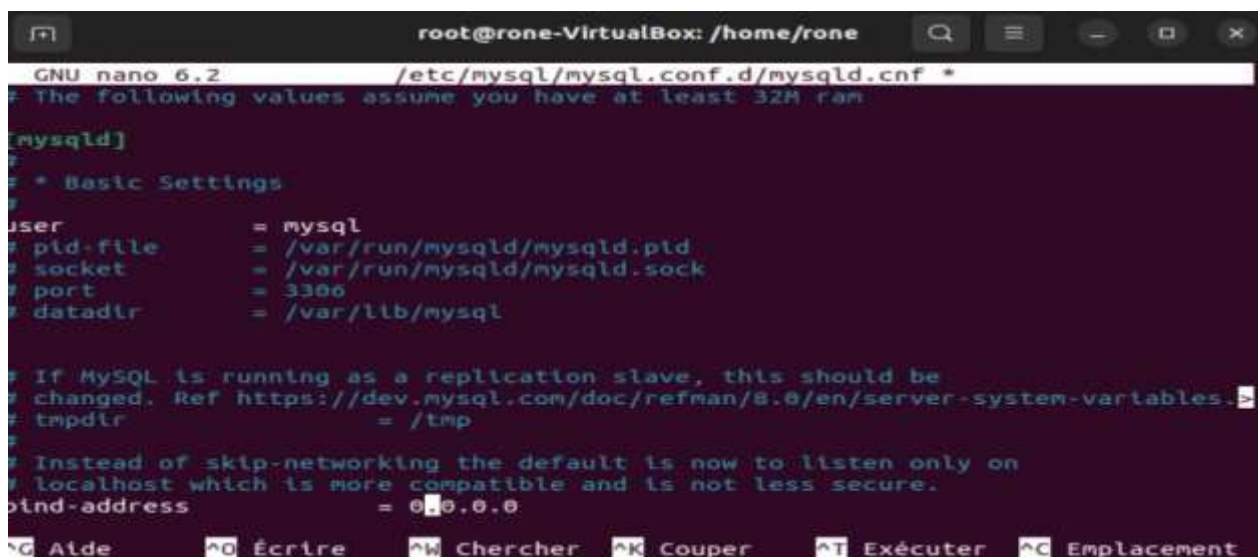
- **Machine physique A** : Elle est équipée d'un processeur intel core i5 de 10^{ème} génération et de 12 Go de RAM. Nous y avons installé Docker au niveau duquel nous avons créé trois conteneurs dont l'un servira de serveur master et les deux autres seront des slaves. Nous y avons également installé une machine virtuelle VM1 dont on a attribué 6 go de mémoire RAM pour y héberger le serveur proxySQL.
- **Machine physique B** : Elle est équipée d'un processeur intel core i5 de 11^{ème} génération et de 16 Go de RAM ; nous y avons hébergé le serveur redis et le logiciel wrk pour effectuer les requêtes.
- **Machine physique C** : Dans cette machine équipée de 16 giga de RAM et d'un processeur AMD, nous avons installé une machine virtuelle avec 4 Go de RAM et 2 cœur de processeur pour y héberger le serveur Web (Apache / PHP / Wordpress).

Pour relier l'ensemble des machines, nous avons utilisé le réseau wifi Konectel. Ces paramètres matériels et de réseau ont été choisis pour assurer une configuration réaliste de l'environnement de test et une comparaison précise des performances entre les différents scénarios testés.

III- Paramètres de Configuration

1. Scénario 0 (initial)

- **Configuration Serveur Mysql** : Le serveur Mysql est installé sur la machine physique A. Pour faire la configuration nous avons modifier le fichier de configuration pour permettre à la machine ayant le serveur apache d'utiliser la base de donnée en mettant le bind-address à 0.0.0.0.



```

root@rone-VirtualBox: /home/rone
GNU nano 6.2 /etc/mysql/mysql.conf.d/mysqld.cnf *
# The following values assume you have at least 32M ram

[mysqld]
#
# * Basic Settings
#
user                = mysql
pid-file            = /var/run/mysqld/mysqld.pid
socket              = /var/run/mysqld/mysqld.sock
port                = 3306
datadir             = /var/lib/mysql

# If MySQL is running as a replication slave, this should be
# changed. Ref https://dev.mysql.com/doc/refman/8.0/en/server-system-variables.html
# tmpdir             = /tmp

# Instead of skip-networking the default is now to listen only on
# localhost which is more compatible and is not less secure.
bind-address        = 0.0.0.0
  
```

- **Configuration Serveur Web** : Nous avons installer le serveur dans la machine virtuelle VM2. Nous l'avons configuré en modifiant le fichier wp-config pour nous connecter à la base de donnée configurée plus haut.

```

/* -- Database settings -- You can get this info from your web host -- */
/* The name of the database for WordPress */
define( 'DB_NAME', 'wordpress' );

/* Database username */
define( 'DB_USER', 'wordpress_user' );

/* Database password */
define( 'DB_PASSWORD', 'password' );

/* Database hostname */
define( 'DB_HOST', '10.100.124.118' );

/* Database charset to use in creating database tables. */
define( 'DB_CHARSET', 'utf8' );

/* The database collate type you'd change this if it were. */
define( 'DB_COLLATE', '' );

/*#@#
MySQL database name prefix. You can use this to change the database name if you have a MySQL database
*/

```

2. Scénario 1 (initial + cache)

- **Configuration du serveur Redis :** Le serveur est installé sur la machine physique B, nous l'avons configuré en modifiant le fichier de configuration pour permettre à n'importe quel machine client de pouvoir se connecter sur le le serveur redis.

```

GNU nano 6.2                                redis.conf
# running on).
#
# IF YOU ARE SURE YOU WANT YOUR INSTANCE TO LISTEN TO ALL THE INTERFACES
# COMMENT OUT THE FOLLOWING LINE.
#
# You will also need to set a password unless you explicitly disable protected
# mode.
#
bind 0.0.0.0 -:::1

# By default, outgoing connections (from replica to master, from Sentinel to
# instances, cluster bus, etc.) are not bound to a specific local address. In
# most cases, this means the operating system will handle that based on routing

```

- **Configuration du Serveur Web :** Dans cette partie nous avons eu à installer un plugging wordpress pour permettre à notre site web de pouvoir utiliser un système de cache avec redis. Mais aussi nous avons modifié le fichier de configuration de façon à préciser les information du serveur redis.

```

/* Absolute path to the WordPress directory. */
if ( ! defined( 'ABSPATH' ) ) {
    define( 'ABSPATH', __DIR__ . '/' );
}

/* Sets up WordPress vars and included files. */
require_once ABSPATH . 'wp-settings.php';

define( 'WP_CACHE_KEY_SALT', 'your_unique_string_here' );
define( 'WP_CACHE', true );
define( 'WP_REDIS_HOST', '10.100.124.118' );
define( 'WP_REDIS_PORT', '6379' );
define( 'WP_REDIS_PASSWORD', '' );

define( 'FS_METHOD', 'direct' );
define( 'FS_CHMOD_DIR', 0777 );
define( 'FS_CHMOD_FILE', 0777 );

```

- **Configuration de la base de donnée :** aucune modification

3. Scénario 2 (initial + base distribuée)

- **Configuration des serveurs slave et du serveur master :** Nous avons utilisé docker avec un fichier yaml pour faire la configurations des conteneurs mysql.

```
1  version: '3'
2  services:
3    mysql-master:
4      image: mysql:latest
5      restart: always
6      environment:
7        MYSQL_ROOT_PASSWORD: passer
8        MYSQL_DATABASE: wordpress
9        MYSQL_USER: wp_user
10       MYSQL_PASSWORD: passer
11       volumes:
12         - ./mysql-master:/var/lib/mysql
13       ports:
14         - "0.0.0.0:3306:3306"
15     mysql-slave1:
16       image: mysql:latest
17       restart: always
18       environment:
19         MYSQL_ROOT_PASSWORD: passer
20         MYSQL_DATABASE: wordpress
21         MYSQL_USER: wp_user
22         MYSQL_PASSWORD: passer
23         MYSQL_MASTER_HOST: mysql-master
24         MYSQL_MASTER_PORT: 3306
25         MYSQL_REPLICATION_USER: repluser
26         MYSQL_REPLICATION_PASSWORD: passer
27       volumes:
28         - ./mysql-slave1:/var/lib/mysql
29       ports:
30         - "0.0.0.0:3307:3306"
31     mysql-slave2:
32       image: mysql:latest
33       restart: always
34       environment:
35         MYSQL_ROOT_PASSWORD: passer
36         MYSQL_DATABASE: wordpress
37         MYSQL_USER: wp_user
38         MYSQL_PASSWORD: passer
39         MYSQL_MASTER_HOST: mysql-master
40         MYSQL_MASTER_PORT: 3306
41         MYSQL_REPLICATION_USER: repluser
42         MYSQL_REPLICATION_PASSWORD: passer
43       volumes:
44         - ./mysql-slave2:/var/lib/mysql
45       ports:
```

- **Cofiguration du serveur proxy :** Nous avons défini des règles permettant à notre serveur proxy de savoir qu'une requête SELECT peut être faite sur un des trois serveur mysql et que la requête d'insertion, de modification et de supressions ne sont faites que sur le serveur master.


```

Admin> INSERT INTO mysql_query_rules (active, match_pattern, destination_hostgroup, cache_ttl) VALUES (1, '^SELECT .* FOR UPDATE', 1, NULL);
Query OK, 1 row affected (0,00 sec)

Admin> INSERT INTO mysql_query_rules (active, match_pattern, destination_hostgroup, cache_ttl) VALUES (1, '^SELECT .*', 2, NULL);
Query OK, 1 row affected (0,00 sec)

Admin> INSERT INTO mysql_query_rules (active, match_pattern, destination_hostgroup, cache_ttl) VALUES (1, '^INSERT .*', 1, NULL);
Query OK, 1 row affected (0,00 sec)

Admin> INSERT INTO mysql_query_rules (active, match_pattern, destination_hostgroup, cache_ttl) VALUES (1, '^UPDATE .*', 1, NULL);
Query OK, 1 row affected (0,00 sec)

Admin> INSERT INTO mysql_query_rules (active, match_pattern, destination_hostgroup, cache_ttl) VALUES (1, '^DELETE .*', 1, NULL);
Query OK, 1 row affected (0,00 sec)

Admin> LOAD MYSQL QUERY RULES TO RUNTIME;
Query OK, 0 rows affected (0,00 sec)

Admin> SAVE MYSQL QUERY RULES TO DISK;
Query OK, 0 rows affected (0,04 sec)

Admin> █

```

4. Scénario 3 (initial + cache+base distribuée)

Dans cette partie nous sommes resté sur la même architecture que celle précédemment effectuée et nous avons en plus activé le serveur redis au niveau de notre wordpress.

IV- Graphe d'évolution

1. Scénario 0

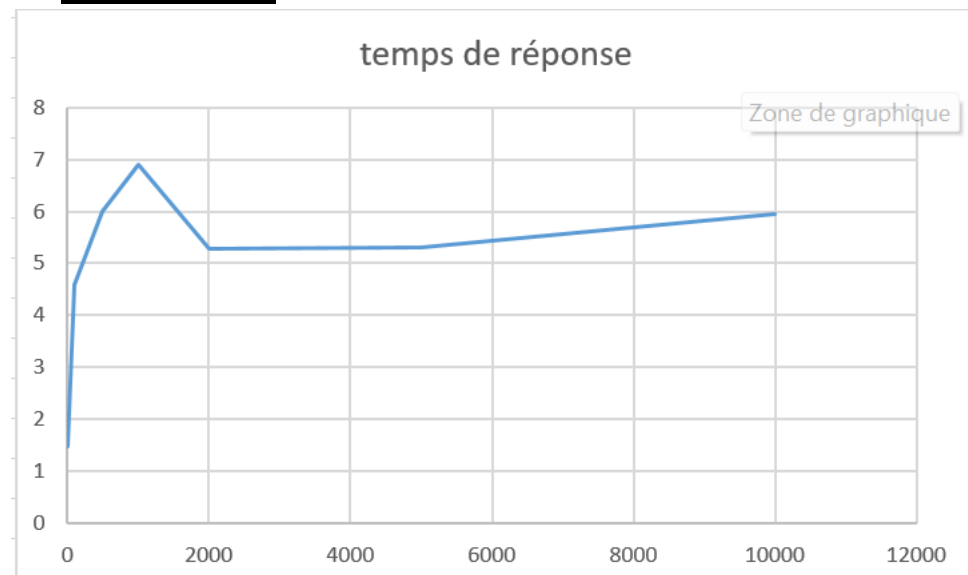


Figure 1 :Graphe d'évolution du nombre de requêtes par rapport au temps

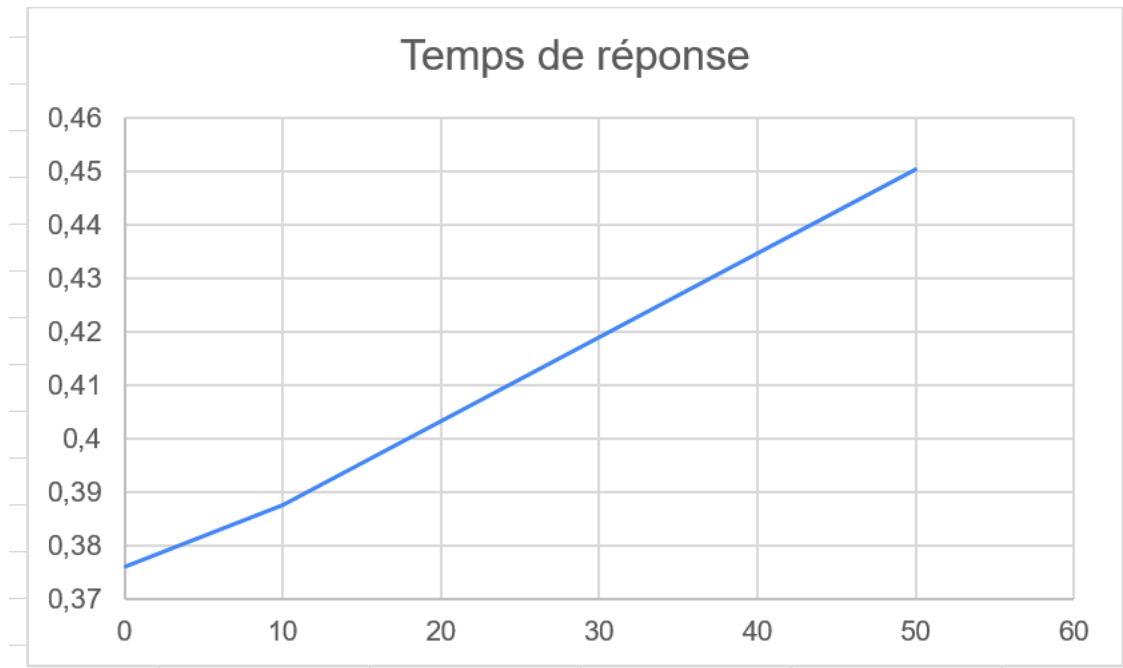


Figure 2 : Graphe d'évolution de la taille des paquets par rapport au temps

2. Scénario 1

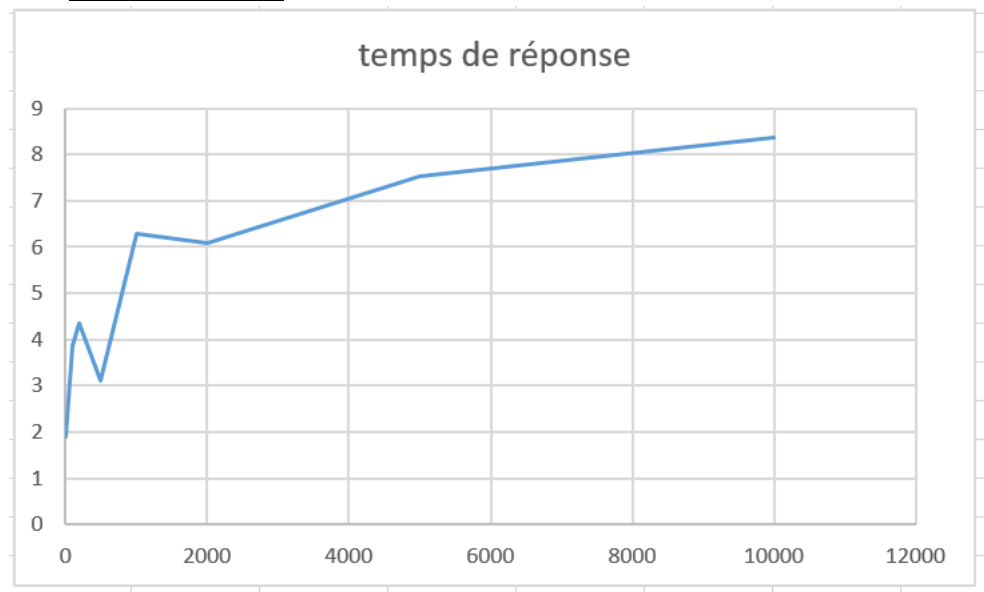


Figure 3:Graphe d'évolution du nombre de requêtes par rapport au temps

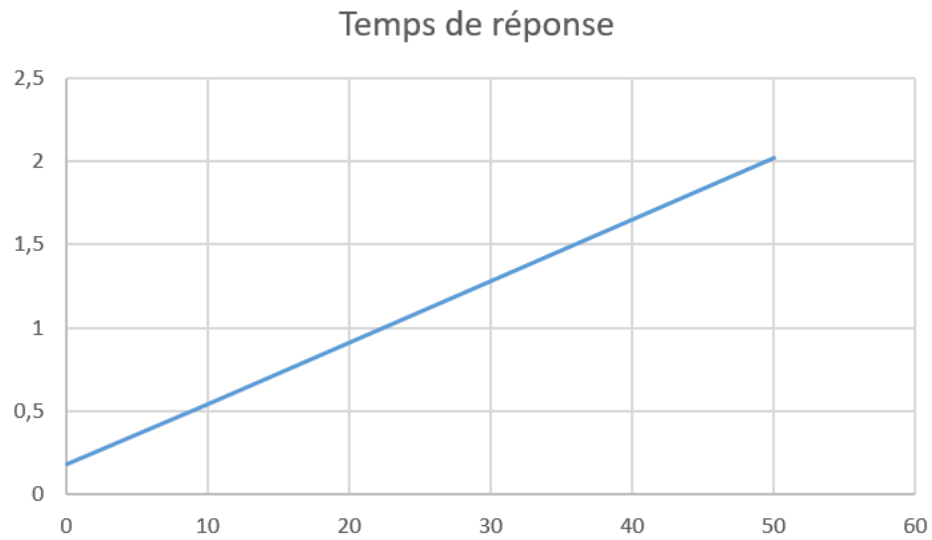


Figure 4: Graphe d'évolution de la taille des paquets par rapport au temps

3. Scénario 2

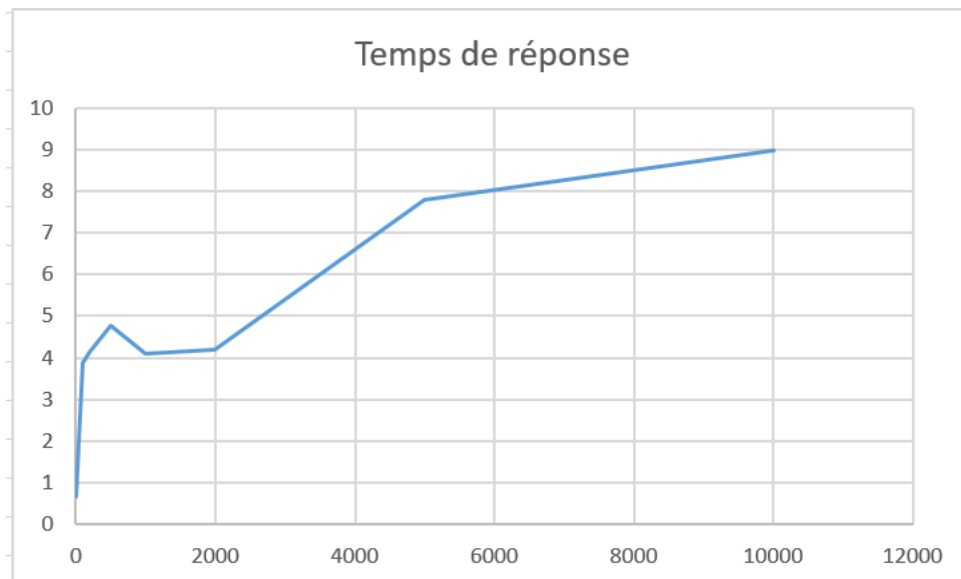


Figure 5: Graphe d'évolution du nombre de requêtes par rapport au temps

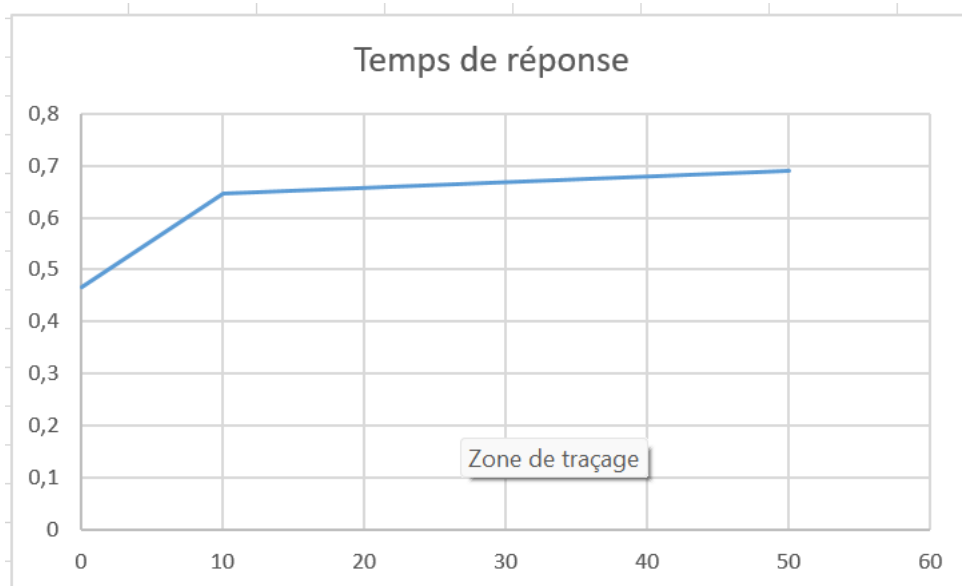


Figure 6:Graphe d'évolution de la taille des paquets par rapport au temps

4. Scénario 3

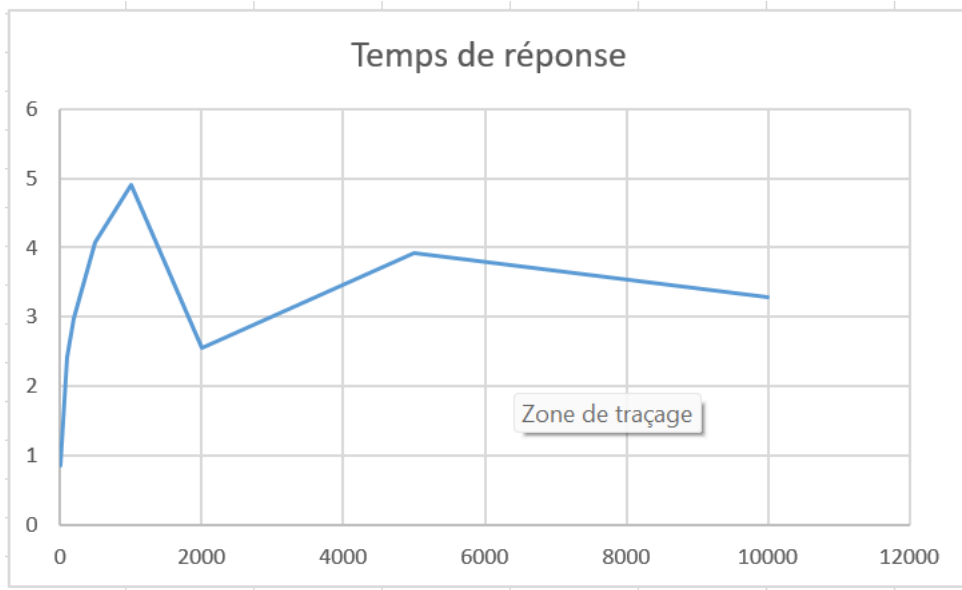


Figure 7:Graphe d'évolution du nombre de requêtes par rapport au temps

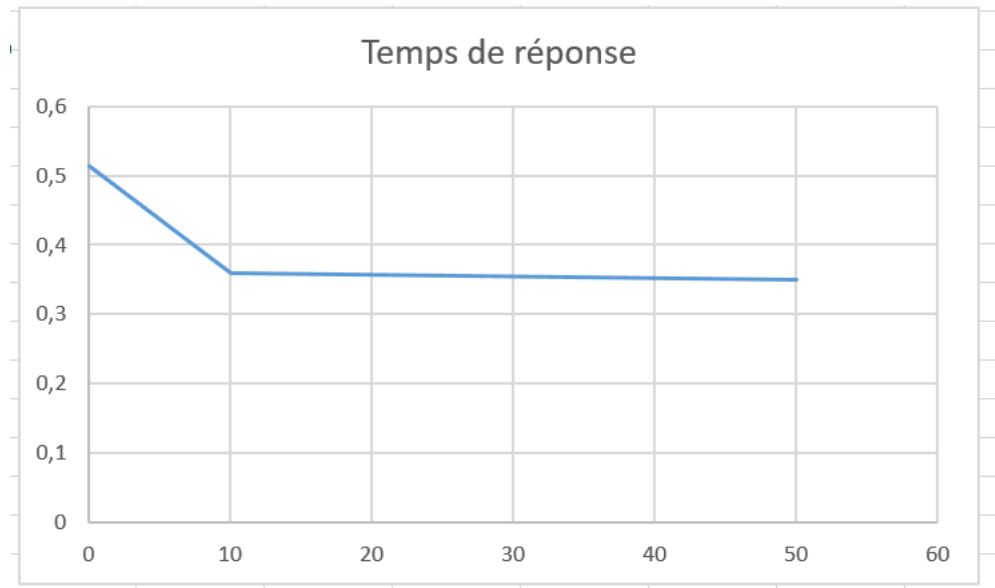


Figure 8: Graphe d'évolution de la taille des paquets par rapport au temps

V- Analyse statistique des données

1. Scénario 0

a) Nombre de requêtes par rapport au temps

➤ Analyse

Au vue des données obtenues nous avons constaté :

- Entre 10 et 1000 requêtes le temps de réponse moyenne des requêtes croît, allant de 1,49 secondes pour atteindre son pic à 6,89 secondes.
- Ensuite de 1000 requêtes à 2000 requêtes le temps de réponse moyenne des requêtes décroît allant de 6,89 secondes pour atteindre 5,27 secondes.
- A partir de 2000 requêtes le temps de réponse croît légèrement allant de 5,27 à 5,96 secondes

➤ Interprétation

Les résultats suggèrent que le serveur peut gérer efficacement jusqu'à environ 2000 requêtes, mais au-delà, le temps de réponse commence à augmenter de manière significative, ce qui peut affecter négativement les performances du système et l'expérience utilisateur.

b) Taille des paquets par rapport au temps

➤ Analyse

Au vue des données obtenues nous avons constaté que plus la taille des paquets est grande plus le temps de réponse est élevé.

➤ Interprétation

Cela peut s'expliquer par le fait que la taille de la requête influe sur la quantité de données à transférer entre le client et le serveur, et que cela peut avoir un impact sur les performances du serveur.

2. Scénario 1

a) Nombres de requêtes par rapport au temps

➤ Analyse

Au vue des données obtenues nous avons constaté que :

- Entre 10 et 200 requêtes : le temps de réponse augmente en passant de 1,19 à 4,36 secondes.
- Entre 200 et 500 requêtes : le temps de réponse diminue passant de 4,36 à 3,11 secondes.
- A partir de 500 requêtes : le temps de réponse augmente jusqu'à atteindre son pic à 8,37 secondes.

➤ Interprétation

Cela peut s'expliquer par le fait que le site peut gérer une charge légère à modérée et que l'ajout du serveur cache REDIS a permis d'améliorer les performances du site pour une charge plus importante.

b) Taille des paquets par rapport au temps

➤ Analyse

On constate que plus la taille des paquets est importante plus le temps de réponse augmente.

➤ Interprétation

Ces résultats suggèrent que le site peut traiter rapidement les paquets de petite taille, mais que le temps de réponse augmente avec la taille des paquets. Cela peut être dû au temps nécessaire pour transférer les données plus volumineuses, ainsi qu'au temps de traitement nécessaire pour traiter les données plus importantes.

3. Scénario 2

a) Nombres de requêtes par rapport au temps

➤ Analyse

Au vue des données obtenues nous avons constaté que :

- Entre 10 et 500 requêtes : le temps de réponse augmente en passant de 0,64 à 4,78 secondes.

- Entre 500 et 1000 requêtes : le temps de réponse diminue passant de 4,78 à 4,09 secondes.
- De 1000 à 2000 requêtes : le temps de réponse est quasi constant à 4,9.
- A partir de 2000 requête : le temps de réponse augmente jusqu'à atteindre 8,98 secondes.

➤ **Interprétation**

Ces résultats suggèrent que le site gère efficacement un nombre de requêtes inférieure à 2000. Mais à partir de 2000 le site du mal à gérer, mais nous constatons tout de même que le système de base de données distribuées permet d'obtenir de meilleures performances par rapport au scénario 0.

b) Taille des paquets par rapport au temps

➤ **Analyse**

Au vue des données obtenues nous avons constaté que :

- De 0,1 K à 10K : le temps de réponse augmente brusque allant de 0,46K à 0,69 secondes.
- Ensuite nous pouvons noter une croissance légère allant de 0,64 à 0,69 secondes.

➤ **Interprétation**

Notre site semble gérer un petit peu un volume de requêtes plus important.

4. Scénario 3

a) Nombres de requêtes par rapport au temps

➤ **Analyse**

Au vue des données obtenues nous avons constaté que :

- Entre 10 et 1000 requêtes : le temps de réponse augmente en passant de 0,86 à 4,91 secondes.
- De 1000 à 2000 requêtes : le temps de réponse décroît en de 4,91 à 2,55 secondes.
- De 2000 à 5000 requêtes : le temps de réponse croît de 2,55 à 3,92 secondes.
- De 5000 à 10000 requêtes : le temps de réponse diminue légèrement de 3,92 jusqu'à 3,28 secondes.

➤ **Interprétation**

Avec des requêtes élevées on remarque que le serveur a été en mesure de s'adapter et de mieux gérer les requêtes à mesure qu'elles arrivaient.

b) Taille des paquets par rapport au temps

➤ **Analyse**

Au vue des données obtenues nous avons constaté que :

- De 0,1 K à 10K : le temps de réponse diminue allant de 0,51 à 0,36 secondes.

- Ensuite nous pouvons noter une faible décroissance de 0,36 à 0,35 secondes.

➤ **Interprétation**

Cela suggère que le serveur peut traiter plus efficacement des paquets plus grands, ce qui permet d'améliorer les performances du site.

Conclusion

Il est difficile de donner une conclusion générale sur les conditions dans lesquelles l'investissement dans le cache est plus intéressant que l'investissement dans la base de données distribuées, car cela dépendra des besoins et des contraintes spécifiques de chaque entreprise. Cependant, voici quelques éléments qui peuvent aider à orienter la décision :

L'investissement dans le cache est particulièrement utile si le site génère beaucoup de trafic, car le cache permet de réduire le temps de réponse et de soulager la charge sur le serveur Web.

L'investissement dans la base de données distribuées est particulièrement utile si le site gère beaucoup de données et que les requêtes sur la base de données sont très fréquentes, car la distribution de la base de données permet de répartir la charge sur plusieurs serveurs et d'améliorer les performances globales.

Dans certains cas, il peut être intéressant de combiner les deux investissements (cache et base de données distribuées) pour bénéficier de leurs avantages respectifs et améliorer encore plus les performances du site.

En somme, le choix entre investir dans le cache ou dans la base de données distribuées dépendra des besoins et des contraintes spécifiques de chaque entreprise, ainsi que de la nature du site et du trafic qu'il génère. Il conviendra donc d'analyser les performances du site dans différents scénarios et de comparer les coûts et les avantages des différentes options pour prendre une décision éclairée.