Rapport de Projet MED

Réseau de files d'attentes

Réalisé par

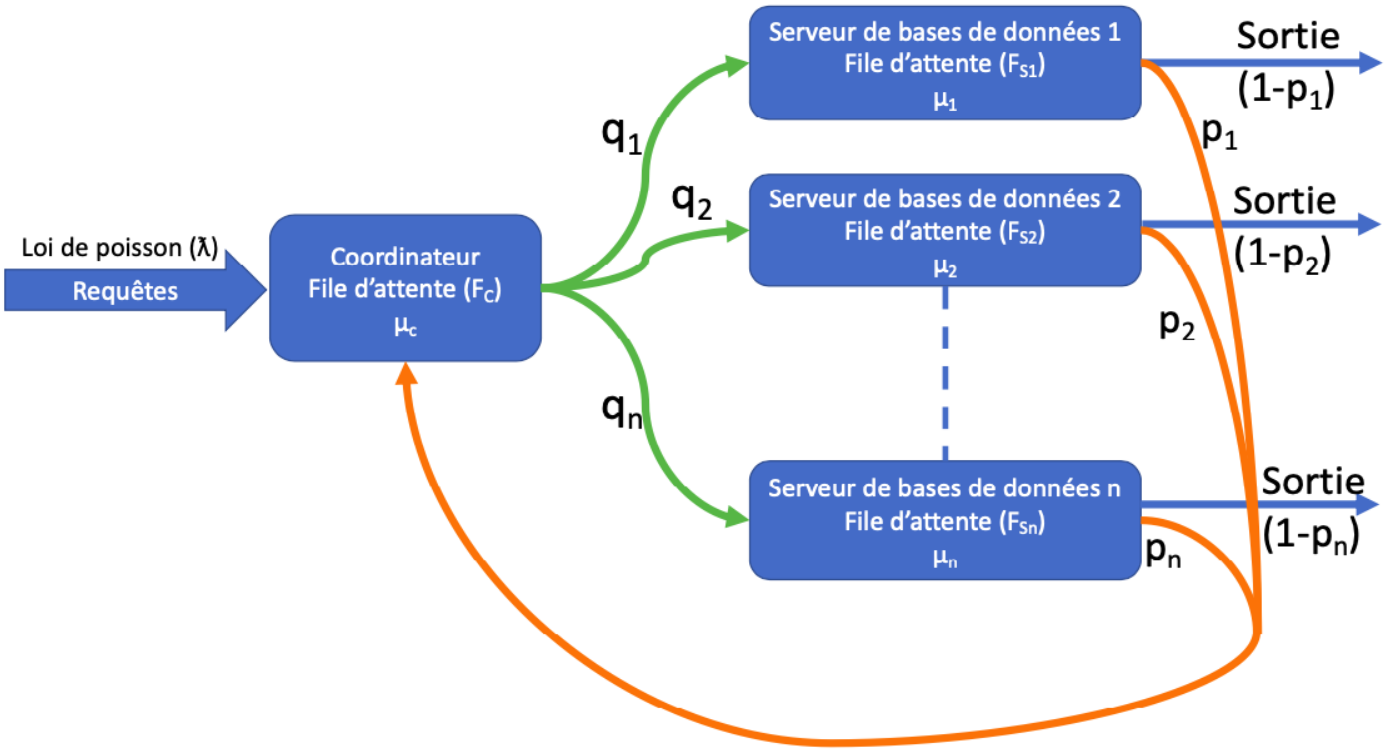
Madjid Taoualit et Yugurten Merzouk

1/Introduction :

-->L'idée de projet consiste à simuler le fonctionnement d'une base de données distribuée sur plusieurs serveurs en utilisant le langage (Java), sachant que les requêtes arrivent à un coordinateur en suivant une **loi de Poisson** de paramètres **(ƛ)**, ce dernier s'occupe de renvoyer ces requêtes aux serveurs de notre base de données.

-->Chaque serveur traite les requêtes qui y sont envoyées par **le coordinateur**, suivant une probabilité (la probabilité que le serveur traite la requête et répond à cette dernière) cette probabilité est définie .

-->Chaque serveur dispose d'un temps pour le traitement de chaque requête élémentaire (Le temps de service).



-->Chaque serveur soit il répond à une requête et la traite complètement, soit la requête est suivie d'une autre requête qui est envoyée elle-même au coordinateur qui s'occupe de la renvoyer à un serveur qui est capable de la traiter.

-->A chaque fin de traitement d'une requête par un serveur donné, la sortie définitive de la requête ou son renvoi au coordinateur est décidé selon une probabilité **(p).**

-->Pour avoir la **meilleure simulation** possible on doit mettre en œuvre un programme qui va rendre possible d'avoir un nombre de **serveurs paramétrés à l'avance** qui disposent eux aussi des paramètres tels que le temps de traitement des requêtes pour **chaque serveur**, aussi les probabilités de traitement d'entrées et sorties dans les différents serveurs de la base de données distribuée.

2/Analyse-conception (modèle objet):

-->Pour l'implémentation de notre programme on doit avoir implémenter 3 classes :

1/La classe (Requete): C'est la classe qui représente une requête, chaque requête est présentée par un identifiant unique, sa date initiale de rentrée au système, sa date courante dans le système et sa date de sortie de système.

```
3   public class Requete{
4       private int id;
5       private double dateEntreeInitiale;
6       private double dateEntreeCourante;
7       private double dateSortie;
8
9       public Requete(int id,double dateEntreeInitiale,double dateEntreeCourante,double dateSortie) {
10          this.id = id;
11          this.dateEntreeInitiale = dateEntreeInitiale;
12          this.dateEntreeCourante = dateEntreeCourante;
13          this.dateSortie = dateSortie;
14      }
```

2/La classe (Serveur) : Cette classe nous permet de présenter un serveur qui est défini par son identifiant, la probabilité d'entrée **(q)**, la probabilité de sortie **(1-p)** (Dans notre cas on la définit par **p**), le temps de traitement des requêtes, une liste chaînée pour enchaîner les différentes requêtes qui arrivent au serveur et un nombre qui représentes le nombre de requêtes envoyées par le coordinateur au serveur.

-->Cette classe va nous permettre de définir la probabilité d'entrée et la probabilité de sortie des requêtes dans chaque serveur, aussi le temps de traitement de chaque requête ce qui va nous faciliter la tâche de simulation et de test de notre programme en attribuant des différents paramètres **(p,q et temps de traitement)** à chaque serveur.

-->Dans ce programme on peut avoir de **1 à n** serveurs.

```
3   public class Serveur {
4
5       private int id;
6       private double q;//Les probabilites d entree au serveur
7       private double p;//Les probabilites de sortie de serveur
8       private double temps;
9       private ArrayList<Requete> serveur;
10      private int compequ;
11
12      public Serveur(int id,double q,double p,double temps,ArrayList<Requete> serveur,int compequ){
13
14          this.id=id;
15          this.q=q;
16          this.p=p;
17          this.temps=temps;
18          this.serveur=serveur;
19          this.compequ=compequ;
20
21      }
```

1/La classe (ReseauFilesAttentes): C'est la classe qui définit toutes les méthodes et les fonctions nécessaires pour que notre programme soit fonctionnel.

-->Cette classe définit 10 **méthodes/Fonctions** :

1/addRequeteCoordinateur(Requete r) : Elle permet d'ajouter les requêtes au coordinateur sachant que les requêtes arrivent à ce dernier en suivant une loi de poisson qui nous permet d'avoir un nombre précis de requêtes pendant une durée donnée.

2/trierCoordinateur() : Elle nous permet de trier le coordinateur en suivant la date courante de la requête dans le système

3/trierSortie() : Elle nous permet de trier les requêtes sorties des serveurs suivant la date de sortie des requêtes, aussi d'ajouter des requêtes à la la liste chaînée qui doit comporter toutes les requêtes sorties de tout les serveurs.

4/tempsTraitementRequete() : Elle nous permet de calculer le temps de traitement de chaque requête sortie de tout les serveurs de notre base de donnée distribuée.

5/leTempsMoyen() : Elle permet de calculer le temps moyen passé dans le système par une requête depuis sa date initiale de sa rentrée dans le système (le coordinateur) jusqu'à sa date de sortie de système (sortie de serveur).

6/processus() : Elle permet d'organiser tout le processus de simulation de la base de données distribuée, notamment gérer le processus d'envoi des requêtes depuis le coordinateur vers les serveurs puis le traitement de ces requêtes puis au final la sortie des requêtes des serveurs ou le renvoi des requêtes vers le coordinateur.

-->Cette méthode est la méthode principale de notre programme elle permet d'utiliser les probabilités d'entrées et sorties de chaque serveur pour simuler le fonctionnement de tout le système de la base de données distribuée.

7/sortieALaFin() : Elle permet de supprimer de la liste chaînées des requêtes en sortie toutes les requêtes dont le temps de sortie est supérieur à la durée de simulation **(duree)**.

8/extraireRequete(int id) :Elle permet d'extraire une requête depuis la liste chaînée qui comporte toutes les requêtes sorties de tout les serveurs en utilisant l'identifiant de la requête.

9/enregistrerLesDonnees(String filename, String partieNom, String contenu) : Elle permet d'enregistrer les données de simulation générées par notre programme dans des fichiers **(.dat)** pour en pouvoir tracer les courbes en utilisant **GnoPlot**.

10/equation4(double lambda) : Elle permet d'alimenter le coordinateur en requêtes en utilisant **une loi de poisson** de paramètre **(ƛ)**

-->En plus de toutes ces méthodes et fonctions on a des **(Getteurs)** et des **(Setteurs)** qui nous permettent de faciliter l'usage des **attributs** et les **(méthodes/fonctions)** de notre classe.

3/Le différentes simulations effectuées :

-->Dans un premier temps on suppose que les probabilités **(entrée q)** et **(sortie p)** sont pareilles pour tout les serveur ce qui veut dite à chaque serveur **(q=1/nombre de serveurs)** pareil pour p, **(p=1/nombre de serveurs)**.

-->Dans ce premier temps on fait une simulation sur deux serveurs, donc **(q=1/2)** et **(p=1/2)**.

-->Pour avoir les meilleurs résultats possibles on fait une simulation sur une durée assez grande **(100000 ms)**.

-->Le temps de service (envoi) des requêtes est **10 ms** (le temps mis par le coordinateur).

-->Le temps de service de **(serveur 1)** est **(100 ms)** et le temps de service de **(serveur 2)** est **(200 ms)**.

Résultats :

-->On remarque qu'un régime permanent se rétablie, donc il y a une certaine convergence dans le système qui se définie par un même temps de traitement des requêtes dans les deux serveurs sachant que le temps de traitement des requêtes est différent dans les deux serveurs, ce qui en résulte un nombre presque pareil de requêtes traitées par chaque serveur

```
Le nombre de requetes traitées (sorties) de serveur numero 1 est 157
Le nombre de requetes traitées (sorties) de serveur numero 2 est 160
```

-->On remarque que le temps de traitement d'une requête (on ne parle pas de sous-requêtes) devient presque pareil pour toutes les requêtes pour les deux serveurs malgré que les temps de de service des deux serveurs sont différents, on explique ça par le retraitement des sous-requêtes non traitées par un serveur donné.

```
Le temps de traitement de la requete numero 1 est 44932.69292636054
Le temps de traitement de la requete numero 2 est 44932.69292636054
Le temps de traitement de la requete numero 3 est 44932.69292636054
Le temps de traitement de la requete numero 4 est 44932.69292636054
```

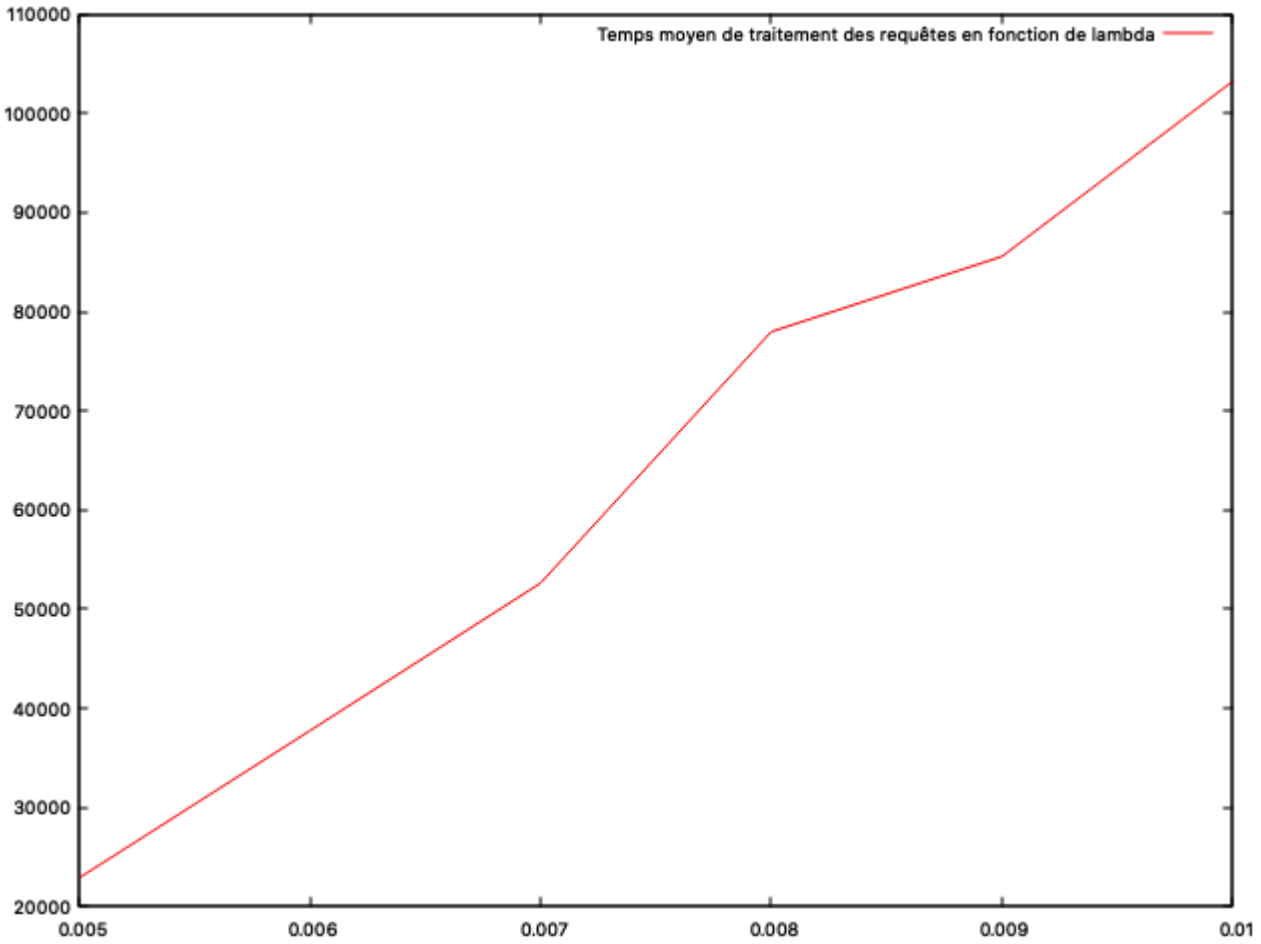
-->On remarque que le nombre de requêtes présentes dans un système augmente en fonction de la durée de la simulation, en fonction de **(ƛ)** aussi en fonction des temps service des différents services de la base de donnée distribuée, quand la durée de la simulation augmente le nombre de de requêtes présentes dans le système augmente, quand le temps de services des serveur diminue le nombre de requêtes présentes dans le système diminue, aussi quand **(ƛ)** est plus élevé e nombre de requêtes présentes dans le système augmente.

Exemple :

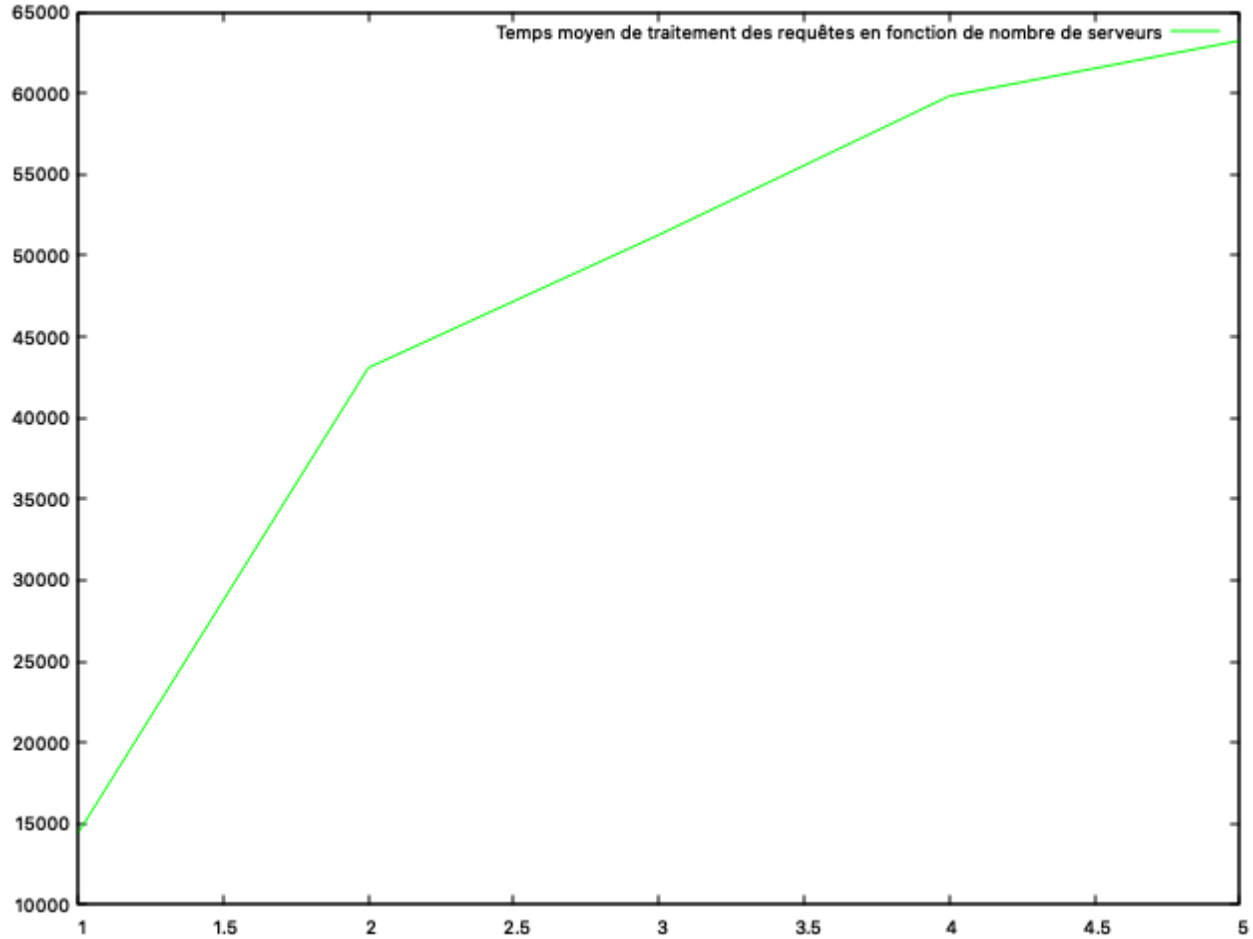
-->Par exemple pour une durée de simulation de **(100000 ms)** et **(ƛ=0.006)** en utilisant uniquement deux serveurs qui ont des temps de service (100 ms) et (200 ms) respectivement :

```
Le nombre de requêtes entrées dans le système durant la période de simulation est : 1930
```

-->Le temps moyen de traitement des requêtes en fonction de **(ƛ),** on a choisi des valeurs minimales de **(ƛ)** pour limiter le temps d'exécution de programme.



-->Le temps moyen de traitement des requêtes en fonction de nombre de serveurs**,** on a utilisé de 1 jusqu'à **5 serveurs** sachant que le temps de service de chaque serveur est égal à **(200 ms)**.



-->Le nombre moyen de clients (requêtes) présents dans chaque file, et dans le système complet :
-->Par exemple pour **(lambda=0.006)**, **(durée=100000ms)** :
-->Le nombre moyen de requêtes dans le système :

467

-->Le nombre moyen de requêtes dans chaque serveur:

```
Le nombre de requetes envoyees au serveur numero 1 est 467
Le nombre de requetes envoyees au serveur numero 2 est 467
```

-->Le nombre de requêtes entrées dans le système pendant la période de simulation :

```
Le nombre de requêtes entrées dans le système durant la période de simulation est : 1894
```

-->Le nombre moyen des requêtes sorties de système :

```
Le nombre de requêtes sorties dans le systeme durant la periode de simulation est : 238
```

-->Le nombre de requêtes traitées par chaque serveur :

```
Le nombre de requetes traitées (sorties) de serveur numero 1 est 168
Le nombre de requetes traitées (sorties) de serveur numero 2 est 170
```

-->Analyse de théorême de Jackson :

-->Le **théorême de jackson** est en fait plus général dans le cas ou **(λ)** varie en fonction du nombre de clients dans le réseau et les taux de service varient en fonction des **(k)**.

-->.Les clients sont servis un par un dans chaque file. Le **i-ième** serveur fonctionne de manière markovienne, avec un taux de service **μi(ni)** qui dépend du nombre **(ni)** de clients présents dans la file. Autrement dit, lorsque **(ni**) clients sont présents dans la file, le temps de service d'un client est exponentiel de paramètre **μi(ni)**.

-->Les temps d'inter-arrivée, les temps de service et les choix aléatoires des clients en entrée et en sortie de chaque file, sont des variables aléatoires indépendantes dans leur ensemble. Les files étant numérotées de 1 à K, un client arrive de l'extérieur, noté comme une **0-ième file**, parcourt un certain nombre de files de manière aléatoire, puis se dirige vers l'extérieur, noté comme une **(K + 1)-ième file**. La suite des files parcourues par un client est une chaîne de **Markov** sur **{0, . . . , K + 1}**, partant de 0 et telle que **K + 1** est un état absorbant. Nous supposons que cette file n'a pas d'autres composantes irréductibles récurrentes, c'est-à-dire qu'un client qui rentre dans le système ne peut pas y rester bloqué : il en sortira forcément. L'état du système sera codé par un vecteur d'entiers **n = (n1, . . . , nk)**, dont la **i-ième** coordonnée représente le nombre de clients présents dans la file numéro **i**.

-->Exactement c'est ce qu'on peut voir dans notre programme dont les requêtes présentent des clients

seulement dans notre programme on a facilité la tâche en ne prenant pas trop d'importance pour les sous requêtes.

-->Le nombre moyen de requêtes dans le système est lié en premier degré à la valeur de **(ƛ)** c'est ce qu'on a pu voir dans la simulation précédente.

-->Le temps moyen de présence représente la durée de vie d'une requête dans notre système de base de données distribuée depuis son arrivée au coordinateur jusqu'à sa sortie.

4/L'amélioration des performances de notre programme:

-->La première façon consiste à modifier les probabilités de routage **qi**

-->En effectuant la simulation sur des différents paramètres **qi** on a remarqué une amélioration des performance de notre programme en matière de temps notamment quand on augmente la valeur de **qi** pour les serveur dont le temps moyen de service est minimal.

-->Le routage dynamique augmente significativement les performances de notre programme, il consiste à envoyer les requêtes au serveurs les moins chargés pour améliorer le temps de sortie des requêtes notamment les serveurs dont le temps de réponse est minimal.

--->Si on utilise le routage dynamique ou on modifie les probabilités de routage **qi** d'une base de donnée distribuée on va remarquer une grande amélioration au niveau de temps de réponses aux différentes requêtes ce qui va améliorer l'utilisation des programmes qui utilise cette base de données

-->On pourra également ajouter plus de clients sans avoir des problèmes d'attente.

4/Répartiton de travail:

-->Pour réaliser ce projet on a travaillé ensemble pendant toutes les étapes.