

Université de Versailles

Saint-Quentin-en-Yvelines

**Rapport du projet de Simulation**

Ferme de Serveur

Réalisé par :

Quentin Soumet

Théotime Turmel

Avril 2025

**Table des Matiere**

1. Introduction

2. Modélisation du système

2.1 - Description générale

2.2 - Hypothèses et distributions

2.3 - Paramètres et contraintes

3. Méthodologie

3.1 - Outils utilisés

3.2 - Architecture de la simulation

3.3 - Déroulement d’une simulation

3.4 - Critères d’évaluation

4. Résultats expérimentaux

4.1. Temps de réponse moyen

4.2. Taux de perte

4.3. Détermination des seuils critiques de λ (taux de perte ≥ 5%)

5. Conclusion

**Introduction :**

Dans notre projet de simulation pour l'année universitaire 2024-2025, nous allons explorer le fonctionnement d'une ferme de serveurs capable de gérer des requêtes informatiques. L'objectif principal est de peaufiner l'organisation des serveurs afin de réduire au maximum le temps que le système consacre au traitement des requêtes, tout en minimisant les pertes dues à la surcharge.

Pour modéliser ce système, nous utiliserons une simulation événementielle en Python. Cette méthode nous permet de représenter de manière dynamique l'arrivée des requêtes, leur traitement par un routeur central, et leur distribution aux serveurs disponibles. Le comportement du système repose sur deux paramètres clés :

* λ, le taux d'arrivée des requêtes, qui suit une loi exponentielle.
* C, le nombre de groupes spécialisés dans lesquels les serveurs sont organisés. Nous visons à évaluer : - Le temps moyen de réponse des requêtes.
* Le taux de perte, qui doit rester en dessous de 5%.
* Le choix de C en fonction de différentes valeurs de λ.

Pour cela, nous commencerons par modéliser le système, suivie de la méthodologie que nous avons adoptée pour cette simulation. Ensuite, nous comparerons les graphiques des résultats pour chaque configuration du système, et enfin, nous conclurons notre rapport en présentant l'option optimale selon la charge.

**2) Modélisation du système**

**2.1. Description générale**

Le système que nous modélisons est une **ferme de serveurs composée de 12 machines** identiques, qui ont pour but de traiter des requêtes entrantes de manière rapide et efficace. Pour optimiser les performances du système, il est possible de **répartir les serveurs en groupes spécialisés** selon les types de requêtes à traiter.

Le nombre de groupes, noté **C**, peut prendre les valeurs 1, 2, 3 ou 6. Chaque groupe contient alors **K = 12 / C** serveurs. Lorsque **C = 1**, tous les serveurs sont généralistes. À l’inverse, plus C est élevé, plus les groupes sont spécialisés dans une catégorie précise.

Un **routeur central** reçoit l'ensemble des requêtes et les achemine vers le groupe spécialisé approprié. Il agit comme un goulot d’étranglement : il a une **file d’attente limitée**, traite les requêtes en **ordre FIFO**, et peut être **bloqué** si aucun serveur du groupe de destination n’est disponible.

**2.2 - Hypothèses et distributions**

Pour simuler ce système, nous avons adopté les hypothèses probabilistes suivantes :

* **Arrivée des requêtes**
  + Les **inter-arrivées** suivent une **loi exponentielle de paramètre λ**.
  + Chaque requête est assignée à une **catégorie choisie uniformément** parmi les C catégories disponibles.
* **Comportement du routeur :**
  + La **file d’attente du routeur** a une capacité maximale de **100 requêtes**, incluant celle en cours de traitement.
  + Les requêtes sont traitées en **FIFO**.
  + Le **temps de traitement du routeur** dépend du nombre de groupes C :
  + Si la file est pleine à l’arrivée d’une requête, **celle-ci est perdue**.
  + Après traitement, la requête est dirigée vers un **serveur libre** du groupe correspondant.
  + Si aucun serveur du bon groupe n’est disponible, le **routeur est bloqué** jusqu’à libération d’un serveur.
* **Serveurs**
  + Un serveur ne peut traiter qu’une seule requête à la fois et n’a pas de file d’attente.
  + Le temps de service d’un serveur suit une loi exponentielle de paramètre dépendant de C :
  + Les serveurs spécialisés ont un taux de service plus rapide (le traitement est accéléré par un facteur configuré).

**2.3 - Paramètres et contraintes**

La simulation dépend des **paramètres suivants** :

* + **λ (lambda)** : taux d’arrivée des requêtes (ex : de 0.2 à 1.8).
  + **C** : nombre de groupes spécialisés (1, 2, 3, 6).
  + **Durée maximale de simulation** : ex. 1000 unités de temps.
  + **Facteur d'accélération** des serveurs spécialisés (ex. 1.5).

**Contraintes du système :**

* Le **taux de perte** des requêtes **ne doit pas dépasser 5%**.
* Le **temps de réponse** doit être minimisé.
* Une **seule file d’attente (dans le routeur)** est autorisée.
* Les serveurs ne doivent pas introduire de file d’attente individuelle.

**3. Méthodologie**

**3.1 - Outils et langage utilisés**

Les modules principaux sont :

* *random :* génération aléatoire (loi exponentielle).
* *collections.deque :* gestion de la file FIFO du routeur.
* *enum :* définition des types d’événements.
* *math :* calculs numériques (exponentielles, logarithmes).

Le code est modulaire, avec une classe dédiée pour chaque composant : événements, requêtes, routeur, serveurs, groupes de serveurs…  
Une **approche événementielle** a été adoptée afin de simuler le système de manière efficace et contrôlée dans le temps.

**3.2 - Architecture de la simulation**

Le cœur de la simulation repose sur un **échéancier**, qui est une liste d’événements ordonnés dans le temps. Chaque événement correspond à l’un des états suivants :

* Arrivée d’une requête,
* Fin de traitement par le routeur (routeur devient disponible),
* Tentative d’envoi vers un serveur,
* Blocage du routeur (aucun serveur libre).

À chaque étape :

1. L’événement le plus ancien est extrait.
2. Le système est mis à jour en conséquence.
3. De nouveaux événements sont potentiellement générés.

**3.3. Déroulement d’une simulation**

Le déroulement global d'une simulation suit les étapes suivantes :

**I - Initialisation**

* Définition des paramètres : C, λ, durée de simulation, facteur d’accélération des serveurs spécialisés.
* Création du **routeur** avec sa file limitée à 100 requêtes.
* Création des **groupes de serveurs** : chaque groupe contient 12/C serveurs.
* Définition des **catégories traitées** par chaque groupe.
* Génération de la **première requête** et insertion dans l’échéancier.

**II - Boucle de simulation**

Tant que le temps simulé est inférieur à la durée limite, et que le **taux de perte reste inférieur à 5%** :

1. On extrait le **prochain événement** dans l’échéancier.
2. Le **routeur traite** la requête si libre, sinon il attend.
3. Si la requête est traitée :
   * Elle est envoyée à un serveur **du groupe approprié**.
   * Si aucun serveur n’est disponible, le routeur **bloque** jusqu’à libération.
4. Un nouvel événement d'**arrivée de requête** est généré.

**III - Mise à jour**

* Tous les serveurs sont mis à jour à chaque événement : s’ils ont terminé un traitement, ils deviennent disponibles.

**3.4. Critères d’évaluation**

À la fin de la simulation, nous calculons :

* Le **temps de réponse moyen** :
* Le **taux de perte** :

Si le taux de perte dépasse 5%, la configuration est considérée comme **non acceptable**.

**4. Résultats expérimentaux**

**4.1. Temps de réponse moyen**

Le graphique ci-dessous présente l’évolution du **temps de réponse moyen** des requêtes en fonction du **taux d’arrivée λ**, pour les différentes configurations de regroupement de serveurs **C ∈ {1, 2, 3, 6}**.

Une image contenant texte, capture d’écran, diagramme, Tracé

Le contenu généré par l’IA peut être incorrect.

On observe les tendances suivantes :

* Pour toutes les valeurs de λ, plus C est grand, plus le temps de réponse est faible, ce qui montre que la spécialisation améliore la rapidité de traitement.
* Le temps de réponse augmente fortement entre λ = 0.2 et λ = 1, surtout pour C = 1 et C = 2. Cela reflète la saturation progressive du routeur et des serveurs généralistes.
* À partir de λ ≈ 1.5, les temps de réponse se stabilisent, mais restent nettement plus élevés pour les petites valeurs de C.
* Les barres d'erreur (zones colorées) montrent que la variabilité des temps de réponse est plus grande lorsque les serveurs sont peu spécialisés.

**4.2. Taux de perte**

Le graphique ci-dessous montre l’évolution du **taux de perte** des requêtes en fonction de **λ**, pour chaque configuration de spécialisation **C**.

Une image contenant diagramme, ligne, Tracé, texte

Le contenu généré par l’IA peut être incorrect.

Observations principales :

* La configuration **C = 1** (tous les serveurs sont généralistes) **ne perd jamais de requêtes**, même lorsque λ augmente fortement. Cela s'explique par le fait que n’importe quelle requête peut être prise en charge par **n’importe quel serveur libre**, ce qui offre une grande flexibilité au routeur.
* Pour **C = 2, 3, 6**, le taux de perte atteint rapidement la **limite maximale autorisée de 5%**, dès que λ dépasse 0.75–1.0 :
  + **C = 2** reste acceptable jusqu’à environ **λ ≈ 0.9**.
  + **C = 3 et 6** dépassent le seuil dès **λ ≈ 0.75**, et le dépassent légèrement au-delà.
* Le taux de perte reste relativement stable au-dessus du seuil de 5 %, ce qui montre que le système **sature** et **rejette continuellement** des requêtes lorsque la charge devient trop importante.

Ces résultats illustrent le **compromis entre spécialisation et tolérance à la charge** :

* Plus **C est élevé**, plus le système est **rapide**, mais **moins il est tolérant** à un flux important (car chaque groupe a moins de serveurs, donc plus de risques de blocage).
* **C = 1 est robuste**, mais au prix de **temps de réponse plus élevés** (voir 4.1).

**4.3. Détermination des seuils critiques de λ (taux de perte ≥ 5%)**

Pour respecter les contraintes du système, le **taux de perte des requêtes ne doit jamais dépasser 5 %**. Nous avons donc cherché, pour chaque configuration de spécialisation C, la valeur maximale de λ tolérée avant que ce seuil ne soit franchi. Cette étude a été menée à l’aide de graphiques à fort zoom autour du point critique.

**Méthode :**

* Pour chaque valeur de C ∈ {6, 3, 2, 1}, nous avons effectué des simulations avec des **valeurs de λ très proches**, afin de **localiser précisément le point de dépassement des 5 % de pertes**.
* Les résultats sont présentés ci-dessous.

**C = 6**

* La courbe du taux de perte dépasse 5 % dès que **λ ≥ 0.676**.
* Une image contenant texte, capture d’écran, diagramme, Tracé

  Le contenu généré par l’IA peut être incorrect.**Seuil critique** identifié :

**C = 3**

* Le seuil des 5 % est franchi autour de **λ ≈ 0.766**.
* **Seuil critique** :

Une image contenant texte, capture d’écran, diagramme, Tracé

Le contenu généré par l’IA peut être incorrect.

**C = 2**

* La configuration reste acceptable jusqu’à **λ ≈ 0.894**, ensuite le taux dépasse 5 %.
* **Seuil critique** :

Une image contenant texte, capture d’écran, diagramme, Tracé

Le contenu généré par l’IA peut être incorrect.

**C = 1**

* Même à **λ = 4.5**, le taux de perte reste inférieur à 5 %.
* Cela montre que **C = 1 est extrêmement robuste** au niveau du routage.
* Une image contenant texte, diagramme, Tracé

  Le contenu généré par l’IA peut être incorrect.**Conclusion** : aucune perte même sous **très forte charge**.

Cette analyse nous permet de conclure que **plus C est élevé**, **moins le système supporte de charge** avant de devenir instable (en termes de pertes).

**5. Conclusion**

Ce projet de simulation nous a permis d’étudier le comportement d’une ferme de serveurs soumise à un trafic variable, et d’évaluer l’impact de la **spécialisation des serveurs** sur les performances du système.

Grâce à une **modélisation événementielle** précise et une série de **campagnes expérimentales**, nous avons comparé quatre configurations (C = 1, 2, 3, 6) selon deux critères essentiels :

* Le **temps moyen de réponse** des requêtes.
* Le **taux de perte**, qui doit rester inférieur à 5 %.

Nos résultats mettent en évidence un **compromis clair** :

* Les **configurations très spécialisées** (C = 6 ou C = 3) offrent les **meilleurs temps de réponse** tant que la charge est modérée (λ faible).
* À l’inverse, les configurations **moins spécialisées** (C = 1 ou 2) sont plus **tolérantes à des charges élevées**, mais au prix d’un **temps de réponse plus important**.

Nous avons également déterminé avec précision, pour chaque valeur de C, la **valeur maximale de λ** autorisant une perte de requêtes inférieure à 5 %. Cette information est précieuse pour le **dimensionnement du système en fonction de la charge attendue**.

En conclusion :

* Pour des charges de niveau moyen à élevé, il est préférable de diminuer la spécialisation (C = 2 ou même C = 1), afin d'éviter les pertes.
* Pour des systèmes faiblement chargés, il est recommandé d'utiliser C = 6 ou C = 3, afin de bénéficier de temps de réponse très courts.
* La sélection idéale de C est directement liée à λ, ce qui souligne l'importance d'une configuration dynamiquement ajustée à la charge du réseau.

Ce projet nous a finalement donné l'occasion d'appliquer des méthodes sophistiquées de simulation, de perfectionner notre compétence en Python et d'élargir notre connaissance des systèmes répartis et de la gestion des files d'attente.