Rapport de Projet ERO2

Table des matières

1. Introduction

Dans le cadre de ce projet ERO2, nous nous intéressons à la moulinette en tant que système d'attente (ou queuing system). La moulinette est une infrastructure de correction automatique permettant d'exécuter des test-suites sur un code soumis par les étudiants. Plusieurs scénarios de mise en attente et d'exécution sont abordés, afin de:

- **Modéliser** la dynamique d'arrivée et de service (exécution des tests) pour différents contextes (Waterfall, Channels and dams).
- **Analyser** l'impact de la taille des files, du nombre de serveurs, des flux d'arrivées et de la gestion des retours pour l'utilisateur.
- **Discuter** et évaluer différentes configurations d'architecture de la moulinette afin d'optimiser le temps de réponse, le taux de refus, la robustesse, etc.

Ce rapport est construit en plusieurs parties. Nous commençons par la mise en place du modèle et la clarification des hypothèses. Ensuite, nous présentons chacun des cas de figure (Waterfall, Channels and dams) avec leurs simulations, analyses et recommandations.

2. Problématique et modélisation globale

2.1 Contexte général

Les étudiants, issus de différentes populations telles que les filières ING, PREPA, APPING ou Parcours Cyber, utilisent la moulinette pour leurs activités académiques. Chacun d'eux peut tagger son dépôt Git afin de déclencher l'exécution d'une suite de tests. Ainsi, chaque tag push correspond à une nouvelle arrivée dans le système. Les serveurs de la moulinette prennent en charge l'exécution de la suite de tests associée, avant d'envoyer le résultat vers un autre serveur ou une file dédiée. Dans ce processus, chaque trace générée représente une sortie du système.

2.2 Indicateurs de performance

Afin d'évaluer ces différents scénarios, nous utilisons plusieurs **métriques** standard en théorie des files d'attente et en ingénierie système:

1. **Taux de trafic** ρ: rapport du débit moyen d'entrée au débit moyen de service.
2. **Temps de séjour moyen** T_{avg}: moyenne des temps d'attente + service pour un tag.
3. **Taux de blocage** (ou refus): probabilité qu'une arrivée soit rejetée lorsque la file est pleine.
4. **Charge moyenne des serveurs** dans le système ou dans la file d'attente.
5. **Variabilité du temps de séjour** (variance empirique), utile pour estimer la volatilité de l'expérience utilisateur.

2.3 Hypothèses de modélisation

Dans le cadre de cette modélisation, les arrivées dans le système sont généralement représentées comme un processus de Poisson d'intensité λ, sauf indication contraire. Les durées de service, qui correspondent au temps nécessaire pour exécuter la suite de tests, sont modélisées par des variables aléatoires exponentielles caractérisées par un paramètre μ. Le nombre de serveurs impliqués varie en fonction des scénarios envisagés : par exemple, plusieurs serveurs (K) peuvent être alloués pour l'exécution des tests, tandis qu'un serveur unique peut être dédié à l'envoi des résultats. Enfin, les politiques de service adoptées dans ces files sont principalement basées sur le principe du FIFO (First In, First Out), où les tâches sont traitées dans l'ordre de leur arrivée.

2.4 Données de Modélisation

2.4.1 Collecte des données

Recherche et collecte des données sur la moulinette

Afin de mieux comprendre la charge de travail générée par les différents types d'utilisateurs de la moulinette et de modéliser de manière réaliste le fonctionnement du système, une grande partie de cette analyse repose sur des informations collectées par des recherches et des interviews. Ces démarches ont permis d'obtenir des données sur l'utilisation quotidienne de la moulinette et les caractéristiques des tags générés dans différents contextes.

Recherche documentaire

Une revue des documents internes concernant la moulinette a été effectuée, afin de mieux cerner son fonctionnement technique, les processus associés, et les besoins des utilisateurs. Cela a permis de comprendre les aspects clés du système et les facteurs influençant la génération des tags.

2.4.2 Sources des Données

Pour cette analyse, les données utilisées reflètent des scénarios réels ou plausibles, basés sur les volumes typiques d'utilisation de la moulinette par les étudiants dans des contextes spécifiques comme la piscine, les examens, ou les projets. Ces données sont essentielles pour établir des hypothèses réalistes et simuler les performances des files d'attente dans divers cas d'utilisation.

Nous avons interviewé la Forge pour obtenir certaines métriques et avoir des jeux de tests les plus proches de la réalité.

Nombre de serveurs:

Le nombre de serveurs de moullinetage de la Forge (aussi appelé runner) est statique, il n'est pas changé dynamiquement en fonction de la charge, néanmoins son nombre peut être adapté en fonction du planning des activités prévus par l'école (ateliers, examens, etc…). Il y a en période d'atelier ou d'examen 20 runners, le reste du temps la moitié. Il faut néanmoins noter une grosse différence entre notre simulation et la réalité: un runner Forge peut exécuter plusieurs tags en même temps en fonction des ressources physiques disponibles (CPU, mémoire), mais dans le cas de notre simulation, un serveur pourra uniquement faire tourner un job à la fois.

Piscine:
Pendant une session de piscine, les étudiants travaillent sur des exercices intensifs, entraînant un volume élevé de tags. En moyenne, 639 étudiants, chacun réalisant environ 100 exercices, et chaque exercice générant 3 tags. Cela donne 191 700 (~192 000 selon les vraies données de la Forge) tags générés sur une période de 12 jours (à raison de 16 heures par jour). En termes de débit, cela correspond à 998.4 tags par heure ou environ 16.6 tags par minute (9-10 tags d'après les données de la Forge). Le code C étant relativement léger les testsuites utilisées par l'équipe des ACUs ne prend qu'en moyenne 1 minute pour s'exécuter. Ces chiffres reflètent une charge élevée et constante, idéale pour tester la capacité des serveurs de la moulinette à absorber une pression continue.

Cependant, un aspect important à considérer est l'augmentation du débit lors de l'ouverture des tags. En effet, dans cette phase, le flux de tags peut atteindre un débit beaucoup plus élevé, pouvant aller jusqu'à 100 tags par minute, en raison de l'afflux simultané de nombreux étudiants à l'ouverture des tags. Cette période de pic représente un challenge supplémentaire pour tester la résilience du système et sa capacité à gérer des périodes de forte affluence au démarrage.

SUP:
Pour les étudiants en SUP, on estime qu'ils généreront des tags pendant une période active de travail, en excluant le temps de sommeil et les cours. En moyenne, 824 étudiants génèrent 4 tags par semaine, avec une estimation d'environ 12 heures par jour de travail actif menant à un total de 84 heures. Cela donne un total de 3296 tags (sur un des tps pris au hasard par la Forge on est aux alentours de 3000 tags) générés sur ces 84 heures, ce qui correspond à un débit moyen de 39 tags par heure ou environ 0.65 tag par minute. Ce volume d'activité permet d'évaluer le système dans un contexte de charge modéré. De plus, les tags liés au langage C# étant plus complexes, leur traitement peut prendre plus de temps, estimé à environ 5 minutes par tag, ce qui représente un défi supplémentaire pour les serveurs. Ces informations permettent de simuler des scénarios réalistes dans un environnement où les durées de service peuvent varier en fonction des technologies utilisées.

2.4.3 Justification des Données

Pour le taux de service de la deuxième file (μ_2), une valeur de 20 a été choisie, correspondant à un débit de 20 uploads par minute. Cette valeur a été choisie de manière arbitraire, mais permet de modéliser un système de traitement avec un taux de service relativement élevé, tout en restant cohérente avec l'objectif de tester la capacité du système sous des conditions de charge variées.

Ces données ont été choisies pour plusieurs raisons :
Pertinence : Elles reflètent les comportements typiques des étudiants et les configurations courantes du système dans différents contextes.
Variabilité : Elles permettent de couvrir une large gamme de conditions, des charges continues (piscine) aux pics soudains (examens).
Applicabilité : Elles sont conformes aux hypothèses de modélisation, comme le processus de Poisson pour les arrivées et la distribution exponentielle pour les temps de service.
Évolutivité : Elles facilitent l'étude de scénarios de surcharge (par exemple, augmentation du nombre de tags par étudiant ou réduction de la capacité des serveurs).

Ces données serviront à calibrer et valider les modèles MM1, MM1k, et MMKK développés dans cette étude, ainsi qu'à analyser des indicateurs de performance clés comme le temps de séjour moyen, le taux de blocage, et la charge moyenne des serveurs dans le système.

Dans la gestion de la charge des serveurs, il est important de maintenir un équilibre entre un taux d'utilisation suffisamment élevé pour maximiser l'efficacité, mais également suffisamment bas pour éviter les risques de surcharge. À cet égard, nous avons cherché à maintenir la charge des serveurs en dessous de 80% en permanence. Cette limite permet d'éviter de

solliciter les serveurs de manière excessive, ce qui pourrait entraîner des pannes ou des dégradations de performances dues à une surcharge prolongée. De plus, en restant en dessous de ce seuil, nous pouvons anticiper et mieux gérer les pics de charge, garantissant ainsi une réponse réactive et une meilleure fiabilité du système.

3. Partie 1 - Waterfall

3.1 Description du modèle et hypothèses

Le modèle Waterfall (ou “en cascade”) est décrit dans l’énoncé:

Première file d’attente (exécution des tests)
Un push tag place le code dans une file **FIFO** de taille potentiellement infinie (dans un premier temps).
Il existe **K serveurs** pour exécuter la test-suite, chacun pouvant gérer un tag à la fois.

Seconde file d’attente (renvoi des résultats)
Une fois les tests exécutés, le résultat est placé dans une **file FIFO** gérée par un **unique** serveur qui transmet ces résultats au front (interface utilisateur).

Ce modèle se rapproche d’une file M/M/c suivie d’une file M/M/1. On étudie d’abord le cas où les deux files sont infinies (pas de refus), puis le cas où elles ont une capacité finie (ks,kf).

3.2 Simulation et analyse des performances (files infinies)

3.2.1 Paramètres en jeu

- λ: taux d’arrivée (nombre de push tags par unité de temps).
- μ1: taux de service de **chaque** serveur d’exécution.
- **K**: nombre de serveurs parallèles pour l’exécution.
- μ2: taux de service du serveur de renvoi des résultats.

3.2.2 Comportement du système et stabilité

- **Stabilité globale**:
    - Première file (M/M/K) : pour être stable, on nécessite $\lambda < K\mu 1$.
    - Seconde file (M/M/1) : elle reçoit un flux moyen de λ (identique au flux d’entrée, car tout tag passé à la 1ere file arrivera dans la 2e). Pour la stabilité, on a $\lambda<\mu 2$.
- **Condition de stabilité globale**: $\lambda<\min(K\mu 1,\mu 2)$.

Si la condition n’est pas remplie, la file s’allonge indéfiniment, provoquant des temps de séjour explosifs.

Prenant en compte le cas d’exemple de la Piscine avec:
- $\lambda = 16.6$
- $\mu 1 = 1$
- $\mu 2 = 20$

Pour pouvoir résoudre ce cas il faudrait donc:
- $\lambda< \mu 2$ soit $16.6 < 20$

- λ<Kμ1 soit K > (λ/μ1)

Sachant qu'on a bien λ< μ2 la seule variable reste K > 16.6 soit K supérieur ou égal à 17.
En choisissant de garder une charge moyenne des serveurs à 80% il faudrait λ<min(Kμ1,μ2)*80% soit:

- λ< μ2*0.8 soit 16.6 < 16
- λ<Kμ1*0.8 soit K > (λ/(μ1*0.8)) ou K > 20.75

Dans ce cas on voit bien que l'on arrive à la limite de la file de renvoi des résultats mais avec approximativement 20 serveurs, cela nous permettrai de rester aux alentours de 80% de charge.

Concernant la charge en cas d'ouverture des tags, les résultats sont bien différents. Afin d'avoir suffisamment de serveurs il faudrait λ<min(Kμ1,μ2)*80% soit:

- λ< μ2*0.8 soit 100 < 16
- λ<Kμ1*0.8 soit K > (λ/(μ1*0.8)) ou K > 125

Étant déjà dans l'impossibilité de pouvoir gérer le flux dans la partie envoi des résultats, il n'est même pas nécessaire de s'attarder sur le calcul du nombre de serveurs nécessaires.

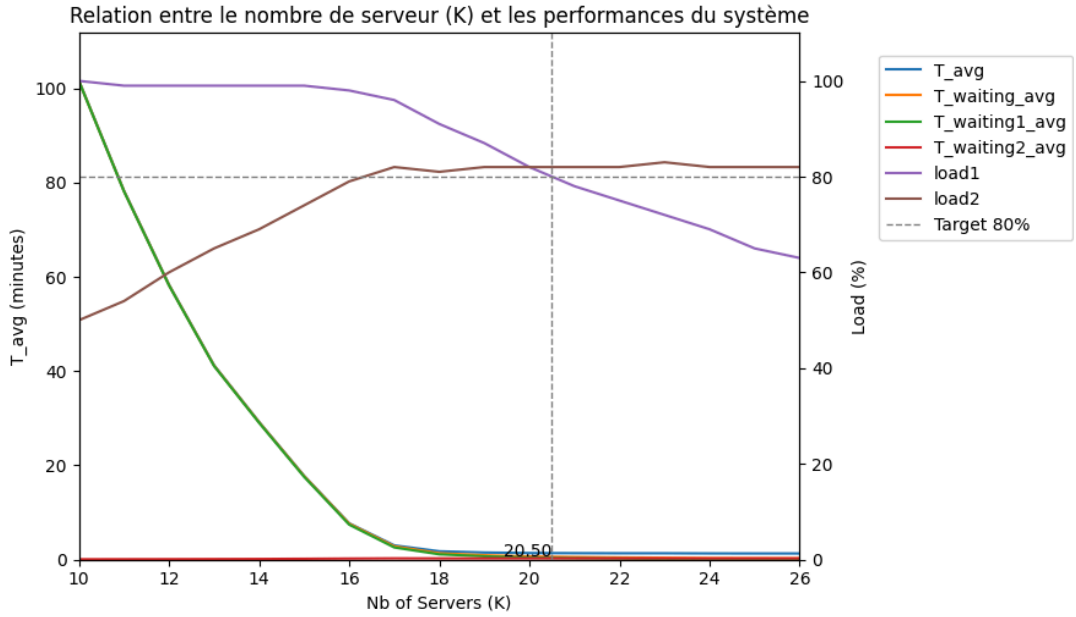
3.2.3 Évaluation selon des métriques standard

- **Temps de séjour moyen (empirique)** : noté $\bar{T}$. Il inclut:
    1. Le temps d'attente + service dans la 1ere file.
    2. Le temps d'attente + service dans la 2e file.
- **Charge moyenne des serveurs** dans le système ou dans la file d'attente.
- **Taux de blocage** : nul (puisque file infinie).

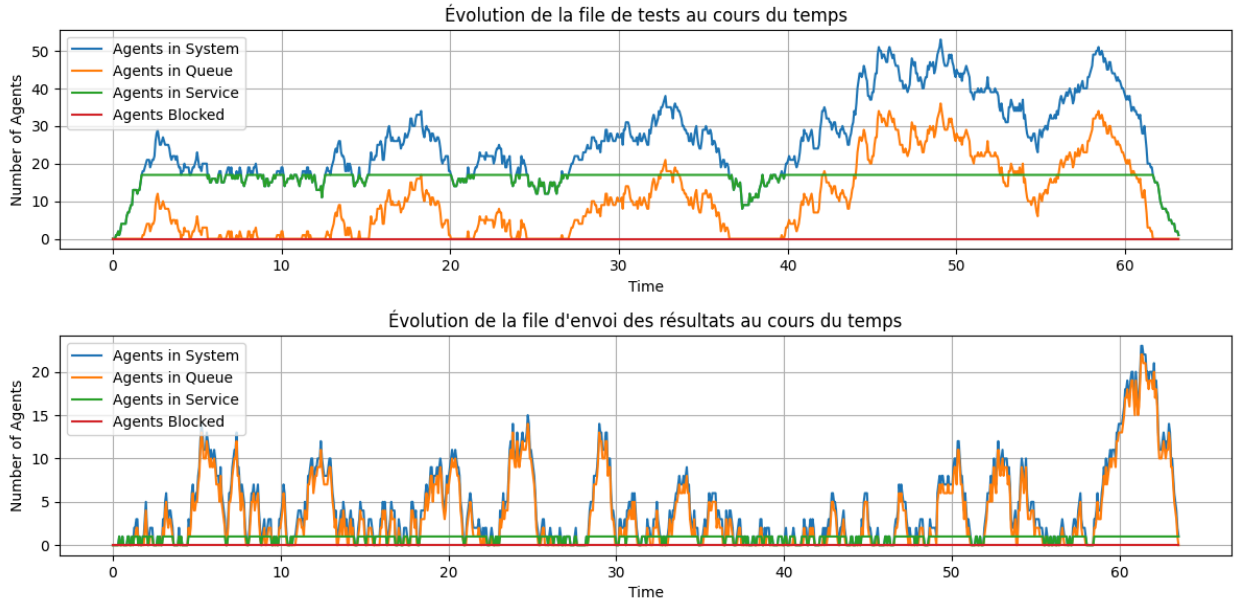
3.2.4 Interprétation des résultats

Pour évaluer les performances de différentes configurations de systèmes à files infinies, nous avons étudié plusieurs scénarios caractérisés par des taux d'arrivée (λ), des nombres de serveurs (K) et des taux de service (μ_1,μ_2).

Configurations pertinentes et leurs performances

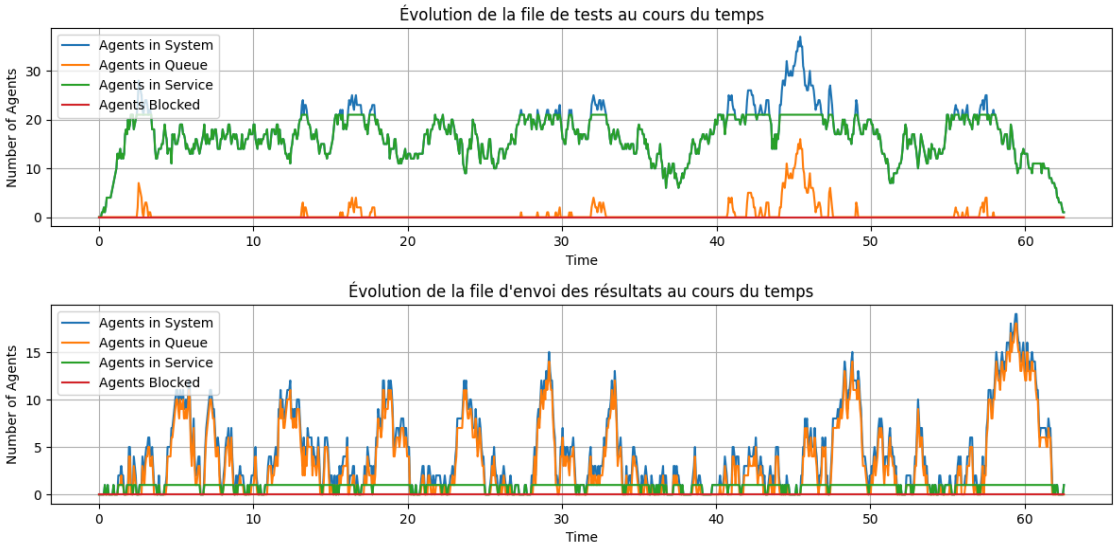


La configuration **Infinite_Sans_Backup_Piscine_K_10to26** est exemple pour montrer des charges utilisant entre 10 et 26 serveurs. Les résultats montrent que pour obtenir des temps moyens en diminuant le temps de séjour tout en restant aux alentours des 80% de charge, un nombre de serveurs aux alentours de 20.50 est nécessaire. Les résultats de cette simulation montrent une cohérence avec les résultats théoriques précédemment calculés.

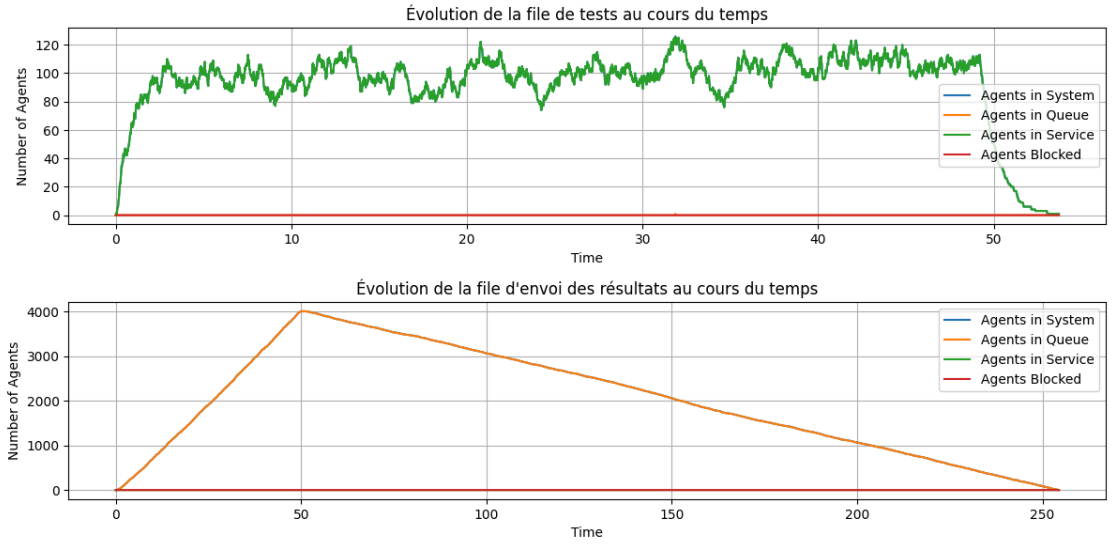


La configuration **Infinite_Sans_Backup_Piscine_K_17** est exemple pour faire tourner les tags en utilisant 17 serveurs (K=17). Les résultats montrent des temps moyens courts (T_{avg}=2.69) et

une variance faible (T_{var}=3.33). Le taux de charge du serveur 1 (0.96) montre un sous-dimensionnement de la configuration pour ne pas surcharger les machines.



La configuration **Infinite_Sans_Backup_Piscine_K_21** est exemple pour faire tourner les tags en utilisant 21 serveurs (K=21). Les résultats montrent des temps moyens extrêmement courts (T_{avg}=1.34) et une variance faible (T_{var}=1.11), garantissant une stabilité parfaite. Le taux de charge du serveur 1 (0.78) montre un bon dimensionnement de la configuration.



La configuration **Infinite_Sans_Backup_Ouverture_Piscine** est un exemple pour faire tourner les tags lors de leur ouverture en début de piscine en utilisant 125 serveurs (K=100) pour un débit de 100 tags par minute (λ=100). Les résultats montrent des temps moyens extrêmement longs (T_{avg}=100) et une variance élevée (T_{var}=3362.25), montrant de gros problèmes de temps.

3.3 Simulation et analyse des performances (files finies)

3.3.1 Hypothèses de file finie

- Taille de la 1re file : ks.
- Taille de la 2e file : kf.
- **Politique de refus** : si la 1re file est pleine, le push tag est rejeté → l'étudiant reçoit un message d'erreur.
- **Politique de refus** dans la 2e file : si la 2e file est pleine, le résultat du moulinette est perdu → l'étudiant reçoit un *retour vide*.

3.3.2 Proportions de refus

En simulation, on observe deux taux distincts:

1. **Taux de refus en entrée** : αs, proportion de tags arrivants et refusés (1ere file saturée).
2. **Taux de pages blanches (ou retours vides)** : αf, proportion de résultats perdus car la 2e file est saturée.

Le comportement dépend de λ, μ1, μ2, ks, kf, et K:

- Si ks est trop petit et λ élevé, αs sera significatif, entraînant frustration chez l'utilisateur (messages d'erreur).
- Si ks est suffisamment grand mais kf est restreint, alors αf sera plus important (nombreux retours vides).

N'ayant pas trouvé de formule théorique pouvant résoudre ce problème, aucune valeur théorique ne pourra être visée. La seule formule trouvée est la formule de la loi d'Erlang fonctionnant pour des files M/M/S/S avec comme seule capacité d'attente les serveurs eux-même, ce cas ne représente donc pas le nôtre contenant aussi une attente sans serveurs.

3.3.3 Expérience utilisateur et impacts

- Un taux élevé de αs est grave sans pour autant être critique : l'étudiant ne peut pas soumettre son code, mais il est prévenu plus tard
- Un taux élevé de αf est beaucoup plus critique : l'étudiant suppose que son tag est passé mais se rend compte que potentiellement beaucoup plus tard que le tag n'est pas passé.

3.3.4 Objectifs

Puisque l'envoi d'un tag à lancer ne nécessite que de l'identifier dans la file pour l'appeler au moment approprié (comme les numéros d'attente dans les services publics), il est facile

d'imaginer que plusieurs centaines de tags puissent être stockés simultanément.

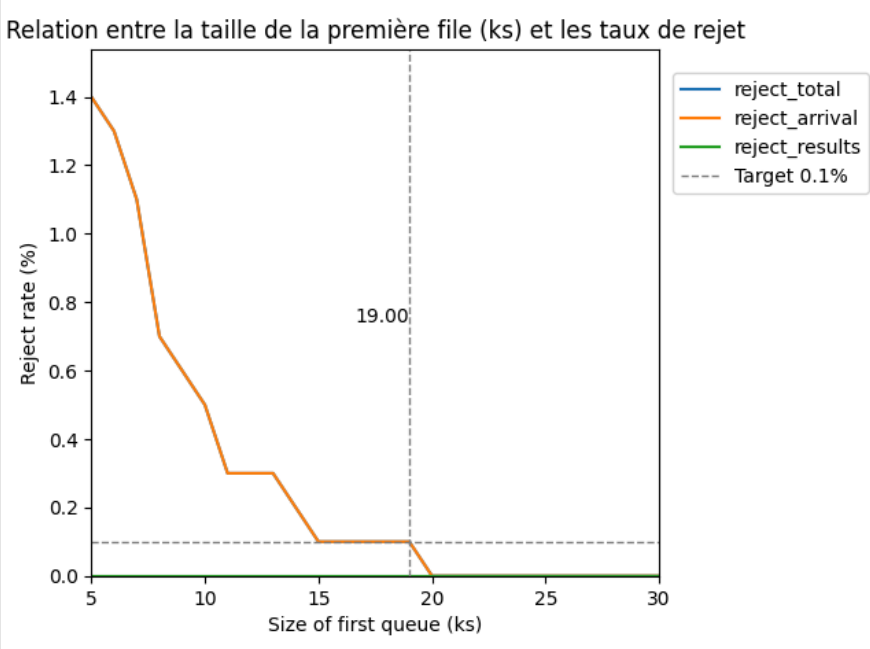
Afin d'éviter au plus ces cas, nous pensons qu'une valeur de αs inférieure à 0.1% serait cohérente avec nos attentes.
Concernant αf, étant selon nous plus critique, avoir une valeur inférieure à 0.05% serait acceptable.

3.3.5 Interprétation des résultats

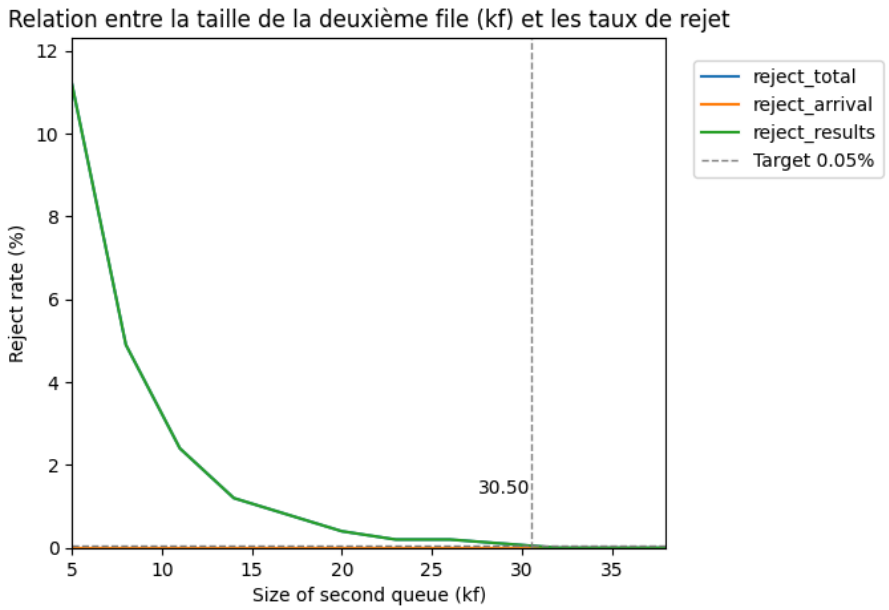
Analyse des configurations et des résultats

Dans le cadre de cette étude, plusieurs configurations de systèmes à **files limitées** ont été analysées pour évaluer leur performance en fonction de différents paramètres, tels que le taux d'arrivée (λ), le nombre de serveurs (K), les capacités des files (max_k_s,max_k_f), et les taux de service (μ1,μ2).

Configurations les plus pertinentes et leurs performances

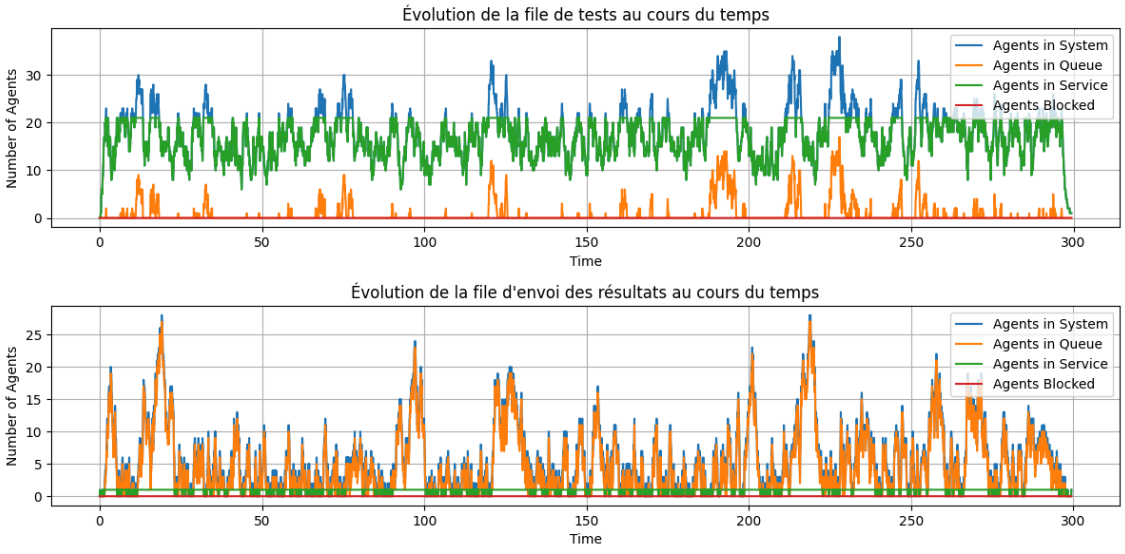


La configuration **Finite_Sans_Backup_Piscine_Ks_5to30** nous permet de simuler le taux de rejet sur la file d'entrée, ce benchmark faisant une moyenne sur 30 cas différent pour chaque valeur de ks voulu nous permet de suggérer une taille de file à peu près 20 afin d'avoir une probabilité de rejet (αs) en dessous des 0.1%. Ce résultat est bien inférieur à la limite fixée en amont et est donc une valeur parfaite pour nos besoins.



La configuration **Finite_Sans_Backup_Piscine_Kf_5to40** nous permet de simuler le taux de rejet sur la file des résultats, ce benchmark faisant une moyenne sur 30 cas différent pour chaque valeur de kf voulu nous permet de suggérer une taille de file à peu près 30 afin d'avoir une probabilité de rejet (αf) en dessous des 0.5%. Ce résultat est bien inférieur à la limite fixée en amont et est donc une valeur parfaite pour nos besoins.

Analyse globale



Les résultats montrent que les performances des systèmes dépendent fortement du dimensionnement des serveurs, des taux de service, et des capacités des files. Les configurations bien dimensionnées, comme **Finite_Sans_Backup_Piscine** établie grâce aux

résultats des configurations **Finite_Sans_Backup_Piscine_Ks_5to30** et **Finite_Sans_Backup_Piscine_Kf_5to40** , offre des performances optimales, avec des temps moyens courts, des rejets limités, et une stabilité élevée. Ces systèmes équilibrent efficacement le flux d'arrivées (λ) avec les capacités de traitement des serveurs (K,μ1,μ2) et des files (max_k_s,max_k_f).

3.4 Mise en place d'un back-up en amont de la seconde file

Pour éviter les pertes de résultats (lorsque la 2e file est pleine), un **système de back-up** stocke temporairement les résultats sur un serveur ou une base de données avant leur envoi:

1. **Impact sur la proportion de pages blanches** :
    - Le back-up permet de re-tenter l'envoi plus tard, donc αf baisse drastiquement. On peut potentiellement atteindre αf ≈ 0.
2. **Problèmes possibles** :
    - Si le back-up n'est pas dimensionné correctement ou si son flush rate n'est pas plus rapide que λ, il peut lui-même saturer.
    - Le coût de stockage/transfert peut augmenter (ressources disque, complexité de code, etc.).
3. **Avantages d'un back-up aléatoire** :
    - Choisir aléatoirement quels résultats sont sauvegardés peut limiter la surcharge côté stockage.
    - Toutefois, on introduit de l'incertitude sur la cohérence des retours pour l'utilisateur (certains seront perdus, d'autres sauvegardés).

3.4.1 Problèmes d'implémentations

L'implémentation d'un système de back-up en amont de l'envoi vers la seconde file pourrait permettre d'assurer leur envoi en cas de surcharge. Cependant, étant donné que la taille de la file peut augmenter considérablement, l'utilisation d'une telle back-up reviendrait tout simplement à augmenter la taille de la file, ne changeant donc rien au problème initial, si μ2 n'est pas assez grand, la seconde file ne pourra jamais rattraper son retard et afficher les résultats qu'elle aurait perdu.

3.5 Comparaison files finies et infinies

Dans cette étude, nous avons comparé les performances de systèmes à **files finies** et **files infinies** en analysant plusieurs paramètres, notamment les taux de rejet (αs,αf), les temps moyens dans le système (T_{avg}), et la variance des temps d'attente (T_{var}).

Pour des files infinies, nous n'avons eu aucune perte de tag ni d'autres problèmes. Les personnes ne restent pas longtemps dans la queue car il y a un nombre suffisant de serveurs.

Pour des files finies, le temps de traitement est proche des files infinies, le tout avec peu d'erreurs.

4. Partie 2 - Channels and dams

4.1 Variations de temps de séjour par population

Dans l'énoncé, on observe que certaines populations (ex. : ING) arrivent plus fréquemment mais ont des temps de traitement potentiellement plus courts (rendu plus rapide a moulinetter), tandis que d'autres (ex. : PREPA) arrivent moins souvent mais peuvent occuper la moulinette plus longtemps (code plus long a executer).

- On paramètre alors deux flux distincts : λING et λPREPA.
- Les durées de service moyennes peuvent différer : μING vs. μPREPA.

La simulation (voir Annexe A) montre des **temps de séjour hétérogènes** :

- La population la plus fréquente saturera rapidement le système, pénalisant tout le monde.
- Quand PREPA a un service plus long, tout push sur PREPA monopolise un serveur plus longtemps.

Représentation dans un modèle

On peut distinguer deux classes d'emplois (multi-classe) dans un système M/M/c. Les formules analytiques précises (voir files **M/M/c** multi-classe) sont plus complexes, d'où l'intérêt de la simulation.

4.2 Mise en parallèle de la population ING1 et SUP

4.2.1 Paramètres en jeu

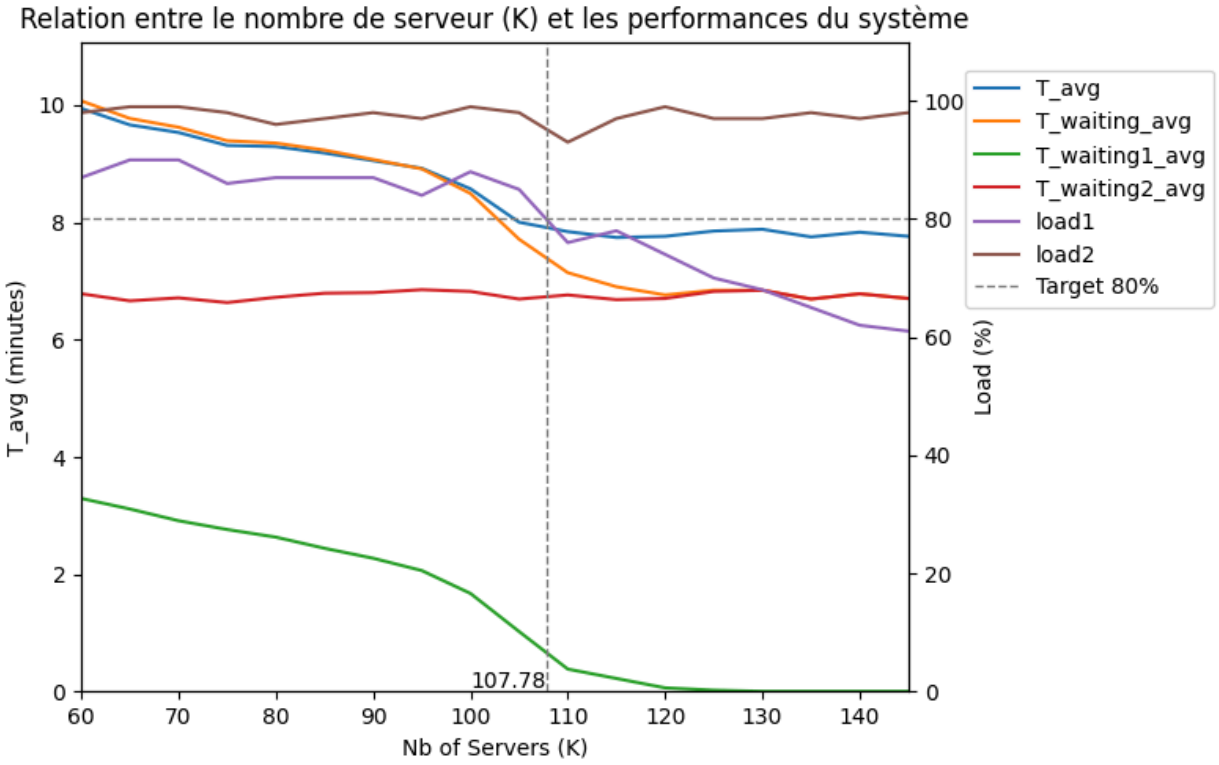
- λ: taux d'arrivée. 100 pour les ING1 et 0.65 pour les SUP
- μ1: taux de service de **chaque** serveur d'exécution. 1 pour les ING1 et 0.2 pour les SUP
- **K**: nombre de serveurs parallèles pour l'exécution.
- μ2: taux de service du serveur de renvoi des résultats.
- ks: taille de la 1re file.
- kf: taille de la 2e file.

4.2.2 Estimation des résultats

Ayant trouvé comme montré précédemment la population ING1 nécessite 21 serveurs (K = 21) et respectivement 20 et 30 places dans les queues (ks=20, kf=30).
Avec les mêmes calculs la population des SUP, elle, nécessite 4 serveurs (K = 4), 15 et 1 places dans les files d'attente (ks=15, kf=1).

Une estimation naïve serait de penser que superposer les deux permettrait d'être une solution correcte, faisant donc un total de 28 serveurs (K=24) et respectivement 35 et 30 places dans les files d'attente (ks=35, kf=31).

4.2.3 Interprétation des résultats



D'après ce graph, on voit bien que le nombre estimé des 28 serveurs est bien loin des résultats obtenus de 108 serveurs.

4.2.4 Proposition d'un autre système d'attente

Pour équilibrer les temps de séjour, on peut :

- **Réserver des serveurs pour chaque population** : les populations Ing1 et SUP fonctionnent comme si séparées avec des serveurs pour réservés, permet de considérablement réduire le nombre de serveur nécessaires
- **Allouer des priorités** : par exemple allouer des priorités sur les tags courts et en cas d'attente, un tag plus long peut être lancé).
- **Implémenter un scheduling** type round-robin, ou Weighted Fair Queuing, etc.
- **Réguler** le nombre maximum de soumissions pour ING par heure, tout en autorisant PREPA à soumettre plus rarement mais sans blocage.

5. Synthèse des résultats et recommandations

- **Synthèse des résultats**
    - **Files infinies** : Absence de rejets, adaptées aux charges modérées, mais instables sous forte charge.
    - **Files finies** : Rejets limitent la saturation, adaptées aux charges élevées, nécessitent un dimensionnement précis.
    - **Populations hétérogènes** : Saturation rapide par populations fréquentes (ex. ING1), équilibre crucial pour éviter des pénalités sur les autres.
- **Recommandations**
    - **Dimensionnement :**
        - Files infinies : Serveurs dimensionnés à 70-80% de charge pour charges modérées.
        - Files finies : Capacités ajustées pour $\alpha s < 0.1\%$ et $\alpha f < 0.05\%$.
    - **Back-up** : Réduire les pertes critiques dans les files finies.
    - **Gestion des populations** : Réserver/prioriser les serveurs ou équilibrer via des politiques adaptées (round-robin, quotas).
    - **Surveillance** : Ajuster les configurations en fonction des métriques clés.

6. Conclusion

Le projet ERO2 nous a permis d'étudier la **moulinette** sous l'angle des **systèmes d'attente**. Nous avons :

1. Proposé des **modèles de queueing** (M/M/K, M/M/1, multi-classes) pour représenter l'exécution et le renvoi des résultats.
2. **Implémenté** des simulations en Python (code en Annexe).
3. **Mesuré** et **analysé** différentes configurations (files infinies, files finies, back-up, etc.).
4. Tiré des **recommandations** pour améliorer l'expérience utilisateur et la robustesse du système.

En conclusion, ce projet nous a permis de combiner l'analyse théorique et la simulation pour évaluer des solutions concrètes. En intégrant également des considérations pratiques, comme les contraintes organisationnelles et les attentes des utilisateurs, nous avons pu formuler des recommandations directement applicables pour optimiser les performances du système et améliorer son expérience utilisateur.

7. Annexes

Annexe A: Code de simulation Python

Commentaire :

- Le code utilise un *Event-driven simulation approach* basique.
- On différencie les événements *arrival*, *departure1*, *departure2*, et *reinsertion* (pour le back-up).
- Les sections clés sont commentées.
- Le résultat final donne un dictionnaire avec le nombre de refus, le nombre de résultats perdus, le temps de séjour moyen et sa variance, etc.
- Une autre version nous permet d'afficher le résultat d'une simulation précise en créant un graphique de la charge en fonction du temps
- Deux autres fichiers de codes permettent de créer des graphiques d'illustration à partir des fichiers d'output de la première version.

Annexe B: Extraits de résultats bruts de simulation

1. Description des Paramètres et Résultats

- **λ** : Taux d'arrivée (push tags par unité de temps).
- **K**: Nombre de serveurs parallèles pour l'exécution des tests.
- **μ1**: Taux de service de chaque serveur d'exécution (push tags traités par unité de temps).
- **μ2**: Taux de service du serveur de renvoi des résultats.
- **max(ks)**: Capacité maximale de la première file (exécution des tests).
- **max(kf)**: Capacité maximale de la seconde file (renvoi des résultats).
- **Back-up**: Indique si un système de back-up est en place (Oui/Non).
- **Taux de refus (αs)** : Pourcentage de push tags refusés en raison d'une file d'attente pleine.
- **Taux de pages blanches (αf)** : Pourcentage de résultats perdus lorsque la seconde file est pleine.
- **Temps de séjour moyen (**T_{avg}**)** : Temps moyen qu'un push tag passe dans le système (en unités de temps).
- **Variance(**T_{var}**)**: Variance empirique du temps de séjour

2. Résultats par Configuration

Toutes les données brutes sont disponibles dans le dossier "results/".