Rapport de Projet : Recherche Opérationnelle 2

Optimisation d'une file d'attente pour moulinette de tests

Lesech Erwann
Aymeric Le Riboter
Abel Aubron
Nathan Claude
Victor Mandelaire
Florian Ruiz

10 janvier 2026

`https://github.com/ErwannLesech/Recherche-Operationnelle-2-File-d-attente`

Introduction

Ce projet propose une analyse approfondie du système de correction automatique de l'EPITA, communément appelé *moulinette*, sous l'angle des systèmes d'attente. La moulinette est une infrastructure logicielle permettant l'exécution de tests unitaires sur du code soumis par les étudiants, fournissant ainsi un retour automatisé sur la conformité du code aux spécifications attendues.

Contexte et motivation

Dans le cadre des activités pédagogiques à l'EPITA, les étudiants soumettent régulièrement leur code pour évaluation automatique via un système de tags git. Ces soumissions déclenchent l'exécution de test-suites qui peuvent varier considérablement en complexité et en temps

d'exécution. Face à une population étudiante hétérogène (Prépa SUP/SPÉ, Ing1) avec des comportements de soumission différenciés, le dimensionnement et l'optimisation de cette infrastructure constituent un enjeu majeur tant sur le plan technique qu'économique.

Les questions centrales auxquelles ce rapport répond sont :

- Comment modéliser efficacement le flux de soumissions et de traitement ?
- Quel impact ont les limitations de capacité (buffer, serveurs) sur les performances ?
- Comment gérer les différentes populations d'utilisateurs avec des patterns d'utilisation distincts ?
- Quelle configuration optimale permet de minimiser les coûts tout en garantissant une qualité de service acceptable ?

Approche méthodologique

Notre approche s'articule autour de trois axes complémentaires :

1. Modélisation théorique : Nous utilisons la théorie des files d'attente pour modéliser le système, en s'appuyant sur la notation de Kendall A/S/c/K/N/D. Les modèles implémentés incluent M/M/1, M/M/c, M/M/c/K, M/D/1 et M/G/1, permettant de capturer différents aspects du système.

2. Simulation Monte Carlo : Pour valider les modèles théoriques et explorer des scénarios complexes, nous avons développé un simulateur événementiel discret. Chaque simulation est répétée 10 fois avec des graines aléatoires différentes pour estimer les moyennes et variances empiriques.

3. Optimisation multi-critères : Nous formulons le problème de dimensionnement comme une optimisation coût/qualité de service, avec une fonction objectif de la forme :

$$\min_{K,c,\mu} [\alpha \cdot \mathbb{E}[T] + \beta \cdot C(K, c, \mu)] \quad \text{avec} \quad \alpha + \beta = 1 \tag{1}$$

où $\mathbb{E}[T]$ représente le temps moyen de séjour et C le coût total incluant serveurs, rejets et insatisfaction. Les échanges menés avec ces étudiants ont permis de recueillir l'ensemble des informations nécessaires à la bonne tenue de ce projet. Ces données sont détaillées en annexes et synthétisées dans la sous-section 7.

Structure du rapport

Ce rapport s'organise comme suit :

1. **Modèle Waterfall :** Analyse du système en cascade avec files infinies puis finies
2. **Gestion des capacités finies :** Étude des taux de blocage et de l'impact du buffer
3. **Mécanismes de backup :** Discussion sur la sauvegarde des résultats et son impact
4. **Channels and Dams :** Modélisation des populations différenciées avec mécanismes de régulation
5. **Optimisation :** Recommandations sur le dimensionnement optimal
6. **Conclusion :** Synthèse des résultats et perspectives

Table des matières

1 Modélisation du système Waterfall

1.1 Description du système

Le modèle Waterfall représente l'architecture en cascade de la moulinette, composée de deux étapes séquentielles :

1. **File d'exécution :** Les soumissions entrent dans une file FIFO desservie par K serveurs (runners) qui exécutent les test-suites
2. **File de renvoi :** Les résultats sont placés dans une seconde file FIFO gérée par un serveur unique pour l'envoi vers le frontend

```
Étudiants → Buffer 1 (K) → Runners (c) → Buffer 2 (K) → Frontend (1 srv)
```

1.2 Modèle avec files infinies

Dans un premier temps, nous considérons des capacités infinies ($K_1 = K_2 = \infty$).

1.2.1 File 1 : Exécution des tests (M/M/c)

La première file est modélisée par une file M/M/c avec :

— Taux d'arrivée : λ (tags/heure)
— Taux de service : μ par serveur (exécutions/heure)
— Nombre de serveurs : c

Condition de stabilité :

$$\rho = \frac{\lambda}{c\mu} < 1 \Leftrightarrow c > \frac{\lambda}{\mu}$$

Métriques clés :

Le nombre moyen de clients en file d'attente :

$$L_q = C(c, a) \cdot \frac{\rho}{1-\rho} \text{où} \quad a = \frac{\lambda}{\mu}$$

Le temps moyen d'attente (formule de Little) :

$$W_q = \frac{L_q}{\lambda} \quad \text{et} \quad W = W_q + \frac{1}{\mu}$$

1.2.2 File 2 : Renvoi des résultats (M/M/1 ou M/D/1)

Le taux d'arrivée en file 2 est égal au taux de service effectif de la file 1 :

$$\lambda_2 = \min(\lambda, c\mu)$$

Pour un modèle M/M/1 :

$$\rho_2 = \frac{\lambda_2}{\mu_2} < 1$$

$$W_2 = \frac{1}{\mu_2 - \lambda_2}$$

Pour un modèle M/D/1 (service déterministe) :

$$W_{q,2} = \frac{\rho_2}{2\mu_2(1-\rho_2)} \quad \text{(réduit de 50\% vs M/M/1)}$$

Temps de séjour total :

$$W_{total} = W_1 + W_2$$

2 Système Waterfall avec capacités finies

2.1 Modélisation M/M/c/K

Avec l'augmentation du nombre d'étudiants, l'hypothèse de capacités infinies devient irréaliste. Nous introduisons :

— K_1 : Capacité maximale de la file d'exécution (buffer + serveurs)
— K_2 : Capacité maximale de la file de renvoi

2.1.1 File 1 : M/M/c/K_1

Le système est désormais **toujours stable** (les arrivées excédentaires sont rejetées).

Distribution stationnaire :

$$\pi_n = \begin{cases} \frac{a^n}{n!}\pi_0 & \text{si } n \leq c \\ \frac{a^c}{c!}\rho^{n-c}\pi_0 & \text{si } c < n \leq K_1 \end{cases}$$

où π_0 est la constante de normalisation.

Taux de blocage (rejet) :

$$P_{K_1} = \pi_{K_1}$$

Un étudiant voit sa soumission rejetée avec probabilité P_{K_1}, recevant un message d'erreur.

Taux effectif :

$$\lambda_{eff} = \lambda(1 - P_{K_1})$$

2.1.2 File 2 : M/M/1/K_2

La file de renvoi reçoit les résultats avec taux $\lambda_2 = \lambda_{eff}$.

Probabilité de perte de résultat :

$$P_{K_2} = \text{Prob(file 2 pleine)} = \pi_{K_2}^{(2)}$$

Un étudiant reçoit une page blanche (résultat perdu) avec probabilité :

$$P_{blank} = (1 - P_{K_1}) \cdot P_{K_2}$$

3 Mécanismes de backup et temps de séjour

3.1 Backup systématique des résultats

Pour éviter la perte de données (pages blanches), un mécanisme de backup est introduit **avant** l'insertion dans la file 2.

3.1.1 Architecture modifiée

```
File 1 → Runners → [BACKUP] → File 2 → Frontend
                       ↓
                   Stockage
```

Impact sur les pages blanches :

$$P_{blank} = 0$$

Les résultats ne sont plus perdus car sauvegardés avant l'insertion en file 2. Si la file 2 est pleine, le résultat est en backup et peut être récupéré ultérieurement.

3.2 Problèmes induits

Augmentation de la charge : Le backup ajoute un temps de service. Si μ_{backup} est lent (I/O disque), le débit d'entré de la file 2 diminue.

3.3 Backup aléatoire

Alternative : backup probabiliste avec probabilité p :

$$P_{blank,random} = (1 - P_{K_1}) \cdot P_{K_2} \cdot (1 - p)$$

Avantages :

- Réduction de $1 - p$ de la charge de backup
- Moins de stockage nécessaire
- Acceptable si P_{K_2} est très faible

Exemple : Pour $P_{K_2} = 0.1\%$ et $p = 0.5$:

$$P_{blank,random} = 0.001 \times 0.5 = 0.05\% \quad \text{vs} \quad 0.1\%$$

3.4 Calcul du temps de séjour moyen et variance

3.4.1 Temps de séjour total

Le temps total est la somme :

$$T = T_1 + T_{backup} + T_2$$

où :

- T_1 : Temps dans file 1 (attente + service)
- T_{backup} : Temps de backup
- T_2 : Temps dans file 2

4 Channels and Dams : Populations différenciées

Personae et patterns d'utilisation

Un élément clé de notre analyse réside dans la modélisation réaliste des comportements étudiants. Nous avons défini quatres personas :

- **Prépa SUP** (915 étudiants) : Soumissions en fin de semaine (dimanche soir) pour leur TP. Tous les tags sont traités immédiatement à l'issue de la deadline.
- **Prépa SPÉ** (660 étudiants) : Tout comme les SUP, mais avec des projets plus complexes (temps d'éxécution de la test-suite plus long).
- **ING1** (606 étudiants) : Soumissions fréquentes, ces derniers sont traités suivant la soumission indépendement de la deadline.
- **Administration / Equipe éducative** (190 utilisateurs [55 ACDC, 45 ASM, 90 ACU/YAKA]) : Ils doivent pouvoir tester leurs modifications en continu, avec des temps d'exécution courts donc en priorité.

4.1 Constat : Inégalités entre populations

Dans le modèle Waterfall unifié, nous observons des disparités importantes :

- **Population ING :** Arrivées fréquentes, temps de service courts
- **Population PREPA :** Arrivées rares, temps de service longs

Conséquence : Les ING subissent des temps d'attente élevés à cause des PREPA ou alors lors des gros pics de début de journée de piscine et/ou de rush.

4.2 Modèle avec blocage périodique (Dam)

Pour réguler les soumissions ING, afin d'éviter un abus sur le nombre de tag par heure, on introduit un mécanisme de barrage (dam) :

- La moulinette est bloquée pour les ING pendant t_b minutes
- Puis ouverte pendant $t_b/2$ minutes
- Cycle répété

4.3 Alternative : Files séparées (Channels)

4.3.1 Architecture multi-files

Nous proposons deux files distinctes :

```
ING  → File 1 (M/M/c/K) → Runners ING  → Frontend
PREPA → File 2 (M/M/c/K) → Runners PREPA → Frontend
```

Dimensionnement : Proportionnel à la charge :

$$c_1 = \left\lceil \frac{\lambda_{ING}}{\lambda_{total}} \cdot c_{total} \right\rceil$$

$$c_2 = c_{total} - c_1$$

Avantages des files séparées :

- Temps d'attente supposé indépendant pour tous
- Isolation des populations (pas d'interférence)
- Maintien du débit global

Inconvénients :

- Moins de flexibilité (serveurs dédiés)
- Sous-utilisation possible si déséquilibre ponctuel
- Capacité totale de serveur sous-utilisé pouvant mener à un congestion de la file.
- Complexité d'implémentation

4.4 Alternative hybride : Pool partagé avec priorités

Plutôt que des files séparées, utiliser une file unique avec priorités :

- Priorité basse : PREPA (haute charge)
- Priorité haute : ING (faible charge)

Modèle : M/M/c avec classes de priorité (non-preemptive).

Les ING bénéficient d'un traitement préférentiel, les PREPA compensent par leur volume.

5 Optimisation coût/qualité de service

5.1 Modèle de coût (Infrastructure On-Premise)

Puisque l'EPITA est propriétaire de son infrastructure, nous remplaçons le modèle de location horaire par un modèle de Coût Total de Possession (TCO). La fonction de coût devient :

$$C_{total} = C_{amortissement} + C_{energie} + C_{maintenance} \tag{2}$$

5.1.1 Coûts détaillés

1. **Coût d'amortissement ($C_{amortissement}$)**

Il représente le lissage de l'investissement initial du matériel sur sa durée de vie utile.

$$C_{amortissement} = \frac{N_{serveurs} \cdot P_{achat}}{D_{vie} \cdot 365 \cdot 24} \tag{3}$$

Hypothèses et sources :

— $P_{achat} = 1\,200$: Prix moyen d'une tour PC standard ou d'un nœud de calcul 1U (type Dell PowerEdge T150 ou équivalent assemblé) adapté aux *runners* CI/CD. [cite$_start$]

— $D_{vie} = 5$ ans : Durée d'amortissement comptable standard pour le matériel informatique en milieu éducatif[cite : 308].

— **Résultat :** Coût horaire par machine $\approx 0.027\,/h$.

2. **Coût énergétique ($C_{energie}$)**

Ce coût inclut la consommation des serveurs et le refroidissement (PUE) associé au local serveur de l'école.

$$C_{energie} = N_{serveurs} \cdot W_{conso} \cdot PUE \cdot p_{kwh} \tag{4}$$

Hypothèses et sources :

— $W_{conso} = 0.15$ kW (150W) : Consommation moyenne d'un serveur sous charge modérée (CPU à 60-70% durant les tests).

— $PUE = 1.5$: *Power Usage Effectiveness*. Pour 1W consommé par le serveur, 0.5W est utilisé pour la climatisation/onduleurs (standard pour une salle serveur d'école, moins efficace qu'un datacenter Google à 1.1).

— $p_{kwh} = 0.22\,/kWh$: Prix moyen du kWh pour les entreprises en France (base 2025, incluant taxes TICFE).

— **Résultat :** Coût électrique par machine $\approx 0.049\,/h$.

3. **Coût de maintenance matérielle ($C_{maintenance}$)**

Correspond au remplacement des pièces d'usure (disques SSD, alimentations, ventilateurs) hors garantie.

$$C_{maintenance} = N_{serveurs} \cdot (P_{achat} \cdot \tau_{panne}) \tag{5}$$

Hypothèses et sources :

— $\tau_{panne} = 10\%$ par an : Taux standard de maintenance préventive et curative (remplacement de 10% du parc ou des composants chaque année).

— **Résultat :** Coût lissé $\approx 0.014\,/h$ par machine.

5.1.2 Synthèse du coût horaire effectif

En sommant ces composantes, le coût horaire "interne" d'un serveur revient à :

$$p_{serveur/h} = 0.027 + 0.049 + 0.014 \approx 0.10\,/h \tag{6}$$

5.2 Fonction objectif

Nous formulons le problème d'optimisation :

$$\min_{c,K,\mu} [\alpha \cdot \mathbb{E}[T] + \beta \cdot C_{total}(c, K, \mu)] \tag{7}$$

avec $\alpha + \beta = 1$ permettant de pondérer performance vs coût.

Contraintes :

- $c \geq c_{min} = \lceil \lambda/\mu \rceil$ (stabilité)
- $K \geq c$ (cohérence)

5.3 Recommandations de configuration

A travers nos différentes simulations, nous avons pu jouer avec les différents paramètres ainsi que les deux personnae afin de recommander des configurations optimales.

5.3.1 Configuration standard (semaine normale)

Paramètres :

- Serveurs : $c = 6$
- temps de service : $\mu = 30$ secondes

Performances attendues :

- Temps moyen : $\mathbb{E}[T] = 30$ sec
- Coût : 0.6 €/h

5.3.2 Configuration rush (période de deadline)

Paramètres :

- Serveurs : $c = 12$
- temps de service : $\mu = 30$ secondes

Performances attendues :

- Temps moyen : $\mathbb{E}[T] = 30$ sec (exactement le temps de la testsuite)
- Coût : 1.20 €/h

5.4 Stratégies d'auto-scaling

5.4.1 Scaling réactif

Ajustement basé sur l'utilisation courante :

- Si $\rho > 0.8$ pendant 10 min $\rightarrow$ Ajouter 2 serveurs
- Si $\rho < 0.3$ pendant 30 min $\rightarrow$ Retirer 1 serveur

Limites : $c \in [9, maxCapacity]$

5.4.2 Scaling programmé

Configuration par tranche horaire :

Cas hors piscine & rush :

Période	Heures	Serveurs
Nuit	4h-8h	9
Journée	10h-20h	9
Soirée	20h-4h	18

Table 1 – Exemple de scaling programmé

5.4.3 Scaling prédictif

Utilisation de modèles prédictifs (ML) basés sur :

- Historique des soumissions
- Calendrier académique (proximité deadlines)
- Jour de la semaine
- Événements planifiés (examens, rush)

Permet d'anticiper les pics et d'adapter proactivement l'infrastructure.

6 Conclusion

Ce projet a permis de modéliser et d'analyser en profondeur le fonctionnement de la moulinette de tests de l'EPITA à l'aide des outils de la recherche opérationnelle. En combinant modèles théoriques de files d'attente, simulations Monte Carlo et optimisation coût/qualité de service, nous avons mis en évidence les principaux leviers d'amélioration du système, tant sur le plan des performances que de la qualité de service.

L'étude montre que l'introduction de capacités finies et de mécanismes de régulation est indispensable pour maîtriser les pics de charge et limiter les rejets, tout en garantissant la stabilité du système. Les mécanismes de sauvegarde des résultats permettent quant à eux d'éliminer les pertes critiques au prix d'un surcoût maîtrisé. Par ailleurs, la prise en compte de populations étudiantes hétérogènes révèle l'importance de politiques de différenciation (priorités ou files séparées) afin d'assurer une équité de service et d'éviter les interférences entre usages.

Enfin, l'approche d'optimisation intégrant un modèle réaliste de coût d'infrastructure montre qu'un dimensionnement dynamique, appuyé par des stratégies d'auto-scaling, permet d'atteindre un compromis satisfaisant entre performance et coût opérationnel. Ces résultats fournissent des recommandations concrètes et directement applicables pour l'évolution de la moulinette, tout en ouvrant des perspectives futures autour de la prédiction de charge et de l'optimisation adaptative à plus grande échelle.

7 Annexes

A. Notation et formules

Notation de Kendall : A/S/c/K/N/D

- **A :** Loi d'arrivée (M=Markov, D=Déterministe, G=Générale)
- **S :** Loi de service (M, D, G)
- **c :** Nombre de serveurs
- **K :** Capacité totale du système
- **N :** Taille de la population (infinie par défaut)
- **D :** Discipline (FIFO, LIFO, PS, etc.)

Formules principales M/M/c

Utilisation : $\rho = \dfrac{\lambda}{c\mu}$

Stabilité : $\rho < 1$

Nombre moyen en file : $L_q = C(c, a) \cdot \dfrac{\rho}{1 - \rho}$

Temps moyen d'attente : $W_q = \dfrac{L_q}{\lambda}$

B. Code source et implémentation

Le code complet du projet est disponible sur GitHub :

`https://github.com/ErwannLesech/Recherche-Operationnelle-2-File-d-attente`

C. Données et calibration

Sources de données :

- Monitoring Grafana EPITA : `https://grafana.ops.k8s.cri.epita.fr/`
- Expériences personnels (Florian, Aymeric & Erwann) en tant qu'assistants, nous permettant d'interagir avec la moulinette
- Enquêtes auprès d'étudiants travaillant à la Forge

Paramètres estimés & recueillis :

- Population totale active : 900 étudiants en pic d'activité
- Taux de soumission moyen : 3 tags/étudiant/heure
- Temps d'exécution moyen : 30 secondes (varie de 30 secondes à plusieurs minutes)
- Combien de temps de timeout les test-suite disposent en moyenne : 3 minutes (peut varier lors de la conception)
- Combien de serveurs dans la moulinette réelle d'EPITA : 9 Runners
- Coût moyen total d'un serveur : 0.10 €/runner/heure
- Il y a t'il un serveur de traitement uniquement pour l'équipe pédago : Non
- Il y a t'il un auto scaling en cas de surutilisation : Non, seulement manuel si besoin