La Prépa des INP

Rapport de stage 2A

Robustesse de stratégies d'ordonnancement pour applications stochastiques

---

Stagiaire :
Lisa Weisbecker
Entreprise d'accueil :
Inria

Tuteur de stage :
M. Guillaume Pallez
Enseignant référent :
M. Pierre Langot

3 Mai 2021 — 18 Juin 2021

Table des matières

1 Introduction

1.1 Remerciements

Je tiens à remercier l'ensemble des personnes qui ont rendu cette expérience possible.
Tout d'abord, je remercie vivement mon Maître de stage Mr Guillaume Pallez pour son accueil et sa bienveillance, et pour avoir partagé ses connaissances très pédagogiquement tout au long de mon stage.
Je remercie également Nicolas Vidal qui, bien qu'il soit occupé par l'écriture d'une thèse, a toujours pris le temps pour m'aider, répondre à mes questions ou discuter de son expérience.
Je remercie l'ensemble de l'équipe TADAAM pour son accueil chaleureux et le reste des employés d'INRIA avec qui j'ai pu interagir et qui ont rendu cette expérience enrichissante et agréable.
Enfin, je remercie l'équipe pédagogique de La Prépa des INP ainsi que mon enseignant référent, Mr Pierre Langot, pour son accompagnement le long de ma période de stage.

1.2 Lieu et période du stage

J'ai réalisé mon stage à l'INRIA du 3 mai 2021 au 18 Juin 2021, période durant laquelle j'ai pu intégrer l'équipe TADAAM au sein du laboratoire. J'ai particulièrement travaillé avec Guillaume Pallez, mon tuteur de stage et chercheur au laboratoire qui étudie principalement des problèmes d'ordonnancement mais aussi des problèmes algorithmiques consistant par exemple à trouver de bons algorithmes d'approximation et Nicolas Vidal, qui réalise sa troisième année de thèse et travaille notamment sur l'optimisation de stratégies de réservation, en se concentrant sur des aspects différents de ceux de mon stage.

1.3 Présentation de l'entreprise

L'INRIA (Institut National de Recherche en Informatique et en Automatique) est un établissement publique de recherche en informatique et en mathématiques. Parmi ces missions, on trouve le développement de la recherche et de la valorisation en sciences et techniques de l'information et de la communication.

«En résumé, l'ambition stratégique d'Inria est d'accélérer le processus d'asseoir le leadership scientifique, technologique et industriel de la France, à la

fois dans et par le numérique, dans une dynamique européenne plus large. Inria doit remplir son rôle de garant de l'autonomie stratégique et de la souveraineté de la France dans le numérique. Cette ambition repose sur la cohérence et la synergie entre sa politique nationale et sa politique régionale, engageant pleinement Inria dans le développement d'universités de recherche de premier plan au sein d'écosystèmes entrepreneuriaux et industriels portés par les technologies numériques.»
Objectives and performance contract 2019-2023 – Between the French government and Inria

L'équipe TADAAM (Topology-aware system-scale data management for high-performance computing) est une équipe multiculturelle composée de nombreux chercheurs et étudiants en thèse ou stagiaires travaillant principalement sur des problèmes d'optimisation et de calculs haute performance. Les rôles dans l'équipe couvrent de nombreux domaines, car chacun a sa spécialité. Ainsi, certains travaillent au plus près de la machine tandis que d'autres, comme mon maître de stage et moi, étudient plus théoriquement les algorithmes et les protocoles à fournir à ces machines.

1.4 Ordonnancement et stratégies

En informatique, l'ordonnancement est un principe visant à optimiser la répartition de "tâches" sur un ordinateur ou un système. Il s'agit d'optimiser le temps d'exécution, la part d'utilisation, et l'emprunte mémoire de l'exécution. Mon stage porte sur l'optimisation des temps d'exécution.
En effet, lorsqu'un utilisateur veut exécuter une tâche sur un ordinateur, il doit demander à l'avance ce qu'on appelle un temps de réservation. Si on connaissait la durée exact de cette tâche, il suffirait de réserver ce temps ci. Mais qu'en est-il lorsqu'on ne connait pas cette durée à l'avance ? Par exemple, considérons une tâche dont le temps d'exécution suit une loi de probabilité (une loi normale par exemple) et en constitue un tirage aléatoire. C'est ce qu'on appelle une application, ou un processus stochastique. Quelle serait alors le temps de réservation idéal ?
On peut penser qu'il suffit de prendre un temps de réservation très grand afin d'être sûr d'avoir le temps d'exécuter notre tâche. Cependant, l'utilisateur ne paye pas le temps d'exécution de la tâche, mais le temps qu'il a réservé. Il s'agit donc de minimiser ce temps de réservation, et donc le coût pour l'utilisateur. Afin de répondre à cette problématique, différentes stratégies peuvent être adoptées.

Il faut savoir que le temps de réservation n'ai pas forcément constitué d'un seul "bloc". Ainsi, une première stratégie peut être de faire une première réservation de durée t_1, et si celle ci ne suffit pas pour réaliser la tâche, faire une autre réservation plus longue de durée t_2 et retenter l'exécution. On peut répéter cette opération jusqu'à ce que la tâche soit exécutée entièrement.
Autant cette stratégie peut se révéler efficace, autant elle peut conduire à un coût d'exécution très élevé si on est "malchanceux", puisqu'on payera $t_1 + t_2 + ... + t_n$, n étant le premier entier tel que $t_n \geq t$, t le temps d'exécution de la tâche.
Pour améliorer cette stratégie, introduisons ce qu'on appelle les "checkpoint". Le Checkpoint permet de sauvegarder ce qui a déjà été calculé durant le temps t_1 au lieu de repartir à 0. Cependant, un checkpoint implique un coût, car il faut réserver ce "temps de sauvegarde", et également un temps de restart pour recharger les données avant de reprendre l'exécution pendant t_2.
Ainsi, une stratégie optimisée comportera des temps d'exécution et éventuellement des checkpoint et des restart. Le coût d'une tâche sera ainsi par exemple $t_1 + C + R + (t_2 - t_1) + ...R + (t_n - t_{n-1})$ avec n le premier entier tel que $t_n \geq t$ et C et R respectivement les coûts de restart et Checkpoint. La figure ci-dessous illustre différentes stratégies.

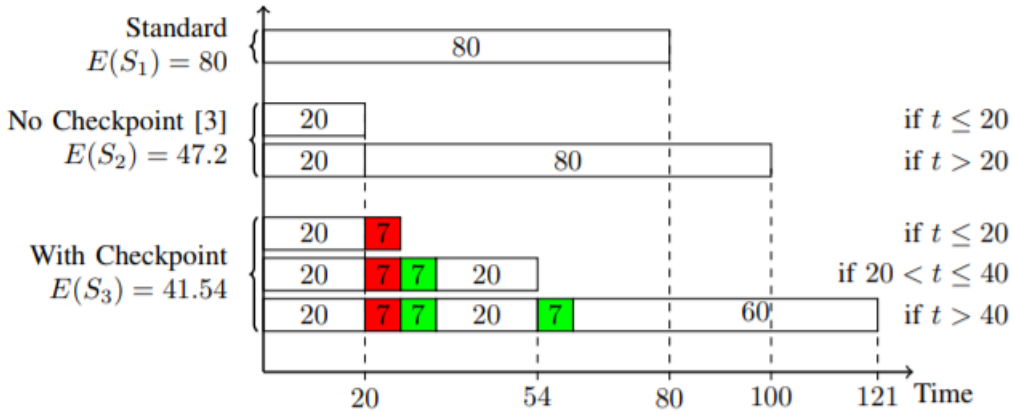

FIGURE 1 – Illustration de stratégies de réservation (en rouge les checkpoint et en vert les restart) [6]

1.5 Objectif du stage

Mon stage se déroule dans la continuité de la thèse réalisée par Valentin Honoré [6]. Durant sa thèse à INRIA, il a pu réaliser des algorithmes qui, à partir d'une distribution donnée, renvoient la stratégie optimale pour li-

miter le temps de réservation pour l'exécution d'une tâche. Ces algorithmes s'appuient sur une distribution dont on connait les paramètres. Un problème se pose donc lorsqu'on ne connait pas les paramètres de la loi régissant les temps d'exécution. Une manière de trouver une stratégie optimale est alors de réaliser un certain nombre de test et d'obtenir ce qu'on appelle un échantillon aléatoire ("sample") de temps d'exécution. A partir de cet échantillon, on peut approximer les paramètres de la distribution inconnue et en déduire une distribution approximative, qu'on utilisera pour créer notre stratégie.
Mon travail durant le stage est donc, dans un premier temps, de réaliser des méthodes algorithmiques d'approximation de lois de probabilité sur Python, puis d'étudier la robustesse des stratégies d'ordonnancement. Concrètement, il s'agit de regarder comment se comporte la "stratégie approximée" lorsqu'elle est appliquée à la distribution inconnue, donc de l'évaluer, et déterminer la taille de l'échantillon dont on a besoin pour que le coût de cette stratégie se rapproche du coût optimale.
Dans un second temps, j'ai étudié l'évolution de la stratégie (en particulier la durée t_1 de la première réservation) en fonction de différents paramètres comme la moyenne de la distribution, son écart type, etc... A partir de ces résultats expérimentaux nous avons tenté, dans un deuxième temps, de mathématiquement expliquer ces comportements.
Durant le stage, j'ai majoritairement utilisé l'algorithme "All-Checkpoint" qui, comme son nom l'indique, renvoie une stratégie comportant un checkpoint après chaque temps de réservation.

2 Partie expérimentale

Supposons que nous ne connaissons pas les paramètres de la loi régissant les temps d'exécutions des tâches, mais seulement sa forme (par exemple normale, exponentielle, lognormale, etc...). Comment déterminer une stratégie optimale adaptée et évaluer son coût ? Cette évaluation reposera sur quatre étapes : l'estimation des paramètres de la loi pour l'interpoler, la discrétisation de cette nouvelle distribution approximée, la détermination de la stratégie à partir de la distribution discrète, et enfin l'évaluation de cette stratégie.

2.1 Rappels loi de probabilité

Pour faciliter la compréhension, cette section contient un bref rappel des fonctions à densité ainsi que des fonctions de répartition des lois de probabi-

lité utilisées dans la suite du rapport.

Loi normale $\mathcal{N}(\mu,\sigma^2)$

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\frac{x-\mu}{\sigma}^2} \qquad F(x) = \frac{1}{2}\left[1 + erf\left(\frac{x-\mu}{\sigma\sqrt{2}}\right)\right]$$

Loi exponentielle $\mathcal{E}(\theta)$

$$f(x) = \begin{cases} \lambda e^{-\lambda x} & si\ x \geq 0 \\ 0 & si\ x < 0 \end{cases} \qquad F(x) = \begin{cases} 1 - e^{-\lambda x} & si\ x \geq 0 \\ 0 & si\ x < 0 \end{cases}$$

Loi de Weibull $\mathcal{W}(k,\lambda)$

$$f(x) = \begin{cases} \frac{k}{\lambda}\left(\frac{x}{\lambda}\right)^{k-1} e^{-(\frac{x}{\lambda})^k} & si\ x \geq 0 \\ 0 & si\ x < 0 \end{cases} \qquad F(x) = \begin{cases} 1 - e^{-(\frac{x}{\lambda})^k} & si\ x \geq 0 \\ 0 & si\ x < 0 \end{cases}$$

2.2 Les étapes algorithmiques de l'évaluation

2.2.1 Approximation

Le but de cette étape est de créer un algorithme qui, à partir d'un échantillon de valeurs obtenu à partir d'une loi de probabilité continue, permet d'obtenir une approximation de la fonction à densité d'origine.
Pour cela, on peut utiliser de manière classique les estimateurs communs de la variance et de l'espérance d'une loi de probabilité, qu'on a pu rencontrer dans nos cours de probabilité en deuxième année.
Soit $(x_1, ..., x_n)$ un échantillon aléatoire. Pour estimer la moyenne, on prend $\bar{X} = \frac{1}{n}\sum_{i=1}^{n} x_i$. Pour l'écart type, on prend $\sigma^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{X})^2$.

Ces estimateurs fonctionnent relativement bien pour la loi normale et la loi exponentielle, même pour un faible nombre de valeurs.
Cependant, il n'existe pas de simples estimateurs pour déterminer les paramètres de la loi de Weibull il faut employer une autre stratégie : l'utilisation de la fonction de vraisemblance. La vraisemblance est une fonction qui permet de mesurer l'adéquation entre une distribution observée sur un échantillon aléatoire et la loi de probabilité dont on suppose est tiré cet échantillon. Cette fonction prend ainsi en entrée une distribution (l'échantillon) et un ou

plusieurs paramètres. Maximiser la vraisemblance, en faisant varier les paramètres en entrée, c'est trouver la meilleure approximation des paramètres de la loi originale, et donc interpoler sa fonction à densité. Elle s'exprime de la façon suivante, pour un échantillon tiré d'une loi $\mathcal{L}$:

$$L_n \ : \ (x_1, ..., x_n; \theta) \ \mapsto \ L_n(x_1, ..., x_n; \theta) \ = \ \prod_{i=1}^{n} \mathbb{P}_\theta(X_i = x_i) \ , \ pour \ X_i \hookrightarrow \mathcal{L}(\theta)$$

On peut comparer cette méthode à celle des estimateurs sur la loi normale. En effet, avec la méthode des estimateurs (pour un échantillon de petite taille), l'erreur sur l'écart type de la loi était parfois élevé, allant jusqu'à 25% dans les plus mauvais cas. En utilisant la méthode de la vraisemblance, les résultats pour un petit échantillon sont différents de la méthode des estimateurs mais pas meilleurs dans tout les cas, voire le graphique ci-dessous, avec en bleu la distribution originale et son tirage aléatoire de 5 valeurs en rouge, en orange la courbe obtenue par la méthode des estimateurs et en vert la courbe obtenue par la méthode de la vraisemblance, pour 2 tirages différents.

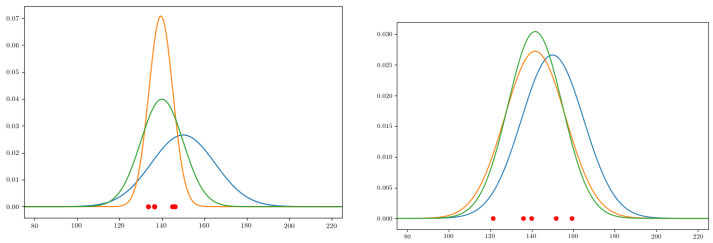

FIGURE 2 – Interpolation d'une loi normale par différentes méthodes d'approximation pour 2 tirages de 5 valeurs

Pour un tirage de plus grande taille (par exemple 50 valeurs), les résultats des deux méthodes se ressemblent considérablement.
Pour un tirage encore plus grand (par exemple 200 valeurs), les deux méthodes donnent un résultat très précis (exact à 10^-2 près).

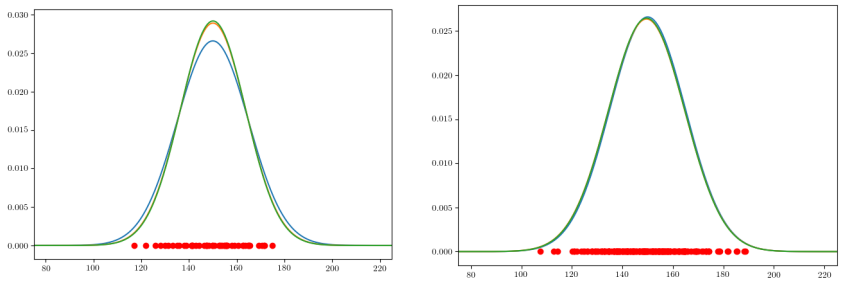

FIGURE 3 – Interpolation d'une loi normale par différentes méthodes pour un tirage de 50 valeurs (gauche) et 200 valeurs (droite).

Pour la loi exponentielle, avec la méthode des estimateurs, on estime de la même façon la valeur de la moyenne μ afin d'obtenir la valeur de $\theta = \frac{1}{\mu}$. la valeur du paramètre θ obtenue est la même par la méthode de la vraisemblance et par la méthode des estimateurs, à 10^{-2} près, même pour un petit échantillon (5 valeurs). Avec un plus grand échantillon (20 valeurs), le résultat est rapidement très proche de la valeur originale de θ (égal à 10^{-2} près).

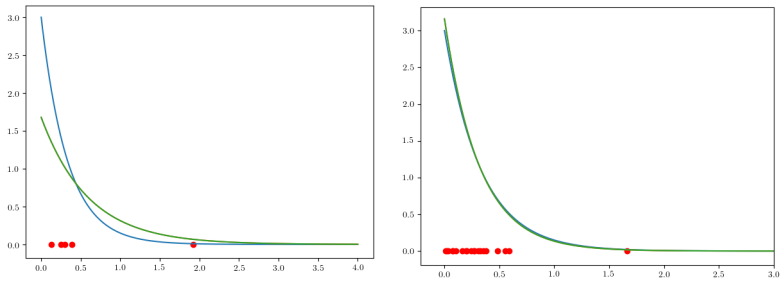

FIGURE 4 – Interpolation d'une loi exponentielle par différentes méthodes pour un échantillon de 5 valeurs (gauche) et 20 valeurs (droite)

Ainsi, j'ai pu observé que la méthode de la vraisemblance ne donne pas de meilleurs résultats pour la loi exponentielle, et des résultats souvent légèrement plus précis mais pas systématiquement pour la loi normale. Cependant, la méthode de la vraisemblance étant beaucoup plus longue à exécuter, j'ai décidé, pour des raisons de complexité algorithmique, de garder la méthode des estimateurs pour la loi normale et la loi exponentielle.
Elle est cependant nécessaire pour la loi de Weibull dont les paramètres ne peuvent pas être directement calculés avec les estimateurs de la moyenne et de l'écart type. J'ai d'abord utilisé la fonction "fit" directement implantée

dans la bibliothèque scipy de python qui renvoie les paramètres optimaux pour la distribution proposée. Pour cette partie, afin de simplifier l'étude de la distribution de Weibull, nous nous placerons dans le cas où le paramètre d'échelle $\lambda = 1$, et nous ferons simplement varier le paramètre de forme k.
La particularité de la fonction de Weibull est que sa forme change considérablement selon son paramètre k. Ainsi, pour $k \leq 1$, la fonction à densité ressemble à une loi exponentielle. Pour $k > 1$, elle ressemble plutôt à une lognormale. Une mauvaise estimation de ce paramètre entraîne une grande erreur sur l'approximation de la fonction à densité. C'est souvent ce qui arrive pour de petits échantillons, comme on peut le voir sur l'exemple ci dessous. La courbe bleu (loi originale dont est tirée l'échantillon) est la même sur les deux graphiques, il s'agit d'une loi de Weibull de paramètre $k = 1.5$. A deux reprise, j'ai réalisé un tirage de 10 valeurs suivit d'une approximation de la loi originelle. Pour le premier tirage, l'approximation a donnée $k = 1.19$, on a donc obtenu une loi ressemblant à la loi originale. Cependant lors du deuxième tirage, on a obtenu $k = 0.95$, et donc une courbe de type "exponentielle".

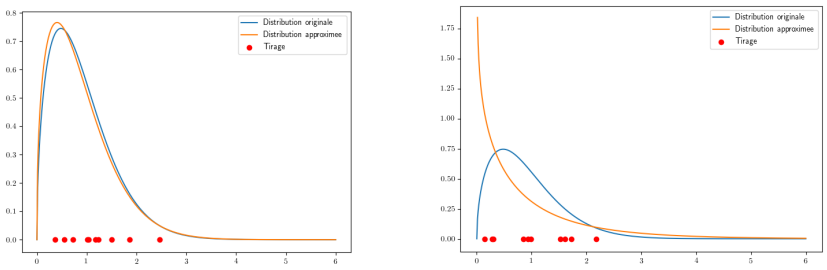

FIGURE 5 – Interpolation d'une loi de Weibull pour deux tirages de 10 valeurs

Ce problème n'apparait plus avec un grand nombre de valeurs et l'approximation devient relativement exact :

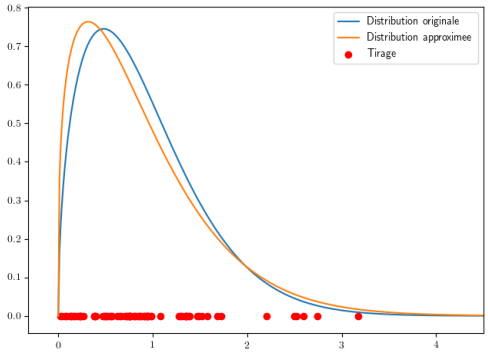

FIGURE 6 – Interpolation d'une loi de Weibull avec un échantillon de 75 valeurs (méthode scipy)

Pour régler ce problème, j'ai tenter de coder un algorithme utilisant la méthode de la vraisemblance sans utiliser la bibliothèque scipy. J'ai alors obtenu de bien meilleurs résultats, comme on peut le voir ci dessous, avec en bleu la distribution originale, en rouge le tirage aléatoire, en orange la méthode utilisant la bibliothèque scipy et en vert la méthode utilisant directement la fonction de vraisemblance que j'ai codé. J'en ai déduit que la fonction "fit" de scipy, qui renvoie la fonction à densité dans sa bibliothèque qui semble correspondre le plus au tirage aléatoire, nécessite un plus grand échantillon pour être efficace.

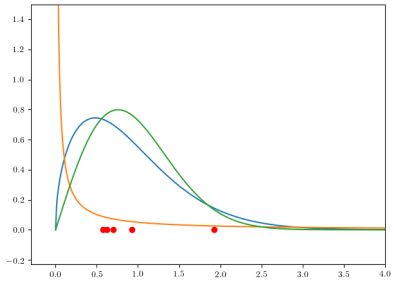

FIGURE 7 – Interpolation d'une loi de Weibull par différentes méthodes pour un tirage de 5 valeurs

Ces résultats se confirmant sur un grand nombre de test, j'ai décidé d'utiliser cette méthode pour la suite, bien qu'elle soit plus longue à exécuter que celle utilisant la bibliothèque scipy.

2.2.2 Discrétisation

La discrétisation consiste à échantillonner la distribution obtenue, non pas aléatoirement cette fois mais à intervalles réguliers. Elle comporte un grand nombre de points et permet d'obtenir une distribution discrète à partir d'une fonction à densité continue.
Cette étape est nécessaire pour la partie suivante, car l'algorithme permettant d'obtenir une stratégie optimale prend en entrée une distribution discrète. Puisqu'on veut obtenir un nombre finit de point, il faut borner la fonction. Par exemple, si on note a et b les bornes pour la discrétisation de la fonction X, on peut choisir le critère minimal suivant :
$\mathbb{P}(X < a) \leq 1\%$ et $\mathbb{P}(X > b) \leq 1\%$.
Puisque je serai, par la suite, amenée à faire varier certains paramètres comme la variance ou la moyenne tout en gardant des bornes constantes, j'ai choisi dans mes tests des bornes très larges. Par exemple ici, pour une loi normale de paramètres $\mu = 150$ et $\sigma = 15$, j'ai choisi les bornes 0 et 300. On a $\mathbb{P}(X < 0) \approx 0$ et $\mathbb{P}(X > 300) \approx 0$, ce qui est largement satisfaisant.

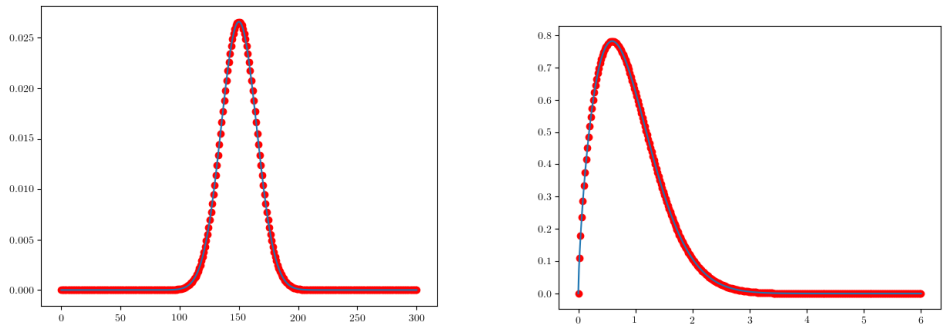

FIGURE 8 – Discrétisation d'une loi normale (premier graphique) et d'une loi de Weibull (deuxième graphique) avec 300 points

Comme on peut l'observer, contrairement à un tirage aléatoire, cet échantillonnage est réalisé à intervalles réguliers et la densité de points est la même le long de la courbe. Pour obtenir la probabilité de chaque événement (les n ν_i relevés à intervalles réguliers entre nos bornes), on utilise la fonction de répartition, avec $\mathcal{P}(X = \nu_i) = F(\nu_i) - F(\nu_{i-1})$

2.2.3 Optimisation

Comme précisé précédemment, cette partie s'appuie essentiellement sur l'algorithme de Valentin Honoré [6], dont j'ai simplement modifié quelques

paramètres pour pouvoir l'utiliser sur mes propres distributions. Je vais cependant en expliquer brièvement son principe.
L'algorithme prend en entrée la distribution (concrètement une liste composée de deux sous liste, l'une comportant l'ensemble des "évènements" ν_i et l'autre les $f(\nu_i)$), le minimum de la distribution (donc la borne inférieur de la discrétisation), la moyenne de la distribution, le nombre d'éléments dans la distribution (donc le nombre de ν_i) et enfin la fonction de coût qui associe à une durée t de réservation son coût pour l'utilisateur associé. Ici, il s'agira tout simplement de la fonction identité. L'algorithme s'appuie également sur des paramètres fixé en amont et qui resteront constants lors de l'étude : les coûts de Checkpoint et de Restart, respectivement notés C et R.
L'algorithme repose sur la programmation dynamique [9], une méthode algorithmique populaire dans la résolution de problèmes d'optimisation. Elle consiste à séparer un problème en "sous problèmes" et à résoudre d'abord les sous problèmes dont on stock les résultats intermédiaires. Ici, plus précisément, l'algorithme utilise le "Backtracking", consistant à examiner toutes les résolutions possibles d'un problème, et revenir en arrière si la solution n'est pas satisfaisante (d'où le nom "Backtracking"="Retour sur trace").

2.2.4 Évaluation

Cette dernière étape consiste à évaluer le coût de la stratégie qu'on a déterminé. Concrètement, il s'agit de l'espérance mathématique de la fonction de coût.
On le rappelle, pour l'algorithme qu'on utilise ici (qui checkpoint après chaque réservation), une stratégie sera de la forme suivante :
$(t_1 + C, R + t_2 + C, R + ..., R + t_n + C)$ où t_n sera la borne supérieur de la discrétisation. Si on note t la durée d'une tâche aléatoire, le temps $t_2 - t_1$ n'est réservé que si $t > t_1$, $t_3 - t_2$ n'est réservé que si $t > t_2$, et ainsi de suite. Donc si, pour un t quelconque suivant la la loi de probabilité à partir de laquelle on a déterminé la stratégie, on note k le premier entier tel que $t_k > t$, le coût de la réservation sera le suivant :

$$C(t) = t_1 + C + R + (t_2 - t_1) + C + R... + R + (t_k - t_{k-1}) + C$$

Il est ainsi aisé de déterminer l'espérance d'une telle fonction de coût, qui s'exprimera comme ceci :

$$E(C(X)) = t_1 + C + \sum_{i=2}^{n} \mathbb{P}(X > t_{i-1}) * (R + (t_i - t_{i-1}) + C)$$

En notant F la fonction de répartition de la loi de probabilité, cette formule se note donc :

$$E(C(X)) = t_1 + C + \sum_{i=2}^{n} (1 - F(t_{i-1})) * (R + (t_i - t_{i-1}) + C)$$

2.3 Code et expériences

2.3.1 Code principale

Le code principale qui réalise les tests et les expériences vise à associer les 4 étapes algorithmiques décrites précédemment. On commence par générer une loi de probabilité. Pour des raisons de cohérences avec le contexte, on choisit une loi dont tout les évènements sont positifs ou dont la probabilité qu'un évènement négatif advienne tende vers 0 (comme c'est le cas avec la loi normale décrite précédemment). En effet, la durée d'une tâche à exécuter ne sera jamais négative, à moins qu'on apprenne à remonter le temps. Ensuite, on va échantillonner aléatoirement cette loi, déterminer une approximation de ces paramètres en imaginant qu'ils sont inconnus, discrétiser cette loi approximée, établir une stratégie et l'évaluer.
Cependant, il ne faut pas perdre de vue le but de mon stage, c'est à dire d'étudier la robustesse de ces stratégies. L'objectif est donc de déterminer la taille que doit avoir l'échantillon qu'on possède afin que le coût de la stratégie obtenue grâce à nos approximation se rapproche du coût de la stratégie optimale pour la distribution.
Pour cela je vais donc réaliser une série de test pour différentes tailles d'échantillons. En l'occurrence, 50 tests pour des échantillons de 5 valeurs, 50 tests pour des échantillons de 10 valeurs, et de même pour 20, 30, 50, 75 et 100 valeurs. Pour chaque test, qui correspond à un tirage différent, on estime les paramètres de la loi, on détermine une stratégie et on l'évalue en l'appliquant à la distribution originale.
Enfin, plutôt que de tracer un graphique qui, pour chaque tirage, trace l'espérance du coût de la stratégie, j'ai plutôt tracé le rapport entre l'espérance du coût de la stratégie et le coût optimale pour cette distribution (simplement obtenu avec la même méthode mais en sautant la première étape, donc en discrétisant directement la fonction à densité originale). Plus cette valeur se rapproche de 1, plus la stratégie déterminée est efficace, 1 étant la valeur optimale. Afin de mieux observer cette tendance, j'ai également tracé sur le graphique la droite y=1.

2.3.2 Premiers résultats

J'ai donc réalisé ces séries de test pour une loi normale, une loi exponentielle et une loi de Weibull. Lors de l'exécution de code, le programme demande le nom de la loi, ces paramètres, les bornes d'études et le nombre de tests pour chaque taille d'échantillon. La durée d'exécution varie beaucoup selon le nombre de tests, le nombre de points utilisés pour la discrétisation (qui influe beaucoup sur la durée d'exécution de l'algorithme de détermination de la stratégie) et la méthode utilisé. Ainsi, pour une loi normale, avec 20 tests, 300 points sur la discrétisation et la méthode des estimateurs, il est possible d'obtenir un graphique en moins d'une minute. Cependant, en augmentant le nombre et la précision des résultats (par exemple 600 points sur la discrétisation et 50 tests) le programme doit parfois tourner pendant plus d'une heure. J'ai donc dû raisonnablement doser ces paramètres pour obtenir les premiers résultats, que j'ai tracé sous forme de nuage de point. Chaque point bleu correspond à un tirage et est égal au rapport du coût de la stratégie approximée sur le coût optimale pour la distribution générée. Pour chaque taille de tirage, j'ai marqué en rouge la médiane de l'ensemble de ces rapports.

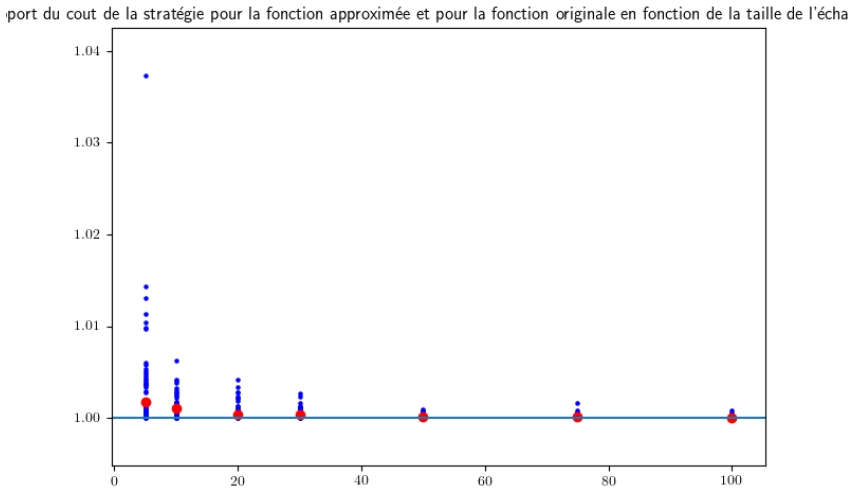

FIGURE 9 – Évaluation des stratégies obtenues par approximation en fonction de la taille de l'échantillon pour une loi normale $\mathcal{N}(150, 15^2)$

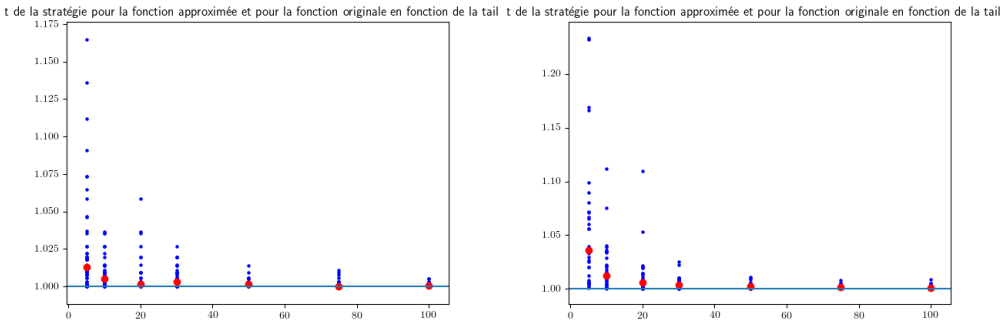

FIGURE 10 – Évaluation des stratégies obtenues par approximation en fonction de la taille de l'échantillon pour une loi exponentielle $\mathcal{E}(3)$ (gauche) et une loi de Weibull $\mathcal{W}(1.5, 1)$ (droite)

Les résultats obtenus témoignent bien de la robustesse des stratégies. En effet, même à partir d'un échantillon de seulement 5 valeurs, le coût de la stratégie est seulement environ 4% plus élevé au maximum pour la loi normale, 17% pour la loi exponentielle, et 24% pour la loi de Weibull. Et il s'agit uniquement du pire des cas : si on regarde la médiane (point rouge), on observe que la moitié des stratégies obtenues avec un tirage de 5 valeurs ont un coût moins de 0,3% plus élevé que celui de la stratégie optimale pour la loi normale, environ moins de 1.5% plus élevé pour la loi exponentielle, et moins de 4% plus élevé pour la loi de Weibull.
Pour toutes les lois, et particulièrement la loi exponentielle, la médiane se confond avec 1 lorsqu'on augmente la taille de l'échantillon. On peut donc considérer, à partir de ces observations, qu'il suffit de réaliser un petit nombre de tests pour obtenir une stratégie de réservation adaptée et efficace pour une distribution inconnue. En augmentant le nombre de tests, on augmente drastiquement la probabilité d'avoir une très bonne stratégie.

2.3.3 Nouvelle approche

Dans la section précédente, nous avons expérimentalement pu observé la robustesse des stratégies de réservation. Cependant, on a considéré connaître à l'avance le type de la loi régissant les temps d'exécution des tâches, les paramètres de cette loi étant les seuls inconnus. J'ai donc approximé une loi normale par une loi normale, une loi exponentielle par une loi exponentielle, etc... Qu'en est-il lorsqu'on "se trompe" ? C'est à dire, lorsqu'on approxime par exemple une loi normale par une loi exponentielle, et inversement.

J'ai donc ajouté à mon code un nouveau paramètre : la forme de la loi utilisée pour l'approximation. Puis, j'ai réalisé exactement le même processus que précédemment mais en approximant une loi normale par une loi exponentielle puis une loi exponentielle par une loi normale. Par exemple, dans le premier cas, on calcule la moyenne μ du tirage aléatoire (donc tiré d'une loi normale) et on approxime cette loi par une loi exponentielle de paramètre $\theta = \frac{1}{\mu}$. Même chose dans le deuxième cas en faisant le calcul dans le sens inverse. J'ai obtenu les résultats suivants :

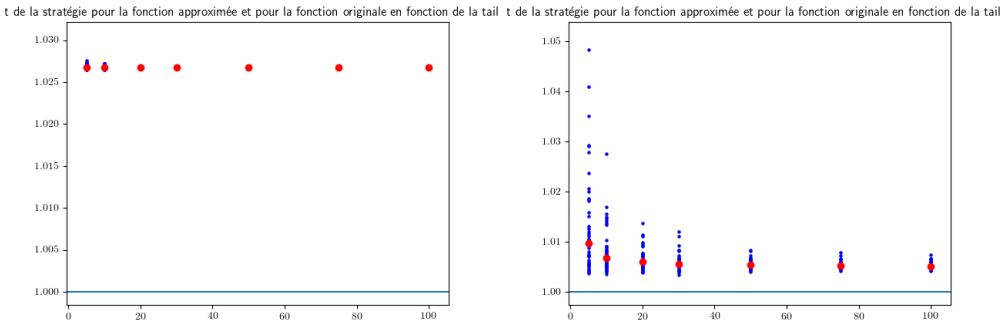

FIGURE 11 – Évaluation des stratégies obtenues par approximation en fonction de la taille de l'échantillon pour une loi normale $\mathcal{N}(150, 15^2)$ approximée par une loi exponentielle (gauche) et une loi exponentielle $\mathcal{E}(1/150)$ approximée par une loi normale (droite)

Ces résultats sont de bons exemples permettant de montrer que les stratégies de réservation obtenues par approximation restent efficaces même lorsqu'on ne connait ni les paramètres, ni la forme de la loi régissant les temps d'exécution. Cependant, il y a 6 combinaisons possibles à tester, si on inclue la loi de Weibull. J'ai donc décidé d'utiliser une autre méthode pour exposer ces résultats.

2.3.4 Mise en forme des résultats

Les résultats obtenus sous forme de nuage de points sont peu lisibles et ne permettent pas de comparer plusieurs graphiques ou de les combiner (par exemple, en mettant sur le même graphe les résultats pour une loi normale approximée par la loi normale et par la loi exponentielle). J'ai donc utilisé le logiciel Rstudio afin de générer des diagrammes de type "boîte à moustache". J'ai pu obtenir les graphes suivants :

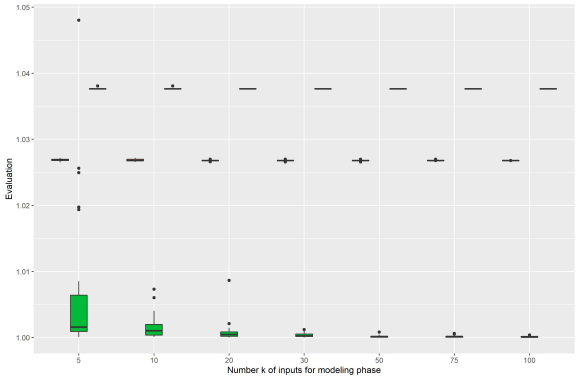

FIGURE 12 – Évaluation des stratégies obtenues par approximation en fonction de la taille de l'échantillon pour une loi normale $\mathcal{N}(150, 15^2)$ approximée par une loi exponentielle (en rouge, autour de 1.03), une loi normale (en vert) et une loi de Weibull (en bleu, autour de 1.04))

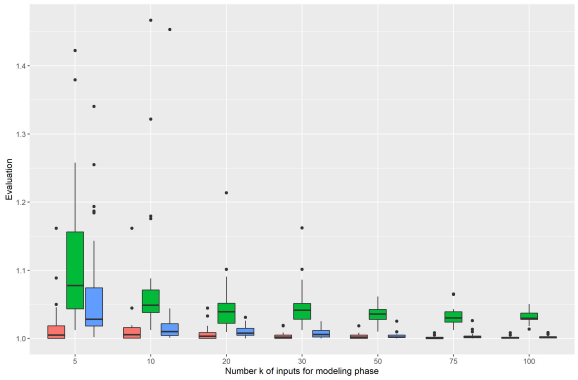

FIGURE 13 – Évaluation des stratégies obtenues par approximation en fonction de la taille de l'échantillon pour une loi exponentielle $\mathcal{E}(1/150)$ approximée par une loi exponentielle (en rouge), une loi normale (en vert) et une loi de Weibull (en bleu)

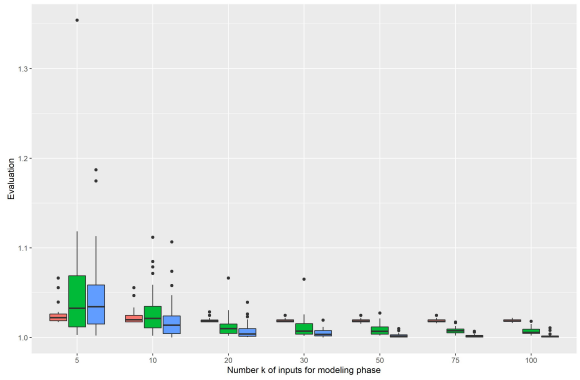

FIGURE 14 – Évaluation des stratégies obtenues par approximation en fonction de la taille de l'échantillon pour une loi de Weibull $\mathcal{W}(1.5, 1)$ approximée par une loi exponentielle (en rouge), une loi normale (en vert) et une loi de Weibull (en bleu)

Ces graphiques mettent en valeur certaines propriétés moins visibles sur les graphes précédents. Tout d'abord, comme attendu, les stratégies obtenues lorsqu'on connait la forme de la loi à l'avance se comportent mieux sur la loi originale que celles obtenues en se "trompant". On observe également que l'étalement des valeurs est bien plus petit pour une approximation par la loi exponentielle, même pour un échantillon de très petite taille, et la médiane reste quasiment constante peu importe la taille de l'échantillon. On peut en déduire que la taille de l'échantillon influe très peu sur l'estimation des paramètres de la loi exponentielle, et qu'on peut obtenir une interpolation de la loi très convenable même avec seulement 5 valeurs. Pour la loi de Weibull, on remarque, comme attendu, qu'une approximation par une loi de Weibull donne des stratégies très efficaces, mais également pour une approximation par la loi normale. Ce phénomène peut être dû à la ressemblance entre la loi normale et la loi de Weibull pour un paramètre k=1,5.

Pour mettre en valeur sa robustesse, on peut également comparer cet algorithme à un autre bien moins efficace et pourtant utilisé, qui consiste à prendre comme premier temps de réservation le maximum de l'échantillon et, si ce n'est pas suffisant, prendre comme second temps de réservation $t_2 = 1.5 * t_1$, et ainsi de suite... Voici les résultats obtenus pour la même loi normale :

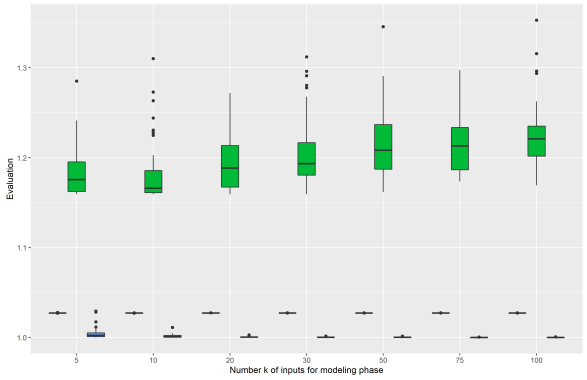

FIGURE 15 – Évaluation des stratégies obtenues par approximation en fonction de la taille de l'échantillon pour une loi normale $\mathcal{N}(150, 15^2)$ approximée par une loi exponentielle (en rouge), une loi normale (en bleu), et de la stratégie du maximum de l'échantillon (en vert)

Ce graphique met en valeur la robustesse de l'algorithme étudié. En effet, l'algorithme simpliste utilisant le maximum de l'échantillon donne des stratégies bien plus coûteuses et pas du tout dans le même ordre de grandeur. De plus, plus l'échantillon est grand, plus le coût est élevé.

2.4 Limites

On peut également observer les limites de cette méthode, par exemple avec une loi normale approximée par une loi exponentielle.

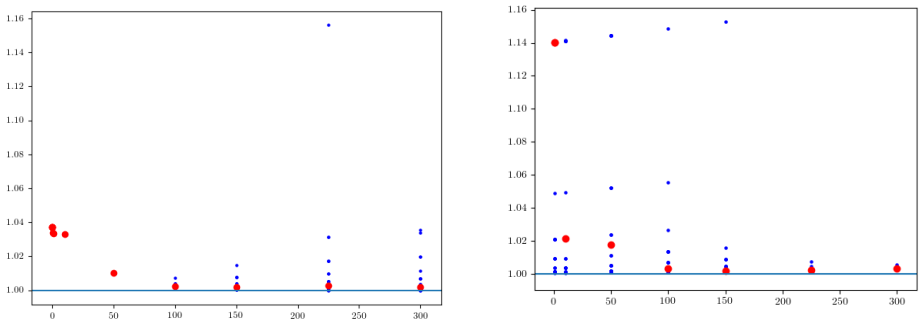

FIGURE 16 – Évaluation des stratégies obtenues par approximation pour des échantillons de 5 valeurs en fonction de σ pour une loi normale $\mathcal{N}(150, \sigma^2)$ approximée par une loi exponentielle (à gauche), et en fonction de μ pour une loi normale $\mathcal{N}(\mu, 150^2)$ approximée par une loi exponentielle (à droite)

Dans la figure de gauche, σ évolue entre $\frac{1}{2\mu}$ et 2μ, μ fixé à 150, et dans la figure de droite μ évolue entre 0.5 et 2σ, σ fixé à 150. Il n'était pas possible de tester une moyenne de type $\frac{1}{\sigma}$ pour le graphique de droite en raison d'"erreurs de plage mathématique" lors du calcul de la fonction de répartition, donc des nombres trop grands. On peut considérer cela comme une première limite du modèle.
On peut également remarquer que, pour une moyenne de 150 et un écart type proche de 0 (donc une fonction déterministe), il n'y a aucun étalement de valeurs mais on ne peut pas obtenir de stratégie moins de 4% plus coûteuses environ, ce qui semble être une autre limite au modèle.

3 Partie théorique

Dans la section précédente, on a pu observer la robustesse par des tests et des résultats expérimentaux, sans pour autant comprendre et prouver ces résultats. Dans cette partie, nous allons donc commencer par étudier à l'aide de Python l'évolution de la stratégie de réservation (particulièrement du premier temps de réservation t_1)en fonction de différents paramètres (comme la moyenne ou l'écart type de la loi) pour intuiter des propriétés mathématiques. Par la suite nous tenterons, dans une partie plus théorique, de prouver ces propriétés grâce à des méthodes d'analyse mathématique.

3.1 Observations avec Python

3.1.1 Loi Normale

Pour la loi normale, j'ai réalisé un programme permettant de tracer la valeur de t_1 pour une moyenne variant de 0 à 300 et un σ fixé, puis la même chose mais avec une moyenne fixé et un écart type σ variant de 0 à 200. Pour les deux études, j'ai gardé des bornes fixes, en l'occurrence [0,300]. J'ai obtenu les courbes suivantes :

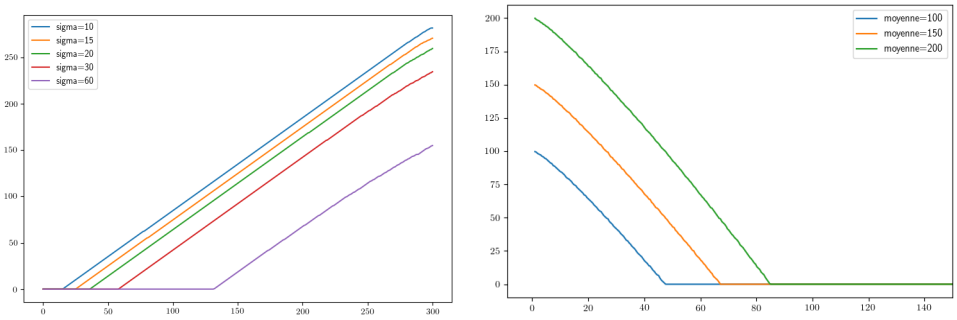

FIGURE 17 – Évolution de t_1 en fonction de μ pour différentes valeurs de σ fixées (gauche) et évolution de t_1 en fonction de σ pour différentes valeurs de μ fixées pour la loi normale

Bien qu'il soit difficile d'observer une tendance particulière pour $t_1 = f(\sigma)$, t_1 semble en revanche évoluer linéairement en fonction de la moyenne. Pour cela, il suffit d'ignorer les effets de bord. En effet, pour des moyennes proches de 0, la courbe de la fonction à densité va "déborder" des bornes, or on a réalisé une troncature en 0. En réalité, il y a donc une probabilité élevée que le temps d'exécution soit de 0, d'où un premier temps de réservation de 0.
De plus, sur ces parties linéaires, les droites sont parallèles : cela signifie que le coefficient directeur est le même peu importe la valeur de σ.
On peut alors réaliser une première conjecture :
$t_1 = g(\mu) + h(\sigma)$ avec $g(\mu) = a * \mu$, a étant une constante. En addition, on remarque que ce coefficient directeur est égal à 1. On peut donc améliorer la conjecture, et se poser dans le cas où $t_1 = \mu + h(\sigma)$ L'objectif reste alors de déterminer la forme de cette fonction $h(\sigma)$ grâce aux propriétés qu'on a sur la fonction de coût. Je développerai cela dans la partie théorique.

3.1.2 Loi exponentielle

J'ai réalisé les mêmes expériences pour la loi normale, mais avec la loi exponentielle. Cependant, celle-ci ne dépendant que d'un paramètre $\theta = \frac{1}{\mu}$, on ne peut tracer qu'une courbe, en faisant évoluer la moyenne sur l'intervalle [0,50] (au-delà, les bornes [0,300] deviennent trop petites pour l'étude).

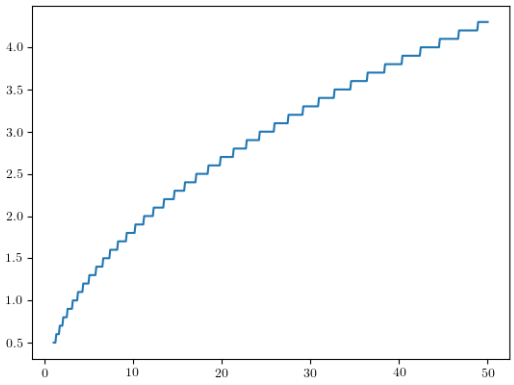

FIGURE 18 – Évolution de t_1 en fonction de la moyenne pour la loi exponentielle

On obtient une courbe en escalier visiblement non linéaire, mais qui semble cependant suivre une fonction particulière. Nous tâcherons de déterminer cette fonction dans la partie suivante. On peut cependant faire une première observation qui confirme la robustesse de la stratégie pour la fonction exponentielle : Puisqu'on obtient un escalier, cela signifie qu'autour d'une valeur de μ, pour de petites variations, la première réservation ne varie pas et donc, comme on l'expliquera plus tard, le coût de la stratégie non plus. Cela est particulièrement le cas pour de grandes valeurs de μ, confirmant nos observations précédentes.

3.2 Propriétés et démonstrations

Dans cette partie, je vais me concentrer sur la démonstration de certains résultats obtenus lors des observations précédentes. Cette partie est cependant incomplète car, à l'heure de la remise de ce rapport, nous travaillons encore sur certaines preuves et démonstrations. Je vais cependant expliquer les avancées faites, les différentes tentatives, et ce qui est prévu pour la suite.

3.2.1 Logique des preuves

Avant d'expliquer les graphiques observés, le but des démonstrations est de prouver la robustesse des stratégies de réservation. Il faut donc préciser les étapes logiques du raisonnement.
Premièrement, grâce aux intuitions donnée par les expérimentations, on tentera de trouver des propriétés sur le premier temps de réservation optimal t_1.

Grâce à ses propriétés, on majorera l'erreur sur t_1 par rapport à l'erreur sur la moyenne. Par la suite, il s'agira de trouver une relation entre t_1 et les autres temps de réservation démontrant que majorer l'erreur sur t_1, c'est majorer le dépassement de coût par rapport à la stratégie optimale.
Une seconde stratégie est d'utiliser les propriétés connues du premier temps de réservation optimale afin d'en déduire une majoration du coût optimal. Plus exactement, on va majorer la dérivée de la fonction de coût pour majorer le coût en lui même, notamment grâce au théorème des accroissements finis. Nous nous focaliserons sur cette méthode.

Pour commencer, rappelons la forme de la fonction donnant l'espérance du coût d'une stratégie :

$$E(C(X)) = t_1 + C + \sum_{i=2}^{n}(1 - F(t_{i-1})) * (R + (t_i - t_{i-1}) + C)$$

Intervient alors une propriété fondamentale : en notant t_i^o le i^{eme} temps de réservation optimal, on a :

$$\frac{\partial E(C(X))}{\partial t_i}(t_i^o) = 0$$

Cette propriété se comprend assez facilement : en effet, puisque le t_i optimale doit minimiser le coût de la stratégie, il constitue un minimum local pour la fonction de coût et en annule donc la dérivée.
En dérivant la fonction d'espérance du coût en fonction de t_1, on peut ainsi obtenir une relation entre t_1^o et t_2^o.

$$1 - f(t_1).(R + (t_2 - t_1) + C) - \mathcal{P}(X > t_1) = 0$$

D'où :

$$t_2 = t_1 + \frac{F(t_1)}{f(t_1)} - (R + C)$$

De la même manière, on obtient t_3^o,...,t_n^o de proche en proche. Il est donc bien possible d'obtenir l'ensemble de la stratégie à partir du premier temps de réservation. Il s'agit cependant d'une équation difficile à manipuler, particulièrement pour la loi normale dont la fonction de répartition F est construite à partir de la fonction erf, uniquement calculable à partir d'une intégrale.

3.2.2 Loi normale

Comme dit précédemment, il est très compliqué de démontrer la linéarité de t_1 en fonction de la moyenne. Cependant, en s'appuyant sur la conjecture

faite et en observant que pour toutes les droites le coefficient directeur est égal à 1, on peut en déduire un théorème.

Theorem 1. *Soit une stratégie composée de temps de réservation, de temps de restart et de temps de Checkpoint, établie pour des temps d'exécution suivant une loi normale* $\mathcal{N}(\mu, \sigma^2)$.
On la notera $((r_1, t_1, c_1), (r_2, t_2, c_2), ..., (r_n, t_n, c_n))$.
Si il existe une fonction g_1 *telle que* $t_1 = \mu + g_1(\sigma)$, *alors* $\forall\, t_i \in (t_1, ...t_n)$, $\exists\, g_i$ *tq* $t_i = \mu + g_i(\sigma)$

Démonstration. La preuve de ce théorème utilise le principe de récurrence.
Initialisation : D'après la conjecture, on suppose que $t_1 = \mu + g_1(\sigma)$.
De plus, comme montré précédemment, $t_2 = t_1 + \frac{F(t_1)}{f(t_1)} - (R + C)$ D'où :

$$t_2 = \mu + g_1(\sigma) + \frac{\frac{1}{2}(1 + erf(\frac{\mu + g_1(\sigma) - \mu}{\sigma\sqrt{2}}))}{\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{\mu + g_1(\sigma) - \mu}{\sigma})^2}} - (R + C)$$

$$t_2 = \mu + g_1(\sigma) + \frac{\frac{1}{2}(1 + erf(\frac{g_1(\sigma)}{\sigma\sqrt{2}}))}{\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{g_1(\sigma)}{\sigma})^2}} - (R + C)$$

On pose $g_2(\sigma) = g_1(\sigma) + \frac{\frac{1}{2}(1+erf(\frac{g_1(\sigma)}{\sigma\sqrt{2}}))}{\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{g_1(\sigma)}{\sigma})^2}} - (R + C)$ (indépendante de μ)
On a bien $t_2 = \mu + g_2(\sigma)$

Hérédité : On suppose qu'il existe (i,i-1) tels que $t_i = \mu + g_i(\sigma)$,$t_{i-1} = \mu + g_{i-1}(\sigma)$.
Rappelons que : $E(C(X)) = t_1 + C + \mathcal{P}(X > t_1).(R + C + (t_2 - t_1)) + ... + \mathcal{P}(X > t_{i-1}).(R + C + (t_i - t_{i-1})) + \mathcal{P}(X > t_i).(R + C + (t_{i+1} - t_i)) + ... + \mathcal{P}(X > t_{n-1}).(R + C + (t_n - t_{n-1}))$.
Or, pour t_i optimal, $\frac{\partial E(C(X))}{\partial t_i}(t_i^o) = 0$. D'où en dérivant et en isolant t_{i+1} :

$$t_{i+1} = t_i + \frac{F(t_i) - F(t_{i-1})}{f(t_i)} - (R + C)$$

D'après l'hypothèse d'hérédité :

$$t_{i+1} = \mu + g_i(\sigma) + \frac{\frac{1}{2}(1 + erf(\frac{\mu + g_i(\sigma) - \mu}{\sigma\sqrt{2}})) - \frac{1}{2}(1 + erf(\frac{\mu + g_{i-1}(\sigma) - \mu}{\sigma\sqrt{2}}))}{\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{\mu + g_i(\sigma) - \mu}{\sigma})^2}} - (R + C)$$

Les μ s'éliminent et on a :

$$t_{i+1} = \mu + g_i(\sigma) + \frac{\frac{1}{2}(1+erf(\frac{g_i(\sigma)}{\sigma\sqrt{2}})) - \frac{1}{2}(1+erf(\frac{g_{i-1}(\sigma)}{\sigma\sqrt{2}}))}{\frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{1}{2}(\frac{g_i(\sigma)}{\sigma})^2}} - (R+C)$$

On pose $g_{i+1}(\sigma) = g_i(\sigma) + \frac{\frac{1}{2}(1+erf(\frac{g_i(\sigma)}{\sigma\sqrt{2}}))-\frac{1}{2}(1+erf(\frac{g_{i-1}(\sigma)}{\sigma\sqrt{2}}))}{\frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{1}{2}(\frac{g_i(\sigma)}{\sigma})^2}} - (R+C)$, ce qui est bien indépendant de μ.
Donc $\exists\ g_{i+1}$ telle que $t_{i+1} = \mu + g_{i+1}(\sigma)$.

Conclusion : La propriété est vérifiée au rang 1 et 2 et est héréditaire, donc elle est vraie. □

3.2.3 Loi exponentielle

La loi exponentielle est bien plus simple à étudier, puisqu'on connait la fonction à densité et la fonction de répartition et qu'on peut les analyser. De plus, on dispose d'une autre propriété qui facilitera les calculs :

$$Pour\ X \hookrightarrow \mathcal{E}(\theta),\quad \forall(t,T),\quad \mathcal{P}(X>t) = \mathcal{P}_{(X>T)}(X > T+t)$$

Concrètement cela signifie qu'on aura $t_1 = (t_2 - t_1) = ... = (t_n - t_{n-1})$ pour la stratégie optimale. Ce théorème a été prouvé par l'équipe lors de travaux précédents [4].
D'où :

$$E(C(X)) = \sum_{i=0}^{n} \mathcal{P}(X > i.t_1)(R + C + t_1)$$

Et donc :

$$E(C(X)) = (R + C + t_1)\ \sum_{i=0}^{n} e^{-\lambda i t_1}$$

On reconnait une série géométrique, et on a :

$$E(C(X)) = (R + C + t_1).\frac{1}{1 - e^{-\lambda t_1}}$$

On dérive l'équation en fonction de t_1 pour trouver une relation entre λ et le t_1 optimal. En utilisant la propriété stipulant que $\frac{\partial E(C(X))}{\partial t_i}(t_i^o) = 0$ et en réalisant les transformations nécessaires, on obtient la relation suivante :

$$(-1 - \lambda(R + C + t_1))e^{-1-\lambda(R+C+t_1)} = -e^{-1-\lambda(R+C)}$$

En posant $y = -1 - \lambda(R + C + t_1)$ et $x = -e^{-1-\lambda(R+C)}$, on a donc l'équation suivante : $ye^y = x$.
On reconnait une équation qui se résout par une fonction de Lambert, en posant $y = \mathcal{W}(x)$.
Ici, $x \in [-\frac{1}{e}, 0]$ on doit donc utiliser la branche secondaire de la fonction $\mathcal{W}$, à savoir $\mathcal{W}_{-1}$. On en déduit donc la relation suivante :

$$-1 - \lambda(R + C + t_1) = \mathcal{W}_{-1}(-e^{-1-\lambda(R+C)})$$

On isole t_1, qui correspond donc au premier temps de réservation de la stratégie optimale pour une loi exponentielle de paramètre λ.

$$t_1^o = \frac{-\mathcal{W}_{-1}(-e^{-1-\lambda(R+C)}) - 1}{\lambda} - (R + C)$$

Etant donné que $\lambda = \frac{1}{\mu}$, on peut donc déterminer t_1 pour n'importe quelle moyenne. Pour vérifier cette relation, j'ai tracé la fonction sur Python et l'ai comparée à mes résultats expérimentaux :

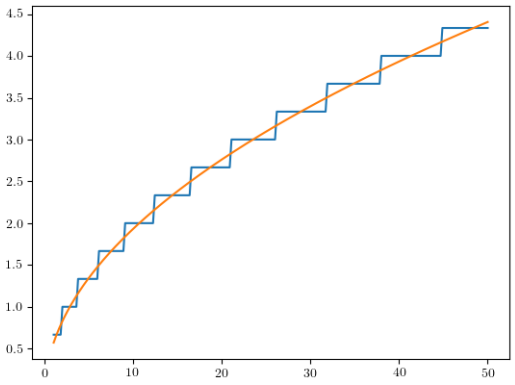

FIGURE 19 – t_1 en fonction de la moyenne avec en bleu les résultats expérimentaux et en orange la courbe théorique

On constate que la courbe expérimentale de $t_1 = f(m)$ qu'on avait tracé grâce à l'algorithme suit bien la courbe théorique.
En trouvant une relation direct entre t_1 et la moyenne, l'idée est par la suite de trouver une implication du type : $|\lambda_a - \lambda| < \epsilon_1 \Rightarrow |t_1 - t_1^o| < \epsilon_2$ avec λ le paramètre de la loi exponentielle régissant les temps d'exécution, λ_a la valeur de l'estimation de ce paramètre, t_1 le premier temps de réservation de la stratégie obtenue grâce au tirage et t_1^o le premier temps de réservation optimal. Autrement dit, montrer que majorer l'erreur sur le paramètre

de la loi exponentielle permet de majorer l'erreur sur le temps de réservation.

4 Conclusion

4.1 Connaissances acquises

Lors de mon stage, j'ai principalement pu développer mes connaissances en mathématiques dans le domaine des probabilités. J'ai manipulé des lois que je n'avais pas étudier avant et j'ai utilisé des méthodes d'analyse que je n'avais pas l'habitude d'employer.
La méthode de recherche était très intéressante, puisqu'elle s'appuyait sur des résultats expérimentaux pour chercher des propriétés à démontrer, concrètement à ce qu'on à l'habitude de faire en cours de manière plus conventionnelle.
J'ai également amélioré mes techniques de programmation en Python, et j'ai pu faire face à des problèmes de complexité algorithmique qui ne s'étaient pas posés avant.
Enfin, j'ai utilisé de nombreux logiciels, que ce soit pour le partage de travaux dans une équipe de chercheurs, la création de graphique, la programmation, etc...

4.2 Résultats obtenus et continuité

Les résultats expérimentaux obtenus grâce à des tests sur Python ont permis de montrer la robustesse de la stratégie d'ordonnancement étudiée, même lorsqu'on ne connait pas la loi régissant les tâches, et même avec un échantillon de taille minimale.
Les propriétés démontrées sur la loi normale permettrons peut être par la suite de prouver plus exactement la robustesse des stratégies pour celle ci.
Pour la loi exponentielle, je suis actuellement entrain d'utiliser les relations et propriétés obtenus pour majorer l'espérance du coût de la stratégie obtenue par approximation.
Dans la suite de mon stage, je vais donc d'abord poursuivre la réalisation de mes preuves pour la loi exponentielle. Par la suite, je pourrais continuer à travailler sur la loi normale, notamment en étudiant les propriétés de erf, et peut être travailler sur la loi de Weibull qui n'a pas encore été abordée dans la partie théorique.

5 annexe

5.1 Codes Python

Vous pouvez accéder sur le lien suivant à l'ensemble des codes Python réalisés durant le stage :
https://gitlab.inria.fr/lweisbec/codes-robustesse-de-strategies-d-ordonnancement

5.2 Bibliographie

Références

[1] Modeliser une distribution avec python. https://www.stat4decision.com/fr/distribution-donnees-python/.

[2] Objectives and performance contract 2019-2023 – between the french government and inria. https://www.inria.fr/sites/default/files/2020-02/VDEF_COP_INRIA_2019-2023_version-CS-2019-11-27-EN%20%281%29.pdf, 2019.

[3] Ana Gainaru, Brice Goglin, Valentin Honoré and Guillaume Pallez (Aupy). Profiles of upcoming hpc applications and their impact on reservation strategies. https://people.bordeaux.inria.fr/gaupy/ressources/pub/journals/stochastic_model.pdf.

[4] Ana Gainaru, Brice Goglin, Valentin Honoré, Guillaume Pallez, Padma Raghavan, Yves Robert, Hongyang Su. Reservation and checkpointing strategies for stochastic jobs (extended version). https://hal.inria.fr/hal-02328013v2/document.

[5] Ana Gainaru, Guillaume Pallez (Aupy). Making speculative scheduling robust to incomplete data. https://people.bordeaux.inria.fr/gaupy/ressources/pub/confs/scala19_robustness.pdf.

[6] Guillaume Aupy, Ana Gainaru, Valentin Honoré, Padma Raghavan, Yves Robert, and Hongyang Sun. Reservation and checkpointing strategies for stochastic jobs. In *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 166–175, 2019.

[7] Frank G. Lether. Elementary approximation for erf(x). *Journal of Quantitative Spectroscopy and Radiative Transfer*, 49(5) :573–577, 1993.

[8] E. Morice. Quelques problèmes d'estimation relatifs à la loi de weibull. *Revue de Statistique Appliquée*, 16(3) :43–63, 1968.

[9] Wikipédia. Programmation dynamique — wikipédia, l'encyclopédie libre. http://fr.wikipedia.org/w/index.php?title=Programmation_dynamique&oldid=182806672, 2021.

[10] Wikipedia contributors. Lambert w function — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Lambert_W_function&oldid=1025357246, 2021. [Online ; accessed 10-June-2021].