Techniques d'ordonnancement pour les applications stochastiques sur plateformes HPC

Valentin Honoré

Compas'2020 : Parallélisme / Architecture/ Système/ Temps Réel
MILC - Lyon, France, du 30 juin au 3 juillet 2020

Techniques d'ordonnancement pour les applications stochastiques sur plateformes HPC

Valentin Honoré

Université de Bordeaux LaBRI - Inria BSO,
200 Avenue de la Vieille Tour
33405 - Talence
valentin.honore@inria.fr

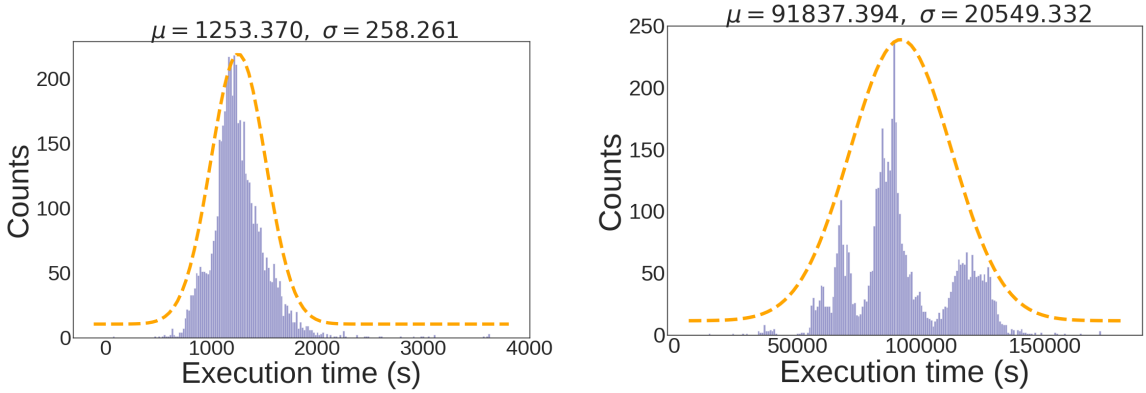
Résumé
Dans ce papier, nous résumons nos récents travaux portant sur des stratégies d'ordonnancement pour des applications dont le temps d'exécution est difficile à estimer et s'exécutant sur des plateformes orientées calcul haute-performance (HPC). Ces applications, dites stochastiques ou irrégulières, font partie des applications de "seconde génération". Elles émergent de domaines tels que le *Big Data*, *Machine Learning* ou les neurosciences. Ces applications ayant des besoins de plus en plus importants en terme de calcul, les plateformes HPC deviennent un hôte de prédilection pour exécuter ces applications. Dans ce papier, nous présentons tout d'abord les contraintes et problèmes posés par ces applications dans un environnement HPC. Ensuite nous décrirons des stratégies d'ordonnancement efficaces pour ces applications. Enfin, nous discuterons de la combinaison de techniques de sauvegarde en parallèle de ces stratégies d'ordonnancement et donnerons des perspestives de recherches futures.

Mots-clés : ordonnancement, convergence HPC-BigData, tâches stochastiques, plateforme HPC, points de sauvegarde

---

1. Introduction

L'ordonnancement typique d'une application sur une plateforme HPC consiste à réaliser une réservation de ressources de calcul sur cette plateforme pour une durée déterminée, disons t_1 secondes. Il suffit ensuite de lancer l'application sur les ressources réservées jusqu'à ce que soit l'application termine son exécution, soit la réservation est arrivée à son terme. Pour certaines applications, la durée d'exécution peut être estimée assez précisément. En revanche, il y a des profils d'applications pour lesquels on ne peut estimer ce temps d'exécution de manière précise. La Figure 1 montre deux exemples de telles applications. On peut ainsi voir une variation très grande du temps d'exécution en fonction de l'entrée de l'application. Pour ces applications, l'utilisateur doit alors estimer une valeur pour la première réservation t_1.
En régime normal, deux cas se présentent quand on lance l'application : soit celle-ci se termine en un temps $t \leq t_1$ et l'exécution a réussi, soit l'application a un temps d'exécution $t > t_1$ et l'éxécution a échoué. Dans ce cas, l'utilisateur doit faire une nouvelle réservation de taille $t_2 > t_1$. L'utilisateur devra itérer ce procédé jusqu'à ce qu'une réservation permette à l'application de se terminer correctement. Le coût total pour l'utilisateur est alors la somme des coûts de chaque réservation réalisée jusqu'à accomplissement de l'exécution de l'application.



(a) Voxel-based morphometry quality assurance (VBMQA) [7]

(b) Diffusion tensor imaging quality assurance (dtiQA) [6]

FIGURE 1 – Traces sur plus de 5000 exécutions (histogrammes en violet) de Juillet 2013 à Octobre 2016 de deux applications de neurosciences, obtenues à partir de la base de données d'imagerie médicale de l'Université Vanderbilt [5]. Les données sont ensuite extrapolées en une distribution LogNormal (ligne en pointillés orange).

Ainsi, l'utilisateur doit déterminer une stratégie d'ordonnancement consistant en une séquence de réservations. Ces stratégies peuvent être assez simples. Par exemple, un utilisateur pourrait déterminer une première réservation t_1 puis réaliser des réservations successives en augmentant d'un facteur donné la dernière réservation ayant échouée. Une autre stratégie pourrait être de réserver dès la première réservation un valeur garantissant le succès de l'exécution de l'application (par exemple, en réservant au moins le $99^{\text{ème}}$ quantile de la distribution du temps d'exécution). Néanmoins, ces stratégies simples sont souvent coûteuses et inefficaces.
La notion de coût est ici totalement générique. Cela peut représenter aussi bien un budget (coût pécunier en fonction de la durée de réservation des ressources de calcul) qu'une durée (le temps d'attente avant que les ressources soit effectivement attribuées à l'utilisateur). De même, l'approche par réservation est agnostique du type de tâche (séquentielle/parallèle) et du type de ressources de calcul demandé (processeurs, machines virtuelles etc).
Nous proposons d'utiliser ici une fonction de coût générique permettant d'englober une grande variété de scénarios. Ainsi, pour une réservation de durée t_i et un temps d'éxécution réelle de l'application t, le coût est exprimé par

$$\alpha t_i + \beta \min(t_i, t) + \gamma \tag{1}$$

où α, β et γ sont des paramètres constants dépendant soit de la plateforme soit de l'instanciation du modèle de coût. En instanciant $\alpha = 1, \beta = \gamma = 0$, nous obtenons par exemple le modèle des "Instances Réservées" sur Amazon AWS [1], où l'utilisateur ne paie que ce qu'il utilise [1]. Une instanciation suivant un modèle HPC plus classique peut également être réalisée, où le coût est proportionnel à la durée de temps effective de l'application ainsi qu'au temps d'attente dans la file d'attente de l'ordonnanceur. Dans ce cas, la fonction $\alpha t_i + \gamma$ représente le temps passé dans la file de l'ordonnanceur avant l'attribution des ressources de calcul et le début du calcul.

1. D'après [1], cette méthode est jusqu'à 75% plus économe que les instances à la demande sur Amazon AWS.

Même si le temps d'exécution de ces applications n'est pas connu à l'avance, nous considérons que chaque exécution d'une même application sur une entrée différente représente une tâche d'un même type. De plus, le temps d'exécution de ces tâches obéit à une loi de probabilité connue, comme présenté dans la Figure 1. Une importante hypothèse est que chaque tâche est déterministe : pour une même entrée, l'application aura toujours un temps d'éxécution identique. Néanmoins, ce temps n'est pas connu jusqu'à ce que la tâche soit terminée. Ainsi, le temps d'exécution d'une tâche est aléatoirement et uniformément tiré d'une loi de probabilité cible. Remarquons que le fait que chaque tâche soit déterministe implique qu'une séquence de réservations minimisant une fonction de coût donnée est strictement croissante.
Dans ce travail, nous nous intéressons à des distributions continues [2]. Concernant ces dernières, nous ne considérons seulement celles ayant un domaine de définition inclus dans $[0, \infty)$ car le temps d'exécution d'une application ne peut qu'être positif (cela exclut par exemple la loi normale).
Dans la section suivante, nous allons présenter des stratégies de réservation efficaces pour les applications stochastiques.

2. Stratégies de réservation pour les applications irrégulières

Dans cette section, nous présentons des stratégies d'ordonnancement en réponse au problème suivant : étant donné une tâche dont le temps d'exécution est donné par la variable aléatoire X, déterminer une séquence de réservations (possiblement infinie) $S = (t_1, t_2, \cdots, t_i, t_{i+1}, \cdots)$ telle que l'espérance du coût pour exécuter cette tâche est minimisée. D'un point de vue théorique, il n'est pas clair qu'il existe toujours une séquence de réservations ayant une espérance de coût finie. Néanmoins, nous montrons dans cette section que cette assertion est vraie pour les distributions à moyenne et variance finie, ce qui est le cas de toutes les distributions que nous considérons ici.

2.1. Modèle de coût et tâches stochastiques

Nous considérons donc des tâches stochastiques déterministe, uniformément et aléatoirement tirées d'une distribution de probabilité suivant une loi $\mathcal{D}$, ayant une densité de probabilité f, une fonction de répartition F, et un domaine de définition inclut dans $[0, \infty)$. Le temps d'exécution d'une tâche est une variable aléatoire X, et $\mathbb{P}(X \leq T) = F(T) = \int_a^T f(t)dt$.
Pour une séquence S donnée, et une tâche avec un temps d'exécution t, le coût de S est égal à

$$C(k, t) = \sum_{i=1}^{k-1} (\alpha t_i + \beta t_i + \gamma) + \alpha t_k + \beta t + \gamma \tag{2}$$

où k est le plus petit indice dans S tel que $t \leq t_k$. Pour simplifer les sommations, nous introduisons $t_0 = 0$ dans la suite au début de chaque séquence de réservations.

2.2. Vers une solution pour les distribution continues

Dans cette section, nous définissons le problème d'optimisation et sa solution dans le cas de distributions continues. Toutes les preuves des théorèmes ainsi que les détails des calculs présentés dans cette section sont disponibles dans [2].

2. Nous nous concentrons sur des distributions continues usuelles (LogNormal, Uniform, Beta, Weibull, etc). Les résultats présentés ici restent valident dans la généralité des distributions continues candidates. De même, [2] et [3] proposent des solutions exactes pour des distributions discrètes par des algorithmes de programmation dynamique.

2.2.1. Problème d'optimisation

L'objectif est de trouver une séquence de réservations strictement croissante qui minimize l'espérance du coût. Plus formellement, cette espérance pour une séquence $S = (t_1, t_2, \dots, t_i, t_{i+1}, \dots)$ s'écrit comme :

$$\mathbb{E}(S) = \sum_{k=1}^{\infty} \int_{t_{k-1}}^{t_k} C(k,t) f(t) dt = \beta \cdot \mathbb{E}[X] + \sum_{i=0}^{\infty} (\alpha t_{i+1} + \beta t_i + \gamma) \, \mathbb{P}(X \geq t_i) \tag{3}$$

avec $C(k,t)$ tel qu'exprimé dans l'Equation (2).
Nous pouvons désormais écrire le problème d'optimisation à résoudre :

Définition 1 (STOCHASTIQUE)**.** Etant donné une distribution de probabilité (avec fonction de répartition F) pour le temps d'exécution de tâches stochastiques, et étant donné une fonction de coût donnée par l'Equation (1) (ayant pour paramètres α, β et γ), trouver une séquence de réservations S minimisant l'espérance de coût $\mathbb{E}(S)$ donnée par l'Equation (3).

2.2.2. Caractérisation d'une solution optimale pour les distributions continues

Nous proposons dans cette section une caractérisation d'une solution optimale pour STOCHASTIQUE.

Théorème 1. *Pour une distribution continue à moyenne et variance finies, et à support infini, résoudre* STOCHASTIQUE *se réduit à trouver* t_1^o *qui minimise*

$$\sum_{i=0}^{\infty} (\alpha t_{i+1} + \beta t_i + \gamma) \, \mathbb{P}(X \geq t_i) \;\; \textit{avec } t_0^o = 0$$

et pour tout $i \geq 2$*,*

$$t_i^o = \frac{1 - F(t_{i-2}^o)}{f(t_{i-1}^o)} + \frac{\beta}{\alpha} \left(\frac{1 - F(t_{i-1}^o)}{f(t_{i-1}^o)} - t_{i-1}^o \right) - \frac{\gamma}{\alpha} \tag{4}$$

Pour une distribution continue à moyenne et variance finies, mais à support fini, la récurrence de l'Equation (4) *est toujours valable mais s'arrête dès qu'elle atteint* t_i^o *avec* $F(t_i^o) = 1$*.*

Le Théorème 1 nous fournit ainsi une formule récursive pour trouver les différentes réservations, à l'exception de la première, t_1^o, qu'il reste à déterminer. Ce théorème nous suggère qu'une séquence optimale est uniquement caractérisée par sa valeur en t_1^o. Néanmoins, calculer la valeur de t_1^o est un problème difficile à l'exception de certaines distributions telles que la loi Uniforme [2].
Néanmoins, nous sommes capables de borner cette valeur comme nous le montre le Théorème 2.

Théorème 2. *Pour une distribution* $\mathcal{D}$ *à support fini* $[a, \infty)$ *tel que* $\mathbb{E}[X^2] < \infty$*, la valeur* t_1^o *d'une séquence optimale* $S^o = (t_1^o, t_2^o, \dots, t_i^o, t_{i+1}^o, \dots)$ *satisfait* $t_1^o \leq A_1$*, et* $\mathbb{E}(S^o) \leq A_2$*, où*

$$A_1 = \mathbb{E}[X] + 1 + \frac{\alpha+\beta}{2\alpha}(\mathbb{E}[X^2] - a^2) + \frac{\alpha+\beta+\gamma}{\alpha}(\mathbb{E}[X] - a) \quad \textit{et} \quad A_2 = \beta \cdot \mathbb{E}(X) + \alpha A_1 + \gamma$$

Ainsi, notre solution pour STOCHASTIQUE est sub-optimale car nous ne pouvons pas calculer de manière exacte la valeur de t_1^o. En pratique, nous avons proposé dans [2] une procédure appelée BRUTE-FORCE qui teste différentes valeurs candidates pour t_1^o et estime l'espérance du coût associé à chacune de ces valeurs par un processus de Monte-Carlo pour ne garder que la séquence qui la minimize.
Des comparatifs de performance sont effectués dans [2] entre cette solution et d'autres stratégies plus simples. Notre solution surpasse les autres stratégies et montre son possible bénéfice aux utilisateurs.

3. De la sauvegarde du progrès de l'application dans les stratégies de réservation

Dans cette section, nous nous intéressons à étendre les travaux de la Section 2 en ajoutant la possibilité d'effectuer un point de sauvegarde (PdS) à la fin de certaines réservations judicieusement choisies, pour les applications disposant de cette possibilité. En effet, on peut remarquer que le travail effectué dans une réservation qui a échoué est perdu. Ainsi, prendre un PdS avant la fin de cette réservation permettrait de sauvegarder le progrès effectué et ainsi de recommencer la prochaine réservation à l'endroit où le dernier PdS a été pris.
L'idée des PdS est une approche naturelle très utilisée en pratique, qui plus est pour des tâches durant plusieurs heures. Néanmoins, cela complique drastiquement la conception de stratégies d'ordonnancement. Les approches à l'heure actuelles sauvegardent soit toutes les réservations (stratégie utilisée pour les grosses applications) soit aucune d'entre elles. De plus, cela implique qu'une réservation peut dédier une partie de son temps à sauvegarder l'état actuel de la tâche, et ainsi ne pas utiliser ce temps pour du calcul. Ici, on voit clairement apparaître un compromis entre le temps effectif de calcul dans une réservation et le temps dédié à une sauvegarde pour ne pas perdre le travail non-encore sauvegardé. Évidemment, le temps nécessaire à réaliser un PdS aura un fort impact sur la décision de l'activer ou non.

3.1. Modèle de coût et d'application avec points de sauvegarde

Dans cette section, nous présentons les modèles de coût et d'application pour des séquences de réservations avec points de sauvegarde.
Nous utilisons le même modèle d'application présenté dans la Section 2.1. L'application suit une loi de probabilité $\mathcal{D}$ selon les mêmes conditions que précédemment. Nous considérons de plus que l'application peut être interrompue à n'importe quel moment (applications à travail divisible) pour initier un PdS. Ce dernier sauvegarde le progrès courant de l'application, et permet de reprendre l'exécution de cette dernière au même point. Nous faisons également l'hypothèse que la durée d'un PdS (notée C) ainsi que le coût pour redémarrer à partir d'un PdS en début de réservation (noté R) sont constants tout au long de l'exécution de l'application.
Nous employons également le modèle de coût générique présenté dans l'Equation (1). Nous rajoutons simplement la prise en compte des éventuels PdS. Ainsi, au début de la réservation i, un coût de redémarrage R sera appliqué à partir du PdS le plus récent effectué dans les réservations d'indice $1, \cdots, i-1$, si applicable. Ainsi, l'utilisateur génère une séquence de réservations suivant la Définition 2.

Définition 2 (Séquence de réservations pour $\mathcal{D}$)**.** Etant donné une distribution de probabilité $\mathcal{D}$, une *séquence de réservations* $\mathcal{S} = \{(W_1, \delta_1), (W_2, \delta_2), \ldots\}$, est définie comme une séquence de tailles de réservation W_k et une suite de décisions de PdS $\delta_k \in \{0, 1\} : \delta_k = 1$ voulant dire que la $k^{\text{ième}}$ réservation se termine par un PdS, et $\delta_k = 0$ signifiant qu'il n'y aura pas de PdS à cette réservation.

La $k^{\text{ième}}$ réservation peut se décomposer comme $W_k = R_k + T_k + C_k$ avec $R_k = (1 - \prod_{i=1}^{k-1}(1-\delta_i))R$ le temps de redémarrage [3], T_k la durée pendant laquelle la tâche sera exécutée et $C_k = \delta_k C_k$ la durée d'un éventuel PdS. T_k ne représentant que le temps passé en calcul lors de la réservation W_k, nous introduisons une nouvelle notation, t_k qui représente le progrès de l'exécution de la tâche depuis son commencement [4].

3. En posant $R_1 = 0$.
4. La relation entre t_k et T_k est $t_k = T_k + \sum_{i=1}^{k-1} \delta_i T_i$, $T_k = t_k - \sum_{i<k} \delta_i T_i$. Se référer à l'Appendix A pour une explication détaillée.

Concernant le fonction de coût, étant donnée une séquence de reservations $\mathcal{S} = ((t_i, \delta_i))_i$ et une durée d'exécution finale t telle que $t_{k-1} < t \leq t_k$, le coût de $\mathcal{S}$ est defini par :

$$C_{\mathcal{S}}(k, t) = \sum_{i=1}^{k-1} (\alpha W_i + \beta W_i + \gamma) + \alpha W_k + \beta(R_k + t - (t_k - T_k)) + \gamma \quad (5)$$

Suivant le même principe que précédemment, le Théorème 4 décrit dans l'Annexe B présente l'espérance du coût pour une séquence de réservations donnée. De même, la Définition 3 présente le problème d'optimisation considéré.

3.2. Une $(1+\varepsilon)$-approximation pour les distributions continues bornées

Avec les modèles présentés dans la section ci-dessus, nous sommes cette fois en mesure de proposer un schéma d'approximation entièrement en temps polynomial, énoncé par le Théorème 3. C'est un résultat beaucoup plus significatif en terme d'approche de la solution exacte que la solution sans PdS de la Section 2. Tous les détails des preuves se trouvent dans [3].

Théorème 3. *Etant donné une variable aléatoire continue* X *sur un support* $[a, b]$*, où* $0 \leq a < b$*, une constante* $\varepsilon > 0$*,* DYN-PROG-COUNT(X, ε) *est une* $(1+\varepsilon)$*-approximation pour* STOCHASTIQUE *et s'exécute avec une complexité* $\mathcal{O}\left(\frac{1}{\varepsilon^3}\right)$.

L'algorithme DYN-PROG-COUNT est décrit en Annexe C. Nous espérons pouvoir produire un résultat similaire pour les distributions continues non-bornées.
Il est évident que les coûts C et R influencent grandement les séquences produites. Un coup très faible (reciproquement fort) incitera à sauvegarder toutes (réciproquement aucune [5]) réservation(s).

4. Perspectives

Ce travail offre un grand nombre de perspectives de recherche. Notre modèle de coût est flexible et adapté à de nombreuses plateformes (*cloud*, HPC) et à une grande variété de profils d'application. Certains paramètres peuvent être difficile à estimer, notamment la fonction du temps passé dans la file d'attente de l'ordonnanceur HPC en fonction de la taille de la réservation. Néanmoins, il est important de noter que peu de données sont nécessaires sur une application elle-même pour utiliser efficacement ces stratégies d'ordonnancement [4].
Il demeure qu'il est difficile pour les utilisateurs d'applications stochastiques d'utiliser de telles stratégies, notamment du fait que certains n'ont pas l'expertise technique nécessaire pour interagir avec les ordonnanceurs des machines HPC. Une solution serait d'ajouter un module à cet ordonnanceur (par exemple, *SLURM*) afin qu'il puisse automatiquement appliquer ces stratégies en fonction du profil de l'application soumise. Comment différencier les applications stochastiques des applications régulières à la soumission ? A quel niveau doivent-être gérées ces stratégies (ordonnanceur, utilisateur, administrateur système) ? Toutes ces considérations pratiques sont très intéressantes à discuter avec la communauté HPC et les utilisateurs d'applications stochastiques.
Des pistes de recherche plus fondamentales demeurent également. Une piste pour encore améliorer nos stratégies est de ne plus considérer un coût de PdS fixe mais variable en fonction du moment où il se situe dans l'application. C'est une modélisation plus réaliste du comportement des applications. Il va de soit que cela complique encore plus fortement le problème à résoudre.

5. Dans ce cas, la solution fournie se confond avec la solution sans considération de point de sauvegarde de la Section 2.

Remerciements

Ce résumé de travaux est basé sur les travaux [2, 3] réalisés avec Ana Gainaru, Brice Goglin, Guillaume Pallez, Padma Raghavan, Yves Robert et Hongyang Sun. L'auteur tient aussi à remercier Bennett Landman et le laboratoire MASI de l'Université Vanderbilt pour le partage de leur base de données d'imagerie médicale utilisée pour produire les distributions de temps d'exécution des différentes applications. Ces travaux ont été en parti financés par le *National Science Foundation, grant CCF1719674* du *Vanderbilt Institutional Fund*, et par l'équipe associée Inria-Vanderbilt *Keystone*.

Bibliographie

1. Amazon. – Différents modèles de coût d'AWS. – `https://aws.amazon.com/fr/ec2/pricing/reserved-instances/`. Accédé pour la dernière fois : 02/04/2020.
2. Aupy (G.), Gainaru (A.), Honoré (V.), Raghavan (P.), Robert (Y.) et Sun (H.). – Reservation Strategies for Stochastic Jobs. – In *IPDPS 2019 - 33rd IEEE International Parallel and Distributed Processing Symposium*, pp. 166–175, Rio de Janeiro, Brazil, mai 2019. IEEE.
3. Gainaru (A.), Goglin (B.), Honoré (V.), Pallez (G.), Raghavan (P.), Robert (Y.) et Sun (H.). – Reservation and Checkpointing Strategies for Stochastic Jobs. – In *IPDPS 2020 - 34th IEEE International Parallel and Distributed Processing Symposium*, New Orleans, United States, mai 2020.
4. Gainaru (A.) et Pallez (G.). – Making Speculative Scheduling Robust to Incomplete Data. – In *ScalA19 : 10th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems*, Denver, United States, novembre 2019.
5. Harrigan (R. L.), Yvernault (B. C.), Boyd (B. D.), Damon (S. M.), Gibney (K. D.), Conrad (B. N.), Phillips (N. S.), Rogers (B. P.), Gao (Y.) et Landman (B. A.). – Vanderbilt university institute of imaging science center for computational imaging XNAT : A multimodal data archive and processing environment. *NeuroImage*, vol. 124, 2016, pp. 1097–1101.
6. Lauzon (C. B.), Asman (A. J.), Esparza (M. L.), Burns (S. S.), Fan (Q.), Gao (Y.), Anderson (A. W.), Davis (N.), Cutting (L. E.) et Landman (B. A.). – Simultaneous analysis and quality assurance for diffusion tensor imaging. – In *PloS one*, 2013.
7. Mechelli (A.), Price (C. J.), Friston (K. J.) et Ashburner (J.). – Voxel-based morphometry of the human brain : methods and applications. *Current Medical Imaging Reviews*, vol. 1, 2005, p. 105–113.

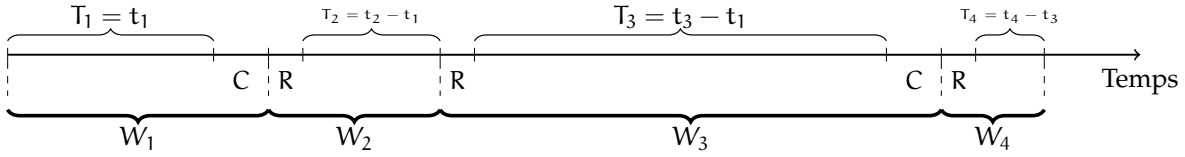


FIGURE 2 – Illustration du temps cumulé d'exécution de l'application pour la séquence de réservation $\mathcal{S} = \{(W_1, 1), (W_2, 0), (W_3, 1), (W_4, 0)\}$.

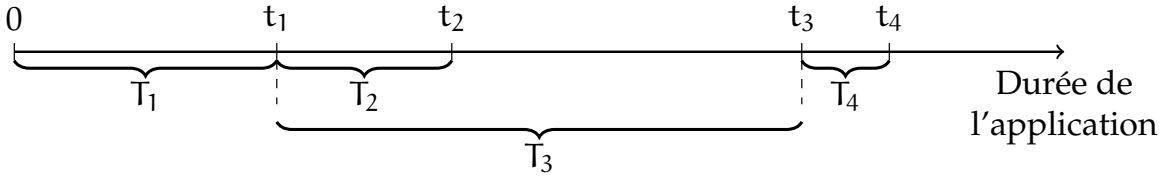


FIGURE 3 – Illustration de la notion de progrès de l'application (t_k versus T_k) pour une séquence de réservation $\mathcal{S} = \{(W_1, 1), (W_2, 0), (W_3, 1), (W_4, 0)\}$.

A. Les notations t_k et T_k dans le modèle avec points de sauvegarde

Dans cette partie, nous revenons sur la différence entre la notion de progrès de l'application à la réservation d'indice k et le temps de travail de l'application T_k dans cette mm̂e réservation. La $k^{ième}$ réservation se décompose comme suit :

$$W_k = R_k + T_k + C_k$$

avec R_k le temps nécessaire pour charger le dernier point de sauvegarde (si applicable), T_k la durée d'exécution de l'application dans cette réservation et C_k le temps de prendre un point de sauvegarde (si applicable).
Nous avons $C_k = \delta_k C$ par définition. Il n'y a un temps de rechargement uniquement s'il existe un point de sauvegarde issu d'une précédente réservation. De fait, $R_k = (1 - \prod_{i=1}^{k-1}(1 - \delta_i))R$ (en posant $R_1 = 0$ pour la première réservation). Néanmoins, la notation (W_k, δ_k) ne permet pas facilement de suivre le progrès de l'application depuis son commencement car ce dernier dépend des évènements des réservations antérieures à celle d'indice k. Considérons par exemple la séquence suivante $\mathcal{S} = \{(W_1, 1), (W_2, 0), (W_3, 1), (W_4, 0)\}$, modélisée dans la Figure 2. Si le temps d'application actuel de l'application est $X = t$, pendant quelle réservation l'application va terminer son exécution ?
Pour répondre à cette question, nous introduisons une autre façon de représenter une séquence de réservation $\mathcal{S}$ un introduisant le progrès de l'application, noté $\{t_k\}$ comme dépeint dans la Figure 3. Le progrès t_k représente la quantité de travail qui a été réalisé par l'application à la fin de la réservation d'indice k. Ainsi, la dernière réservation pour la tâche de durée t est W_k, où $t_{k-1} \leq t \leq t_k$. Evidemment, il faut nécessairement que $t \leq t_4$ pour toutes les valeurs de $\mathcal{D}$ (en d'autres termes, la borne supérieure du domaine de définition de $\mathcal{D}$ est $b \leq t_4$) afin que toutes les tâches se terminent correctement avec les 4 réservations de $\mathcal{S}$.
La relation entre t_k et T_k est $t_k = T_k + \sum_{i=1}^{k-1} \delta_i T_i$. En effet, l'application ne progresse qu'à partir du dernier point de sauvegarde effectué, alors que le travail réalisé lors de précédentes réservations non sauvegardées est définitivement perdu à partir du moment où ces réservations n'ont pas permises à l'application de se terminer correctement.

Une autre manière de représenter la relation entre t_k et T_k est :

$$t_k = T_k + \max\{t_i \mid 1 \leq i \leq k-1 \text{ et } \delta_i = 1\} \tag{6}$$

L'Equation (6) donne une formule récursive pour le calcul de t_k à partir de sa définition.
Enfin, nous récapitulons la relation entre toutes les notations liées à une réservation (W_k, δ_k) et présentée dans les Figures 2 et 3 :

$$\begin{aligned} W_k &= R_k + T_k + C_k \\ R_k &= (1 - \Pi_{i<k}(1-\delta_i))R \\ T_k &= t_k - \textstyle\sum_{i<k} \delta_i T_i \\ C_k &= \delta_k C \end{aligned}$$

Lorsque nous utilisons ce modèle de réservation, nous préférons plutôt la notation t_k à la taille de réservation W_k pour caractériser une séquence de réservations. Nous notons ainsi toute séquence de réservation $\mathcal{S} = \{(t_1, \delta_1), (t_2, \delta_2), \ldots\}$ au lieu de $\mathcal{S} = \{(W_1, \delta_1), (W_2, \delta_2), \ldots\}$. La raison est qu'il est plus aisé d'utiliser la notation de progrès de l'application dans les calculs d'espérance de coût (voir Section 3 et [3]). Par simplification des notations, nous posons $t_0 = 0$ comme l'initialisation du progrès de l'application pour toute séquence de réservation $\mathcal{S}$. Notons qu'il est possible de se restreindre uniquement aux séquences pour lesquelles $t_{k-1} < t_k$, sinon (pour $t_{k-1} = t_k$), l'application ne progresse pas durant la réservation d'indice k.

B. Fonction de coût et problème d'optimisation pour une séquence de réservations avec points de sauvegarde

Cette section présente la fonction de coût associée au problème avec points de sauvegarde ainsi que le problème d'optimisation qui en découle.
Nous commençons par définir $k(t) = k$ pour une tâche de taille t tel que $t_{k-1} < t \leq t_k$.
Pour une variable aléatoire X suivant une distribution $\mathcal{D}$, l'espérance du coût d'une séquence de réservations $\mathcal{S}$ est

$$\mathbb{E}(\mathcal{S}(X)) = \int_0^\infty C_{\mathcal{S}}(k(t), t) f(t) dt = \sum_{k=1}^{\infty} \int_{t_{k-1}}^{t_k} C_{\mathcal{S}}(k, t) f(t) dt \tag{7}$$

avec $C_{\mathcal{S}}(k(t), t)$ comme définit dans l'Equation (5). Le Théorème 4 présente une réécriture de cette fonction de coût pour une séquence de réservations $\mathcal{S} = ((t_1, \delta_1), (t_2, \delta_2), \dots)$ associée à une variable aléatoire X.

Théorème 4. *Etant donné une variable aléatoire X et une séquence de réservations $\mathcal{S} = ((t_1, \delta_1), (t_2, \delta_2), \dots)$, l'espérance du coût $\mathbb{E}(\mathcal{S}(X))$ de $\mathcal{S}$ sous les paramètres α, β et γ est donnée par :*

$$\mathbb{E}(\mathcal{S}(X)) = \beta \cdot \mathbb{E}[X] + \alpha\big(t_1 + \delta_1 C\big) + \gamma + \sum_{i=2}^{\infty} \Big(\alpha W_i + \beta\big(R_i + (1-\delta_{i-1})T_{i-1} + C_{i-1}\big) + \gamma\Big) \cdot \mathbb{P}(X > t_{i-1})$$

Enfin, la définition 3 décrit le problème d'optimisation dans le cas de séquences de réservations avec points de sauvegarde.

Définition 3 (STOCHASTIQUE)**.** Etant donné une variable aléatoire X (avec densité de probabilité f et fonction de répartition F) représentant les temps d'exécution d'une tâche stochastique, et une fonction de coût comme décrite dans l'Equation (1) (avec paramètres $\alpha > 0$ et $\beta, \gamma \geq 0$), trouver une stratégie de réservations $\mathcal{S}$ avec une espérance du coût minimale $\mathbb{E}(\mathcal{S}(X))$ comme dans l'Equation (7).

C. Algorithme de la $(1+\varepsilon)$-approximation pour séquence de réservations avec points de sauvegarde

Tout d'abord, nous introduisons le Théorème 5 qui décrit une solution exacte pour une distribution discrète $Y \sim (v_i, f_i)_{1\le i\le n}$. Cette solution se présente sous la forme d'un algorithme sofistiqué de programmation dynamique [6].

Théorème 5. *Pour une distribution discrete* $Y \sim (v_i, f_i)_{1\le i\le n}$*, l'espérance du coût optimal est retournée par* $\mathbb{E}_{\text{ckpt}}(0,0)$*, où, pour* $0 \le i_c \le i_l \le n$*,* $\mathbb{E}_{\text{ckpt}}(i_c, i_l)$ *est égal à :*

$$
\begin{aligned}
&= \beta \cdot \mathbb{E}[Y], && \text{si } i_l = n\\
&= \min_{\substack{i_l+1\le j\le n,\\ \Delta_j\in\{0,1\}}} \Bigg(\mathbb{E}_{\text{ckpt}}(\Delta_j j, j) + \Big(\alpha\big(v_j + \Delta_j C\big) + \gamma\Big) \cdot \sum_{k=i_l+1}^{n} f_k \\
&\quad + \beta\big((1-\Delta_j)v_j + \Delta_j C\big) \cdot \sum_{k=j+1}^{n} f_k \Bigg), && \text{si } i_c = 0\\
&= \min_{\substack{i_l+1\le j\le n,\\ \Delta_j\in\{0,1\}}} \Bigg(\mathbb{E}_{\text{ckpt}}\big((1-\Delta_j)i_c + \Delta_j j, j\big) \\
&\quad + \Big(\alpha\big(R + (v_j - v_{i_c}) + \Delta_j C\big) + \beta R + \gamma\Big) \cdot \sum_{k=i_l+1}^{n} f_k \\
&\quad + \beta\big((1-\Delta_j)(v_j - v_{i_c}) + \Delta_j C\big) \cdot \sum_{k=j+1}^{n} f_k \Bigg), && \text{sinon}
\end{aligned}
$$

La solution optimale peut être calculée en un temps $\mathcal{O}(n^3)$*.*

Ici, Δ_j indique si la réservation de valeur v_j sera sauvegardée ou non.
En utilisant l'algorithme pour les distributions discrètes présenté dans le Théorème 5, nous proposons l'Algorithme 1 pour une variable aléatoire X suivant une distribution continue bornée. En discrétisant X, l'algorithme permet de retourner une solution quasi-optimale.

Algorithme 1 DYN-PROG-COUNT(X, ε)

1: Soit $[a, b]$ le domaine de définition de X, avec $0 \le a < b$
2: $c_0 = 3(b-a)\min\left(\frac{1}{\min(\max(a, \varepsilon\mathbb{E}[X]/3), R, C)}, \frac{\alpha+\beta}{\gamma}\right)$
3: $n \leftarrow \lceil c_0/\varepsilon \rceil$
4: Définir la distribution discrète $Y_n \sim (v_i, f_i)_{i=1\dots n}$ telle que

$$
\begin{cases} v_i &= a + i \cdot \frac{b-a}{n} & \text{pour } 0 \le i \le n \\ f_i &= \mathbb{P}(Y_n = v_i) = \mathbb{P}(v_{i-1} < X \le v_i) & \text{pour } 1 \le i \le n \end{cases}
$$

5: $\mathcal{S}_n^{\text{dp}} \leftarrow$ Stratégie optimale Y_n (Théorème 5)
6: **Retourner** $\mathcal{S}_n^{\text{dp}}$

6. Un algorithme de programmation dynamique peut également calculer une solution optimale pour une distribution discrète dans le cas où il n'y pas de points de sauvegarde, comme dans la Section 2. Cet algorithme est décrit dans [2] et est de complexité $\mathcal{O}(n^2)$.