LOG8371- Ingénierie de la qualité en logiciel

Hiver 2023

Travaille pratique #2: Efficacité et Performance

Groupe Peel

2088821 – Laurent Quoc Hoa Nguyen

1995039 - Labib Bashir-Choudhury

1956802- Dawut Esse

1954607 - Victor Kim

2085085 - Yasser Kadimi

Soumis à : Armstrong Tita Foundjem

Date de Remise : 22 mars 2023

Table des matières

1. Plan d’assurance qualité

1.1. Introduction

MOA (Massive Online Analysis) est un framework open source pour l’apprentissage automatique en ligne et le traitement de flux de données. Ce cadriciel a été développé en Java et est largement utilisé pour l’analyse de données en temps réel et les applications d’apprentissage automatique à grande échelle. Le présent rapport fait suite au dernier rapport qui a pour objectif d’examiner la qualité du logiciel MOA. Pour être plus précis, ce document va traiter un critère primordial pour le succès d’un produit logiciel : la performance. La performance se définit comme la capacité d'un système logiciel à répondre à ses exigences fonctionnelles et non fonctionnelles avec des contraintes ou une charge de travail. Basé sur le modèle ISO/IEC 25010:2011, nous allons analyser en détail trois sous-critères de la performance :

- Comportement du temps : La mesure de la vitesse d'exécution d’un système avec différentes contraintes ou charge de travail. En d’autres mots, un bon comportement du temps est quand la vitesse d'exécution est rapide avec même des charges de travail élevées.

- Utilisation des ressources : La mesure de la quantité de ressources logicielles et matérielles utilisées par un logiciel pour effectuer ces tâches.

- Capacité : La capacité est la mesure d’un système de pouvoir effectuer des tâches avec une charge de donnée sans diminuer sa qualité. Le système doit être prêt à gérer de grosses charges imprévues et un nombre.

En se basant sur le but mentionné précédemment, ce rapport est séparé en quatre sections. Dans un premier temps, un plan d’assurance qualité sera élaboré pour définir les objectifs que l’application MOA doit atteindre pour être considérée comme étant de bonne qualité. Ensuite, ce plan nous permettra d’effectuer une stratégie de tests afin de valider les objectifs de performance définis dans le plan de qualité. Finalement, une séance de profiling avec JProfiler sera effectuée selon les 3 fonctionnalités principales de MOA (« Naive Bayes Theorem » , « Deep Learning », « Drift ») et une discussion à propos des résultats recueillis sera réalisée.

1.2. Critères de qualité couverts

Critére	Sous-critère	Objectif	Mesures de validation
Performance	Comportement du temps	Avoir un temps de réponse moyenne le plus petit possible (temps de réponse moyen [ISO/IEC FDIS 25023:2015-12 PTb-1-G])	$P = \frac{\sum_{i=1}^{n} (Y_i - X_i)}{n}$ Pour i = 1 à n X_i : Temps à la réception (par le système) de la fonctionnalité « i » Y_i : Temps à la transmission de la fonctionnalité « i » n : Nombre de temps de réponse mesurés
		Avoir un temps de délai moyen le plus petit possible (délai moyen [ISO/IEC FDIS 25023:2015-12 PTb-3-G])	$P = \frac{\sum_{i=1}^{n} (Y_i - X_i)}{n}$ Pour i = 1 à n X_i : Temps au commencement du travail de la fonctionnalité « i » Y_i : Temps à la fin du travail de la fonctionnalité « i » n : Nombre de délais mesurés
	Utilisations des Ressources	Minimiser l'utilisation moyenne du CPU (utilisation moyenne du CPU [ISO/IEC FDIS 25023:2015-12 PRu-1-G])	$P = \frac{\sum_{i=1}^{n} (X_i / Y_i)}{n}$ X_i : Utilisation réelle du CPU pour une durée d'observation de la fonctionnalité « i »

			Y_i : Durée d'observation de la fonctionnalité « i » n : Nombre d'observations
		Minimiser l'utilisation moyenne du mémoire volatile (utilisation moyenne de la mémoire volatile [ISO/IEC FDIS 25023:2015-12 PRu-2-G])	$P = \frac{\sum_{i=1}^{n} (X_i / Y_i)}{n}$ X_i : Espace de mémoire volatile utilisé par le système pour le traitement du fonctionnalité « i » Y_i : Espace de mémoire volatile maximum, pour le traitement du fonctionnalité « i » n : Nombre de traitements de fonctionnalités
	Capacité	Maximiser les transactions simultanées (capacité minimum de transactions simultanées [ISO/IEC FDIS 25023:2015-12 PCa-1-G])	P = X/Y X : Nombre de transactions simultanées complétées dans une durée d'observation Y : Durée d'observation
		Maximiser le nombre d'utilisateurs simultanés (capacité minimum d'utilisateurs simultanés [ISO/IEC FDIS 25023:2015-12	$P = \frac{\sum_{i=1}^{n} X_i}{n}$ Pour i = 1 à n X_i : Nombre d'utilisateurs qui peuvent accéder

		PCa-2-G])	simultanément au fonctionnalité « i » du système n : Nombre de fonctionnalités

1.3. Stratégies de validation

1.3.1. Profiling

Le profilage est un processus utilisé pour analyser la performance d'une application. Il permet de voir le mesure de la quantité de temps nécessaire pour chacune des fonctions, la mémoire utilisée, le nombre de threads, etc. L'objectif du profiling est d'identifier les parties du code qui prennent le plus de temps pour s'exécuter et les points de blocage qui ralentissent l'application, c'est-à-dire les goulots d'étranglement, dans le but d'optimiser un programme et la gestion des ressources. Par exemple, il peut avoir des fonctions qui utilisent beaucoup de ressources ou prend trop de temps pour commencer et pour terminer.

1.3.2. Monitoring

Le monitoring est un processus qui permet de surveiller une application en temps réel. Le monitoring est souvent utilisé pour la détection des anomalies. La différence entre le profiling et le monitoring est que le monitoring est utilisé quand l'application est exécutée en temps réel, tandis que le profiling est utilisé durant un test ou une simulation.

1.3.3. Tests de charge

L'objectif des tests de charge est de surcharger l'application avec beaucoup de données et d' analyser sa performance. La charge doit être plus que les conditions normales. Un exemple de test de charge est d'avoir plusieurs utilisateurs qui utilisent l'application en même temps. Ensuite, nous évaluons la performance de l'application en évaluant le temps de réponse, par exemple.

2. Stratégie de testing

2.1. Plan des tests

Critère	Sous-critère	Objectifs	Test ou diagnostique	Algorithme(s) testé(s)
Performance	Comportement du temps	L'indicateur du temps de réponse moyen P doit être égal ou inférieur à 1000 ms	Profiling ou Monitoring	Bayes Theorem, Deep Learning et Drift
		L'indicateur de délai moyen P doit être égal ou inférieur à 500 ms		
	Utilisations des Ressources	L'indicateur d'utilisation moyen du CPU P doit être égal ou inférieur à 0.40 (40%)	Profiling ou Monitoring	
		L'indicateur d'utilisation moyen de la mémoire volatile P doit être égal ou inférieur à 0.40 (40%)		
	Capacité	L'indicateur du nombre de transactions simultanées P doit être égal ou supérieur à 1000 requêtes par seconde	Tests de charge ou Profiling	
		L'indicateur du nombre d'utilisateurs simultanés P doit être égal ou supérieur à 500 utilisateurs		

2.2. Description des tests

Dans l'objectif de valider le critère de performance du système à l'étude, trois sous-critères seront évalués : le comportement du temps, l'utilisation des ressources et la capacité. Les sous-critères qui seront testés vont nous permettre d'identifier les potentiels problèmes de performance logicielle. Comme mentionné dans l'introduction, les tests décrits vont être appliqués aux trois fonctionnalités du module « classifiers », qui sont « Naive Bayes Theorem », « Deep Learning » et « Drift ».

2.2.1. Sous-critère comportement du temps

Pour tester le comportement du temps de ce logiciel, un profiling de l'application sera fait. L'outil de profilage JProfiler sera utilisé pour profiler MOA. Nous allons simuler une session d'utilisation des trois fonctionnalités mentionnées plus haut à l'aide des tests unitaires présents dans le logiciel. Deux indicateurs seront alors évalués pour tester le comportement du temps, qui sont le temps de réponse moyen et le délai moyen. Pour le temps de réponse moyen, nous nous attendons idéalement que l'indicateur soit égal ou inférieur à 1000 ms, tandis que pour le délai moyen, nous nous attendons idéalement à avoir un délai de 500 ms ou moins.

2.2.2. Sous-critère utilisation des ressources

Afin d'évaluer l'utilisation des ressources par l'application MOA, un profiling de cette dernière sera mis à profit à l'aide de l'outil JProfiler. De la même façon que pour le sous-critère précédent, nous allons évaluer deux indicateurs, qui sont l'utilisation moyenne du CPU et l'utilisation moyenne de la mémoire volatile (RAM). Pour l'utilisation moyenne du CPU et l'utilisation moyenne de la mémoire volatile, nous nous attendons idéalement que l'indicateur soit égal ou inférieur à 0.40 (40%) .

2.2.3. Sous-critère capacité

Pour évaluer le sous-critère capacité, la stratégie adoptée est les tests de charge. Ces tests vont permettre de simuler l'utilisation de l'application MOA à grande échelle et de tester la capacité de l'application à supporter une grande charge de travail. Ce type de tests est un bon moyen pour trouver les goulots d'étranglement du système. Pour ce faire, l'outil JMeter sera utilisé. Deux indicateurs seront alors évalués, qui sont le nombre de transactions simultanées dans le système et le nombre d'utilisateurs simultanés dans le système. Nous allons simuler à l'aide de JMeter 1000 requêtes/opérations par seconde et 500 utilisateurs par seconde qui utilisent le système simultanément. Pour le nombre de transactions simultanées, nous nous attendons idéalement que l'indicateur soit égal ou supérieur à 1000 requêtes/opérations par seconde, tandis que pour le nombre d'utilisateurs simultanés, nous nous attendons idéalement que le système puisse supporter au minimum 500 utilisateurs simultanément par seconde.

3. Préparer MOA et JProfiler pour le profiling

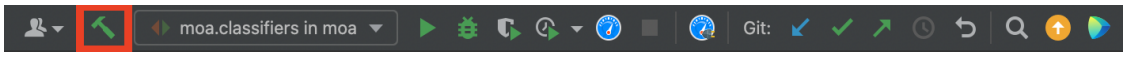
Afin de préparer l'application MOA et JProfiler pour le profiling, il est possible de configurer cette dernière de deux façons. Dans le cadre de ce rapport, nous avons opté pour la première méthode.

3.1. Première méthode de préparation

Pour la première méthode, la préparation requiert que des tests unitaires existent dans le projet. Ainsi, le profiling est fait lorsque les tests unitaires sont exécutés. Les étapes suivantes doivent être alors faites :

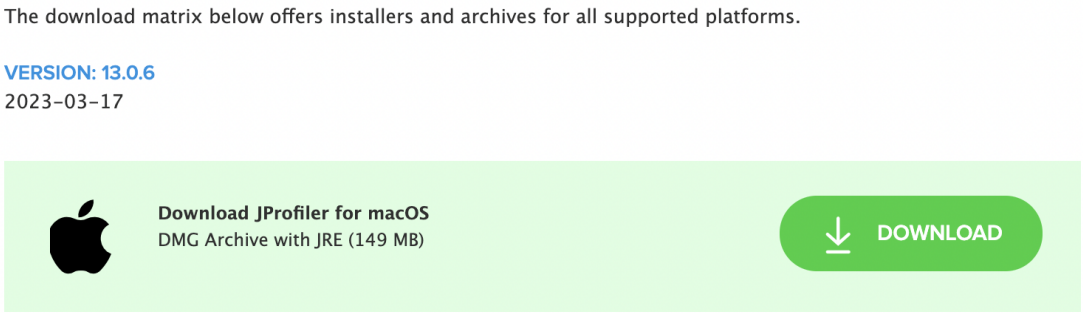
1. Pour préparer MOA, il faut être en mesure de pouvoir le build. Pour ce faire, il suffit d'ouvrir le projet sur IntelliJ IDEA et cliquer sur l'icône de marteau, comme montré dans la figure 1.

Figure 1 : Build du projet MOA



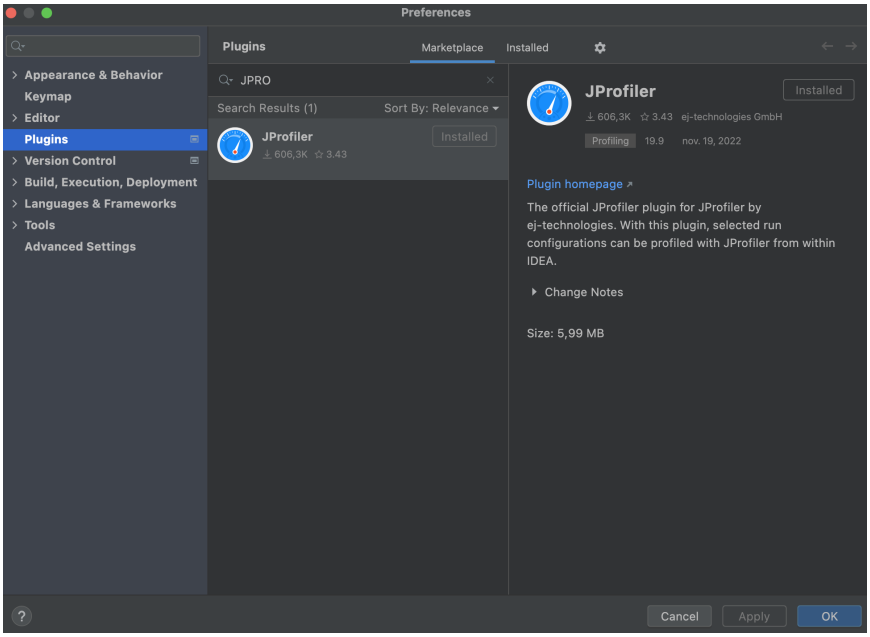
2. Ensuite, il faut installer l'application JProfiler et le plugin JProfiler sur IntelliJ IDEA.

Figure 2 : Installation de l'application JProfiler



Dans le cadre de cette étude, l'installation de la version d'évaluation de l'application JProfiler est suffisante.

Figure 3 : Installation du plugin JProfiler sur IntelliJ IDEA

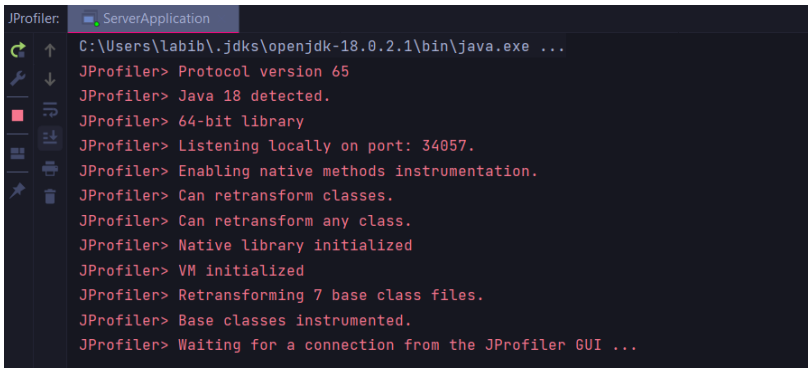


En ajoutant ce plugin, un bouton apparaîtra comme montré dans la figure 4 :

Figure 4 : Bouton JProfiler

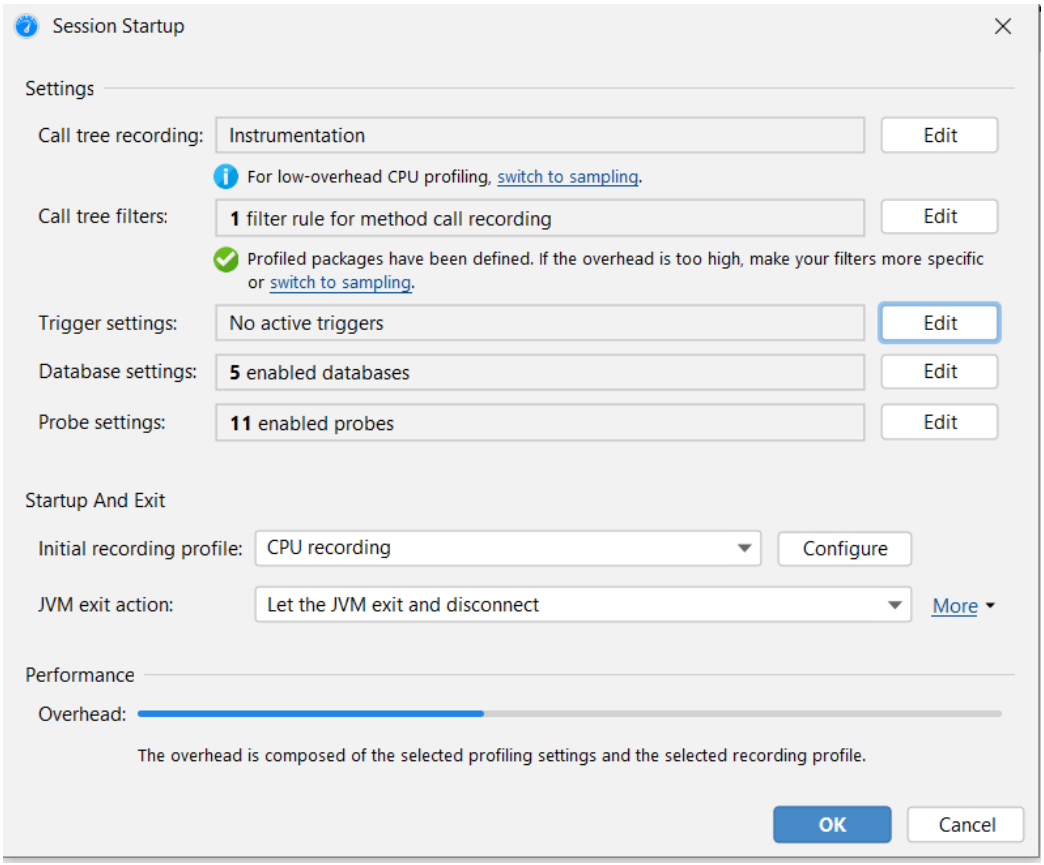
3. Pour démarrer le profiling, il suffit de cliquer sur le bouton montré à la figure 4. Le plugin essayera de se connecter avec JProfiler comme montré dans la figure 5. Le logiciel JProfiler s'ouvrira automatiquement.

Figure 5 : Tentative de connexion avec JProfiler



4. Pour pouvoir établir cette connexion, il faut le faire avec le logiciel JProfiler. Nous pouvons le faire simplement en cliquant sur le bouton « OK » dans la figure suivante :

Figure 6 : Session startup de JProfiler



En cliquant sur ce bouton, le profiling de l'application débute.

3.2. Deuxième méthode de préparation

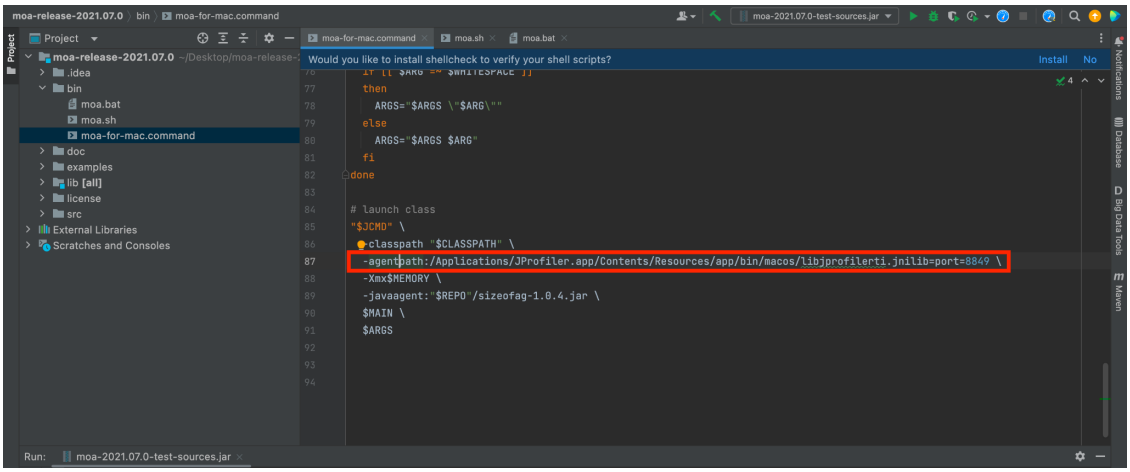
Cette méthode implique l'utilisation de l'interface de l'application MOA. Pour configurer le projet avec JProfiler pour permettre le profiling, les étapes suivantes sont à suivre :

1. Comme à l'étape 2 de la première méthode, il faut télécharger le logiciel JProfiler.

2. Il faut maintenant modifier le script d'exécution de l'application MOA afin de permettre l'exécution d'un agent de JProfiler lorsque l'application MOA est ouverte. Pour ce faire, il faut d'abord localiser l'agent, qui est le fichier « libjprofilerti ». Dans l'exemple, le chemin du fichier est «

/Applications/JProfiler.app/Contents/Resources/app/bin/macos/libjprofilerti.jnilib ». Enfin, dans le script d'exécution, il faut ajouter la ligne qui suit dans la section « launch class » : « -agentpath:/Applications/JProfiler.app/Contents/Resources/app/bin/macos/libjprofilerti.jnilib=port=8849 ». L'agent JProfiler va donc écouter sur le port 8849 les connexions à partir de l'interface graphique JProfiler.

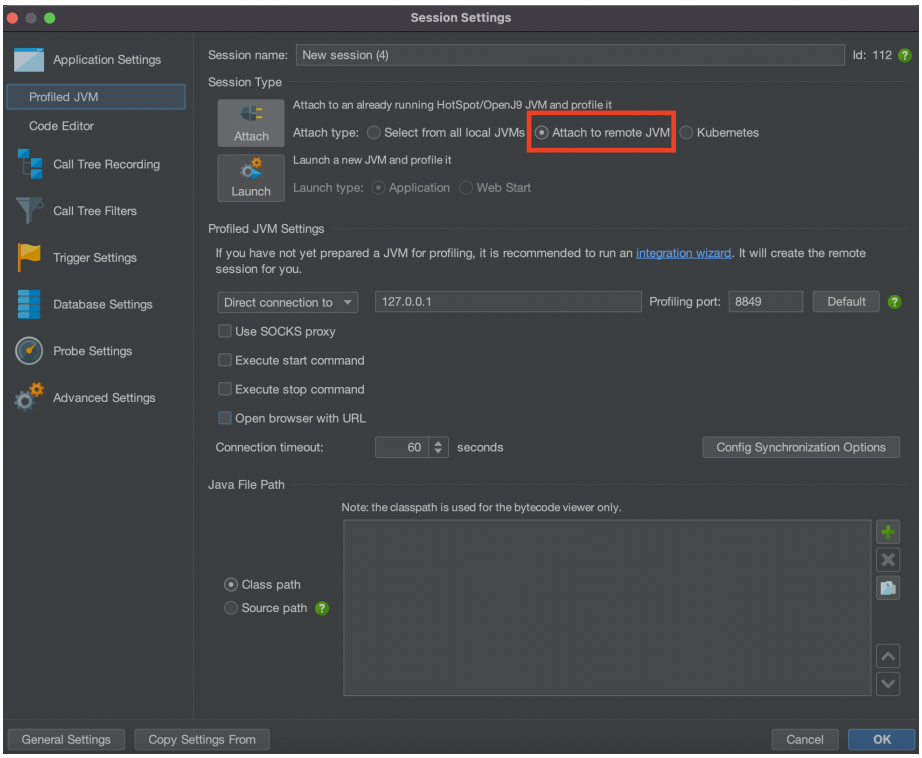
Figure 7 : Agent JProfiler



Dans l'exemple présenté à la figure 7, le fichier script choisi est celui qui permet de rouler l'application MOA sur macOS. Un autre système d'exploitation configurerait le fichier « moa.sh ».

3. À cette étape, il faut lancer l'application JProfiler et configurer une nouvelle session. Dans les paramètres de la session, il faut attacher un JVM. Il faut donc sélectionner l'option « Attach to remote JVM », comme montré dans la figure 8 :

Figure 8 : Nouvelle session de profiling



Nous pouvons garder les paramètres par défaut, tels qu'affichés dans la figure précédente. Ensuite, il faut cliquer sur « OK » et garder JProfiler ouvert.

4. À cette étape, il faut lancer le script d'exécution de l'application MOA, qui est dans le dossier /bin du projet. Une fois le script lancer, une fenêtre de configuration du profiling s'ouvrira (comme celle à la figure 6). Il suffit alors de cliquer sur le bouton « OK » et le profiling débute. Toutes les actions faites sur l'interface de l'application MOA seront alors profilées.

4. Rapport des tests de profiling

Comme décrit dans la description des tests, un profiling sera fait pour les sous-critères comportements du temps, l'utilisation des ressources et la capacité. Cependant, nous allons dans ce cas d'étude seulement présenter les résultats du profiling pour la consommation du CPU, la consommation de la mémoire volatile, le temps de réponse moyen ainsi que le nombre de transactions (travail) simultanées.

4.1. Sous-critère comportement du temps

4.1.1. Temps de réponse moyen

Figure 9 : Temps de réponse moyen pour Bayes

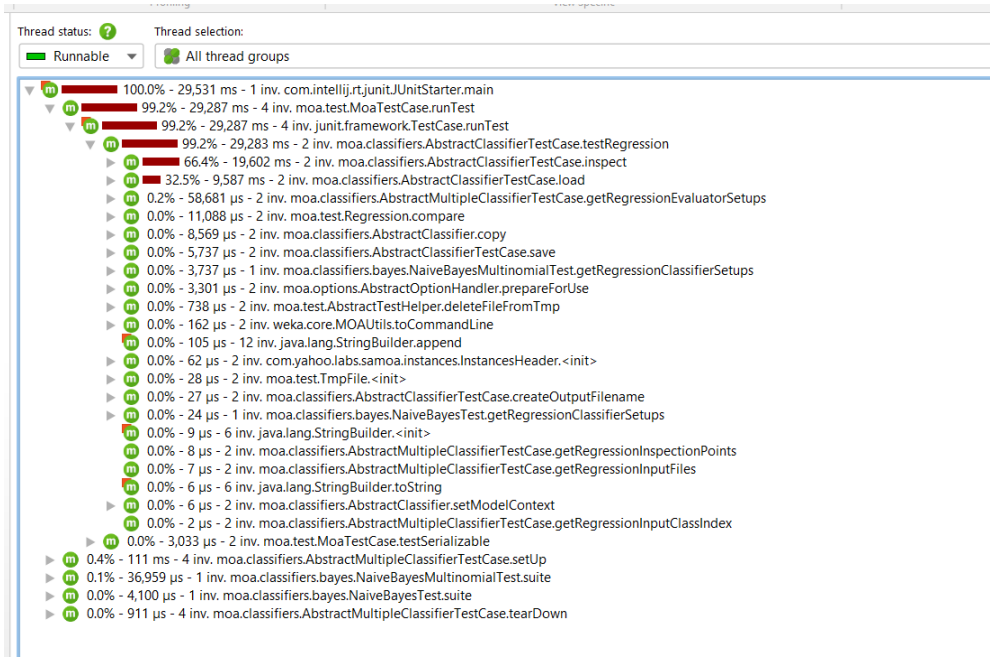


Figure 10 : Temps de réponse moyen pour Deep Learning

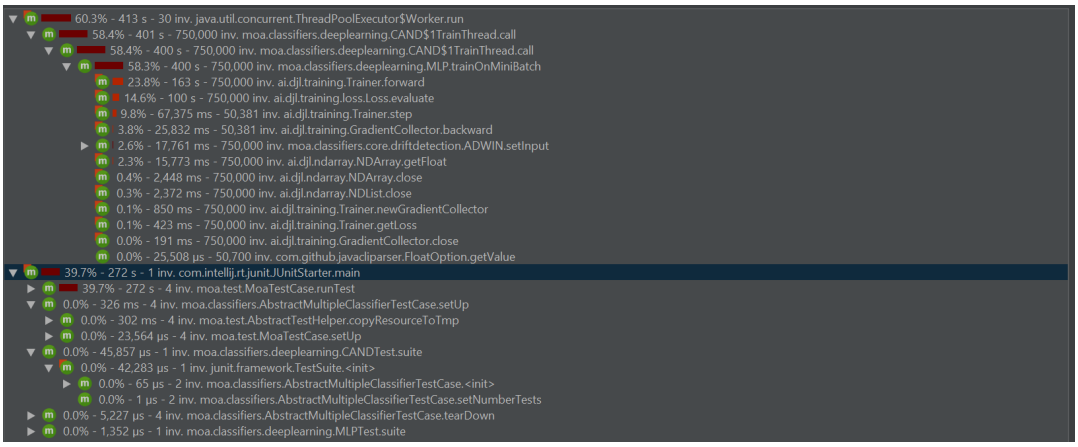
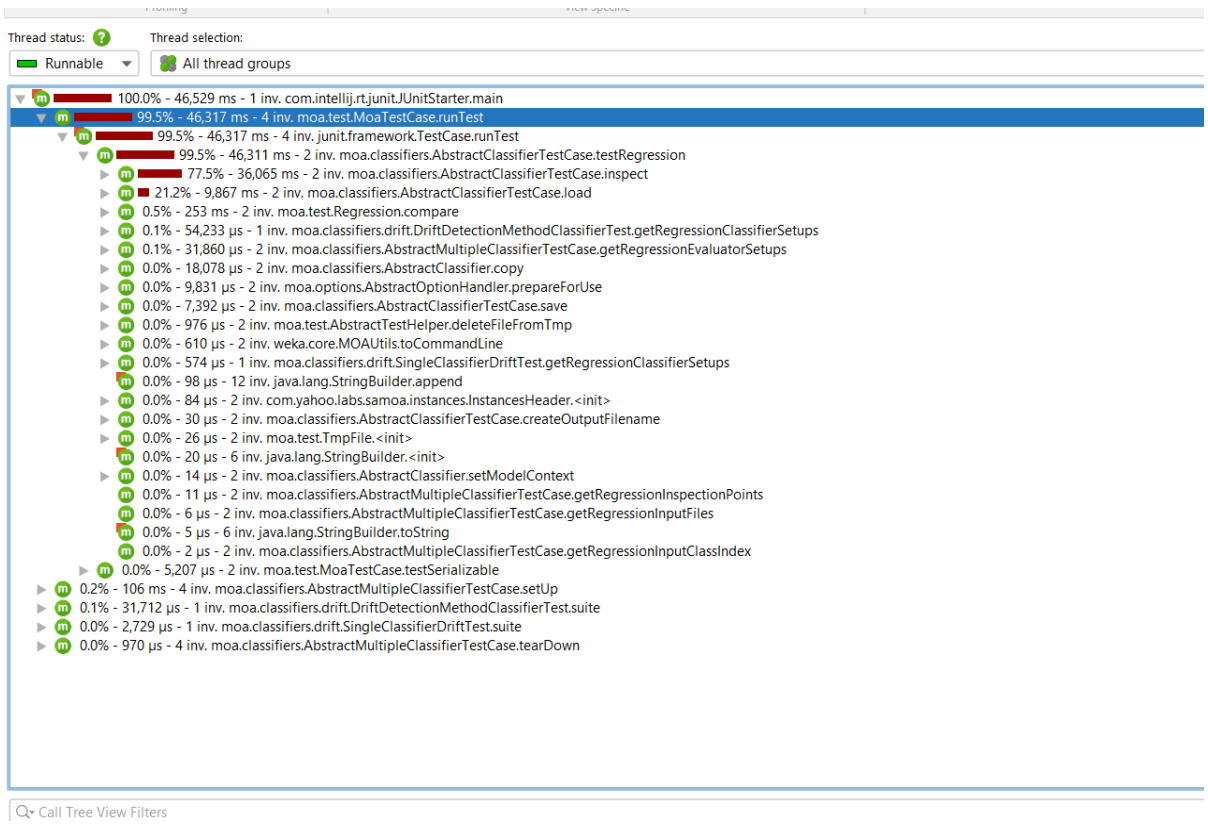


Figure 11 : Temps de réponse moyen pour Drift



Les figures 9, 10 et 11 nous permettent de calculer le temps moyen de réponse de ce système en prenant la moyenne des trois algorithmes. Le temps moyen de réponse que nous avons calculé est de 163 ms. Ce temps de réponse moyen est élevé à cause de l'algorithme « Deep Learning », qui est 413 secondes. Ceci est en bas du seuil de 1000 ms de temps de la réponse moyenne établie dans le tableau 2.1. De plus, il est aussi en dessous du seuil de 500 ms du délai moyen. Cela satisfait alors les objectifs reliés aux sous-critères de comportement du temps.

4.2. Sous-critère utilisation des ressources

4.2.1. Utilisation moyen du CPU

Figure 12 : Utilisation moyen du CPU pour Bayes

Figure 13 : Utilisation moyen du CPU pour Deep Learning

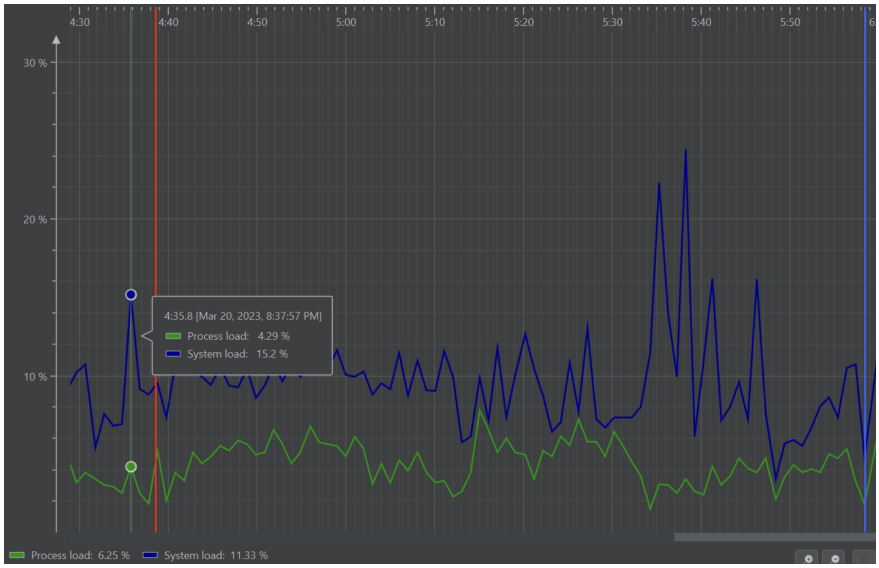
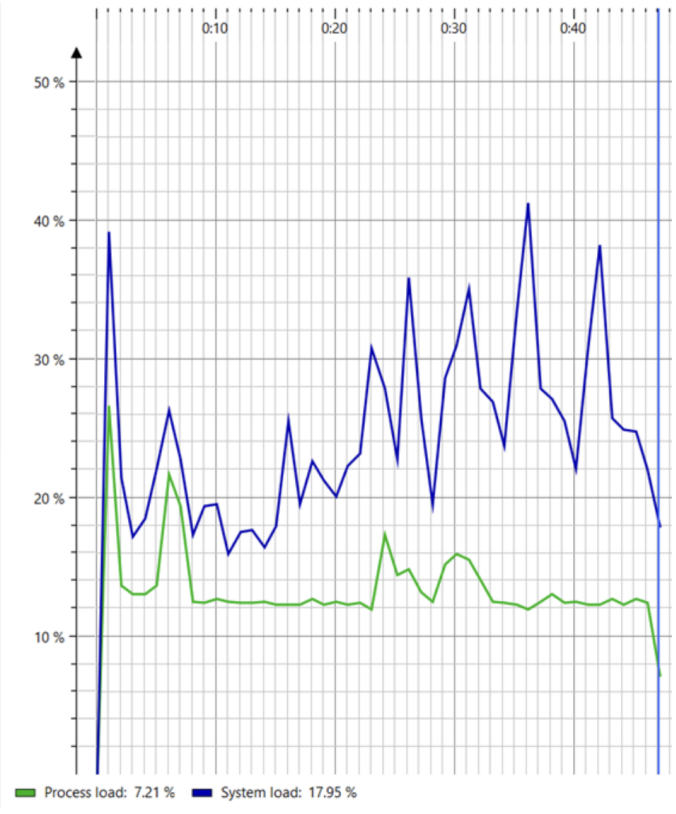


Figure 14 : Utilisation moyen du CPU pour Drift



Dans la figure 12, nous pouvons voir que l'utilisation moyenne de CPU augmente rapidement dès le début du profilage, plus spécifiquement la charge du système était de 49% au début et finit par être de 18.3%. La charge du processus augmente à 25% et se stabilise rapidement vers 9.06%.

Dans la figure 13, l'utilisation moyenne du CPU pour la charge de processus et la charge du système restent assez constantes du début jusqu'à la fin. La charge de système commence à 10% et finit à 10%. Le maximum que la charge de processus atteint est 7% et se stabilise à 6.25%.

Dans la figure 14, nous pouvons voir que l'utilisation moyenne du CPU augmente rapidement dès le début du profilage. En effet, sa charge de processus atteint un niveau de 38%. Sa charge change beaucoup au cours de son exécution et finalement se stabilise à 17.95%. Sa charge de système atteint un niveau de 26% et diminue jusqu'à tant qu'il stabilise à 7.21%.

Selon les trois algorithmes, il a seulement l'algorithme Bayes Theorem qui ne conforme pas à l'objectif de garder l'indicateur d'utilisation moyenne du CPU P sous 0.40 (40%). Les deux autres algorithmes restent confortablement en dessous de ce seuil. Nous pouvons conclure que Bayes Theorem doit être revisitée, car il utilise beaucoup trop de ressources.

4.2.2. Utilisation moyen de la mémoire volatile

Figure 15 : Utilisation moyen de la mémoire volatile pour Bayes

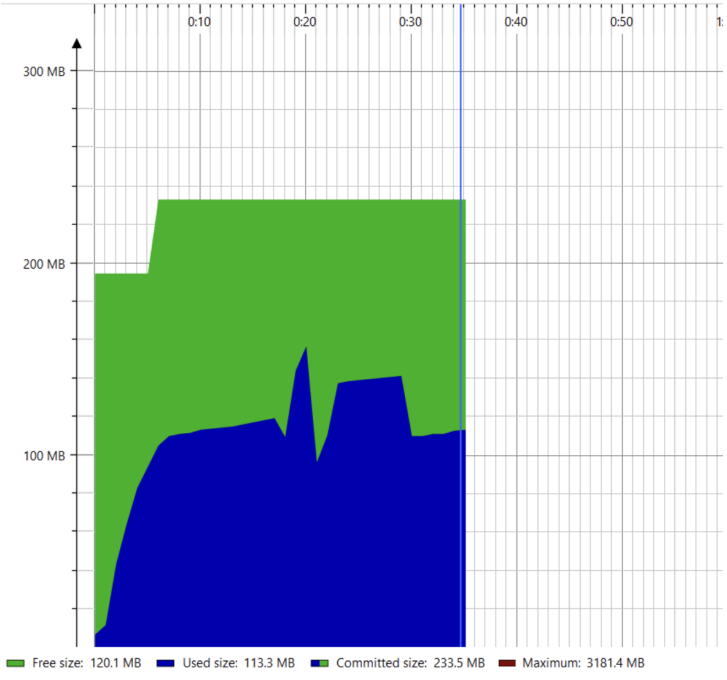


Figure 16 : Utilisation moyen de la mémoire volatile pour Deep Learning

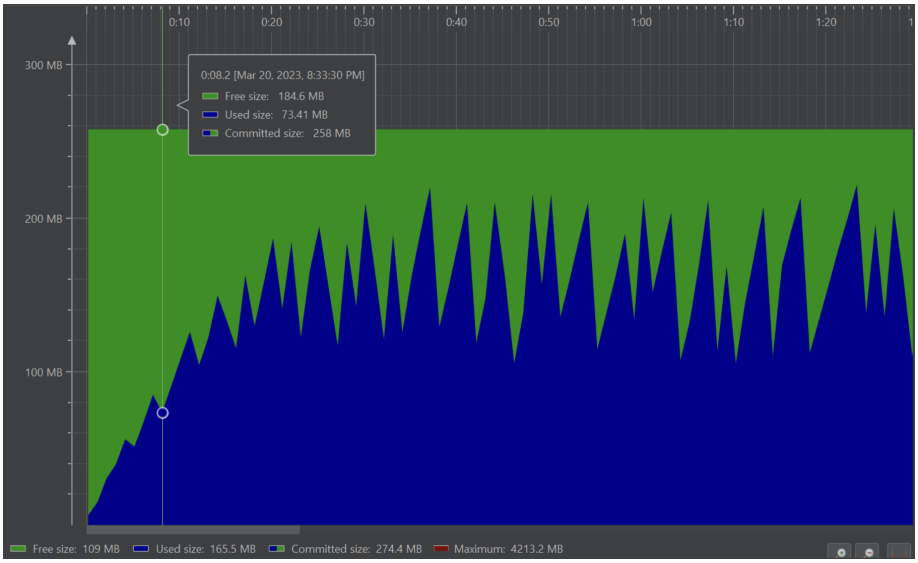
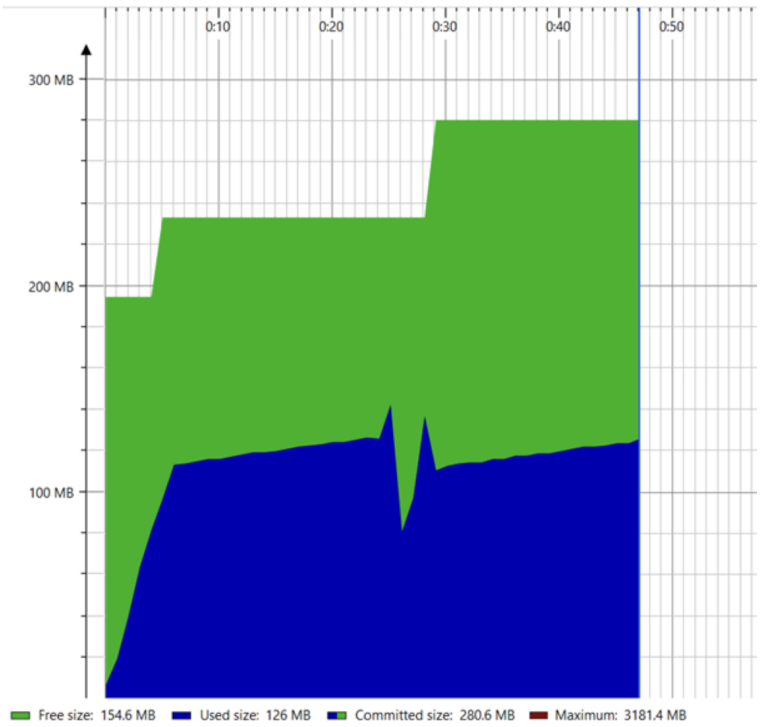


Figure 17 : Utilisation moyen de la mémoire volatile pour Drift



Dans la figure 15, le plus haut volume volatil utilisé est de 170 MB. Dans la figure 16, la plus haute mémoire volatile utilisée est 200 MB. Dans la figure 17, la plus haute mémoire volatile est 120 MB. Selon l'objectif que nous avons mis concernant la mémoire volatile, si on compare les données de la mémoire libre de celui qui est utilisé, nous pouvons voir qu'à plusieurs moments la mémoire volatile dépasse 40% de la mémoire totale, ce qui ne respecte pas l'objectif d'avoir une utilisation moyenne de la mémoire volatile égale ou inférieure à 40%. Il faut donc diminuer l'utilisation des ressources reliées à la mémoire volatile.

4.3. Sous-critère capacité

4.3.1. Nombre de transactions simultanées

Figure 18 : Thread pour Bayes

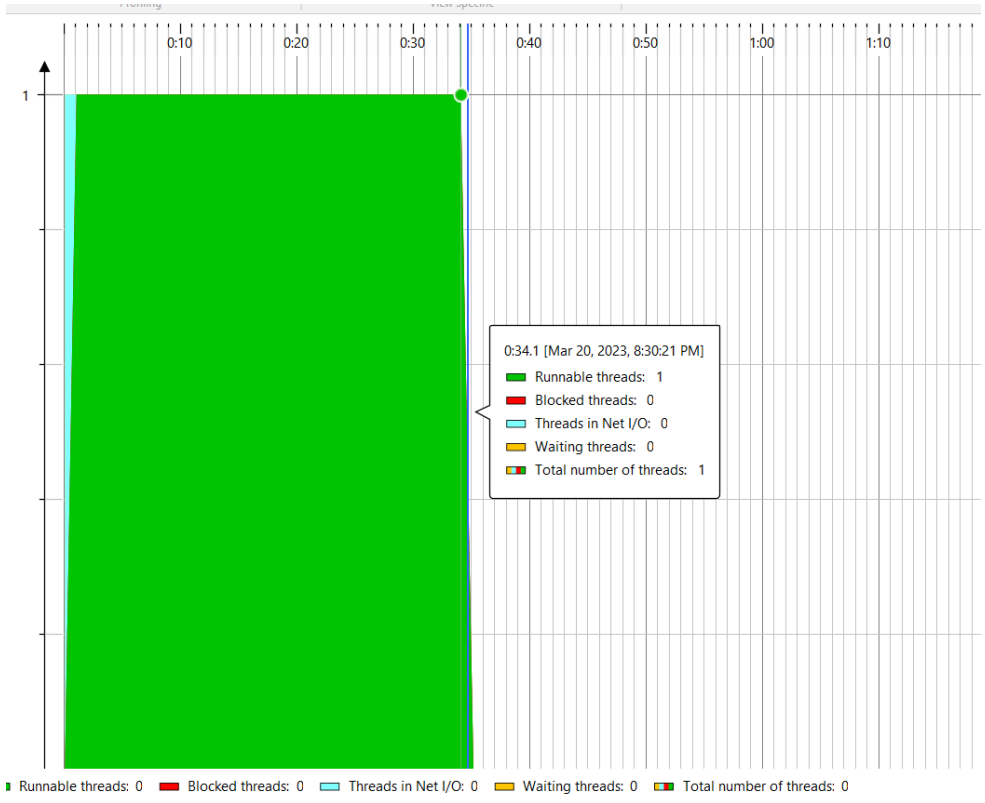


Figure 19 : Thread pour Deep Learning

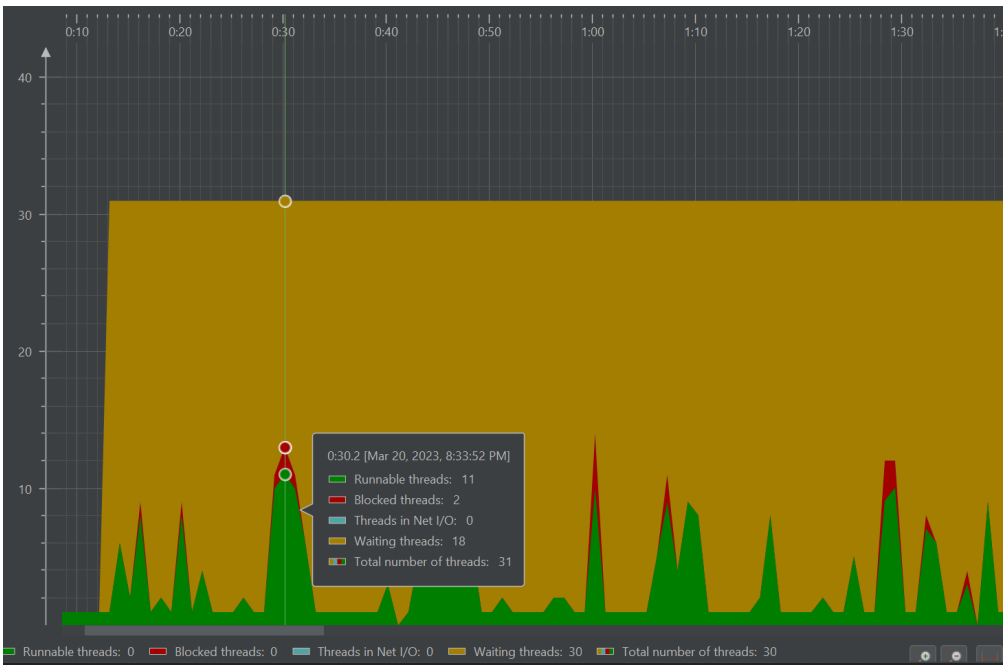
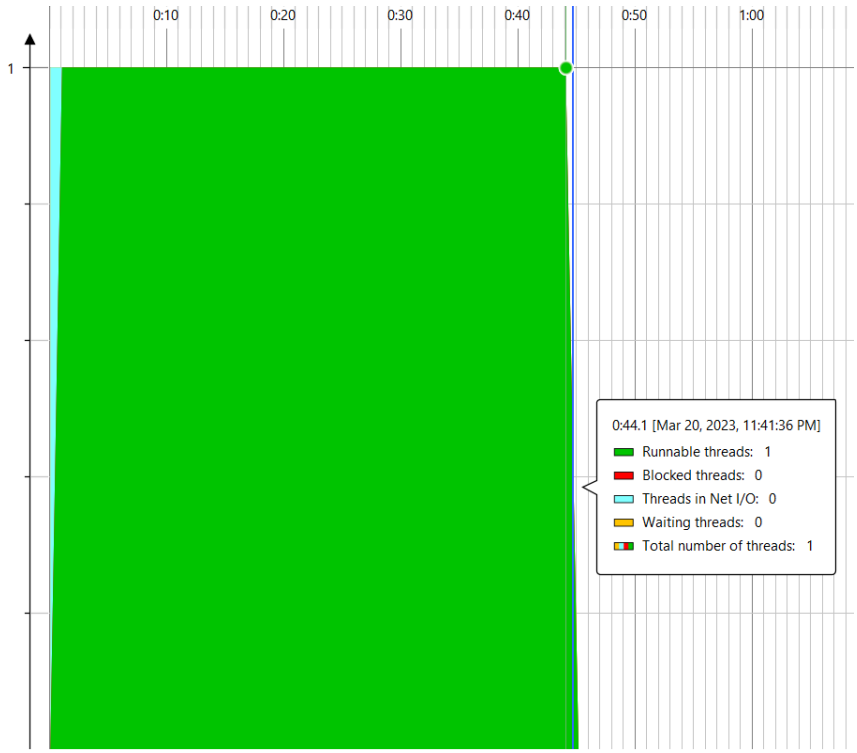


Figure 20 : Thread pour Drift



L'analyse des threads nous permet d'évaluer l'atteinte de l'objectif de maximiser le nombre de transactions simultanées. Dans la figure 19, nous voyons que le nombre total de threads est de 31 threads. Cela signifie qu'avec la fonctionnalité de « Deep Learning », il peut y avoir plusieurs fonctions qui roulent en même temps. Dans ce cas, nous avons exactement 11 fonctions qui roulent sur 11 threads différents. Avec les figure 18 et figure 20, nous remarquons que les algorithmes de « Bayes » et « Drift » permettent seulement de rouler un thread. Nous concluons alors que les fonctions de ces fonctionnalités n'ont pas pu être exécutées simultanément. Cependant, nous ne pouvons pas confirmer avec un profiling à l'aide de JProfiler que les résultats reflètent la réalité. Faire des tests de charge avec JMeter permettrait d'avoir une meilleure conclusion. Ainsi, à la lumière de notre analyse, nous pouvons voir que l'application peut bien rouler simultanément avec 11 threads. Nous extrapolons alors que l'application peut supporter un grand nombre de travails simultanés.

4.4. Conclusion

Objectifs/tests associés	**Résultats des cas de tests correspondants**	**Contre-mesure**
L'indicateur du temps de réponse moyen P doit être égal ou inférieur à 1000 ms	Succès	--
L'indicateur de délai moyen P doit être égal ou inférieur à 500 ms	Non testé	Performer un benchmark
L'indicateur d'utilisation moyen du CPU P doit être égal ou inférieur à 0.40 (40%)	Échec	Optimisation du code ou actions correctives au niveau matériel
L'indicateur d'utilisation moyen de la mémoire volatile P doit être égal ou inférieur à 0.40 (40%)	Échec	Optimisation du code ou actions correctives au niveau matériel
L'indicateur du nombre de transactions simultanées P doit être égal ou supérieur à 1000 requêtes par seconde	Succès	--
L'indicateur du nombre d'utilisateurs simultanés P doit être égal ou supérieur à 500 utilisateurs	Non testé	Tests de charge avec JMeter

5. Références

1. ISO 25000 Portal. (s.d.). ISO/IEC 25010. https://iso25000.com/index.php/en/iso-25000-standards/iso-25010.
2. Hydro-Québec. (s.d.). Normalisation des exigences. https://docs.google.com/spreadsheets/d/0B4cNbsMq3e3bWklBNEtPTkxMMUE/edit?resourcekey=0-drvZT7ONe9z7JIvY6EaV4Q#gid=998574238.
3. EJ technologies. (s.d.). Profiling A JVM. https://www.ej-technologies.com/resources/jprofiler/help/doc/main/profiling.html.
4. SmartBear. (s.d.). Fundamentals of Performance Profiling. https://smartbear.com/learn/code-profiling/fundamentals-of-performance-profiling/#:~:text=Performance%20profilers%20are%20software%20development,poorly%20performing%20sections%20of%20code.