École Polytechnique de Montréal

Département de génie informatique et logiciel

LOG8371

Ingénierie de la qualité en logiciel

Travail pratique 2

Soumis par

Simon Barrette : 1787758
Vincent Dandenault : 1792866
Charles-Olivier Favreau : 1789479

Section 01 – Équipe Québec

21 mars 2019

Q1) Mettez à jour le plan de qualité pour ajouter les critères et sous-critères qui portent sur l'efficacité et la performance de Weka. Mettez à jour les objectifs, les tests et tous les détails nécessaires. Concentrez-vous sur les 5+1 fonctionnalités que vous avez spécifiées au TP1. Soumettez la nouvelle version du plan dans l'entrepôt soumis. (20 points)

Description des critères et des sous-critères de qualités ciblées:

En se basant sur le modèle de la qualité pour un produit logiciel ISO25010, le premier critère, soit la **fonctionnalité**, se définit comme étant le degré auquel un produit ou un système fournit des fonctions qui rencontre les besoins définis lorsqu'utilisés dans des conditions spécifiques. Ainsi, en ce qui concerne le critère de fonctionnalité, le sous-critère le plus important est clairement la *functionnal correctness*, qui se traduit par **exactitude fonctionnelle**. Celui-ci se définit simplement comme étant le degré auquel le produit ou le système fournit des résultats exacts, avec le bon degré de précision.

En ce qui concerne le critère de **fiabilité**, ce dernier se définit comme étant le degré auquel un système, un produit ou une composante effectue des fonctions spécifiques sous des condititions spécifiques, et ce pour une période de temps donnée. Parmi les sous-critères liés à la fiabilité, il est intéressant de se préoccuper de la **tolérance aux fautes** (*fault tolerance)*, qui se définit comme étant le degré auquel un système, un produit ou une composante opère tel que voulu, et ce même en présence de fautes logicielles ou matérielles. Dans le cadre du critère de fiabilité, il est aussi intéressant de regarder la **récupérabilité** (*recovrability)*, soit la capacité d'un système, un produit ou une composante, à récupérer les données directement affectées et rétablir l'état du système dans le cadre d'une interruption ou un bris.

Finalement, le critère de **maintenabilité** se définit comme étant le degré d'efficacité et de rendement avec lequel un produit ou un système peut être modifié intentionnellement. Parmi les sous-critères existants, il y a d'abord la **modularité**, soit le degré auquel un système ou un programme logiciel est composé de parties distinctes de façon à ce que le changement d'une partie n'ait pas d'impact sur les autres parties de ce même programme. De plus, il y a la **réutilisabilité**, qui, comme son nom l'indique, est le degré auquel une partie d'un système peut être utilisé dans plus d'un système, ou dans le but de bâtir d'autres parties. [3]

Finalement, le nouveau critère à considérer pour ce deuxième travail pratique est celui de **l'efficacité de performance** (performance efficiency). Selon la norme ISO25010, ce critère se définit brièvement comme étant la quantité relative de ressource utilisée sous certaines conditions établies. Ces ressources peuvent prendre la forme d'autre produit logiciel, des configurations de logiciel et de matériel, en plus du matériel lui-même, tel que du stockage. De ce critère découle trois sous-critères important, le premier étant le **comportement par rapport au temps** (*time behavior*). Ce dernier se définit comme étant le degré de conformité du système

aux exigences, en rapport au temps de réponse et de traitement ainsi qu'aux taux de traitement. S'en suit l'**utilisation des ressources**, soit le degré de conformité aux exigences d'un système par rapport à la quantité et le type de ressource utilisées par ce même produit ou système lorsqu'il exécute ses fonctions. Finalement, le dernier critère est la **capacité**, soit le degré de conformité des limites maximales de certains paramètres d'un système aux exigences. [norme iso]

Description des objectifs de qualité:

Le tableau suivant présente les objectifs de qualité pour chacun des sous-critères énoncés ci-dessus.

Tableau 1.1 - Objectifs de qualité selon les sous-critères

Critère	Sous-critère	Objectifs
Fonctionnalité	Exactitude fonctionnelle	Produit les résultats attendus dans 95% des cas, avec une exactitude continue de plus de 90% sur ces résultats.
Fiabilité	Tolérance aux fautes	Résistance aux fautes logicielles et matériels dans plus de 75% des cas de figure possibles.
Fiabilité	Récupérabilité	Temps de récupération de moins de 10 minutes dans 90% des cas de défaillances.
Maintenabilité	Modularité	Permet l'ajout d'un module ou la modification d'un module existant en n'ayant aucune répercussion sur le reste du programme, et ce dans 85% des cas.
Maintenabilité	Réutilisabilité	Permet l'utilisation à mainte reprise de plus de 40% du code et de certains modules clés.
Efficacité de performance	Comportement par rapport au temps	Veuillez vous référer au tableau 1.2 explicitant les objectifs en

		fonction des cas de charge
Efficacité de performance	Utilisation des ressources	Veuillez vous référer au tableau 1.3 explicitant les objectifs en fonction des cas de charge
Efficacité de performance	Capacité	Voir note de bas de page[1]

Tableau 1.2 - Objectifs de qualité en fonction du comportement par rapport au temps, pour 4 charges différentes

Charge[2]	**Temps de réponse**	**Latence moyenne**	**Taux de traitement**	**Requêtes par seconde**
Réduite	50 ms	10 ms	50 Ko/sec	50
Moyenne	100 ms	15 ms	100 Ko/sec	100
Augmentée	150 ms	25 ms	200 Ko/sec	150

Tableau 1.3 - Objectifs de qualité en fonction de l'utilisation des ressources, pour 4 charges différentes

Charge	**Utilisation de la RAM**	**Threads**	**Utilisation du CPU**
Réduite	20%	1	10%
Moyenne	35%	2	25%
Augmentée	50%	3	50%

Évidemment, la théorie sur la détermination des objectifs dans le cadre d'une approche SPE (*Software Performance Engineering*) dans un SDLC (*Software Development Life Cycle*) nous apprend qu'il est en fait nécessaire de définir les requis non-fonctionnels en début de projet, puis d'en faire une revue dans le cadre des tests de performance. De ce fait, puisque la documentation existante par rapport à Weka ne fait pas état des requis non-fonctionnels lié à l'efficacité de performance, les objectifs définis dans ces tableaux pour les trois sous-critères liés à l'efficacité de performance sont purement basé sur des évaluations générales des requis

[1] Les objectifs de capacité correspondent aux objectifs des métriques du comportement par rapport au temps ainsi qu'à l'utilisation des ressource, pour chaque niveau de charge définis dans les tableaux 1.2 et 1.3.

[2] Les définitions des métriques explicités dans les tableaux 1.2 à 1.4 sont disponibles en annexe 1.

de Weka, ainsi que sur l'observation de sa performance dans un cas d'utilisation jugé normal. [ppt SPE, cours 6]

Quoique jugé optionnel, puisque la documentation du cours diverge en ce qui concerne la définition même des objectifs de qualité, il est aussi important de définir cette dernière selon la définition globale de Laporte et April. Ainsi, on peut émettre les objectifs de qualité du produit en général, soit:

- Assurer la qualité du produit qui rencontre les requis du client
- Assurer la conformité aux requis de gestion en vue des coûts et de l'échéancier
- S'assurer que la gestion de la qualité peut permettre l'atteinte des résultats voulus
- Assurer avec un niveau acceptable de confiance que le produit est conforme aux requis techniques [5][6]

Plan de vérification et de validation des objectifs:

Afin de valider et de vérifier les objectifs définis ci-haut, une série de mesure, de tests et de techniques permettant la revue du code seront utilisés. Ces dernières sont explicitées ci-dessous, de même que les objectifs que chacune de ces mesures tente de couvrir.

- *Tests unitaires et d'intégration*

Objectif de test:	S'assurer que les fonctions aient le comportement attendu dans tous les cas où elles sont employées
Technique:	Utiliser les tests déjà présent dans Weka et propre aux algorithmes à tester.
Critère de complétion:	Tous les tests sont complétés sans erreur.
Critères vérifiés:	Ces tests permettent de vérifier le critère de Fonctionnalité, et plus particulièrement celui d'exactitude fonctionnelle. En considérant la complexité cyclomatique, il est possible de tester sommairement la maintenabilité. Ceci considère une approche en boîte blanche.

- *Tests de charges (load testing)*

Objectif de test:	S'assurer que le système ou produit logiciel puisse performer de manière efficace et dans un temps raisonnable en fonction des tâches à exécuter et de son allocation de ressource.
Technique:	Tester des charges de 4 tailles différentes et sous plusieurs conditions, en plus de faire varier la configuration logicielle et matérielle dans un environnement virtuel.
Critère de complétion:	Les métriques observées atteignent les objectifs d'efficacité de performance établie pour le logiciel.
Critères vérifiés:	Ces tests permettent de vérifier le critère de Fiabilité, et plus particulièrement celui de tolérance aux fautes et de récupérabilité.

- *Revues par les pairs*

Objectif de test:	Détecter des erreurs et trouver des solutions.
Technique:	Inspections et walk-through, et surtout utilisation des mécanismes de *pull request.*
Critère de complétion:	L'approbation de la *pull request* par les examinateurs assigné à la tâche.
Critères vérifiés:	Tous, mais dans ce cas-ci, il serait suggérer de se focaliser sur les critères n'ayant pas encore été dûment testés, soit la maintenabilité et ses sous-critères: la modularité et la réutisabilité.

- *Audits du code*

Objectif de test:	S'assurer que les critères à tester sont remplis.
Technique:	Inspection (pouvant être faite à l'interne) par l'initiateur, le chef auditeur, les auditeurs, l'enregistreur et l'organisation auditée.
Critère de complétion:	L'approbation du rapport final par l'initiateur.
Critères vérifiés:	Tous, mais dans ce cas-ci, il serait suggérer de se focaliser sur les critères n'ayant pas encore été dûment testés, soit la maintenabilité et ses sous-critères: la modularité et la réutisabilité. Ces audits devraient être faits avant les nouvelles sorties de version du système.

En ce qui concerne les stratégies d'intégration continue et de développement continu, les outils suivants seront utilisés:

Type de test	Outil
Tests unitaires et d'intégration	TravisCI et GitHub, Maven, JUnit
Tests de charges	Docker, JProfiler, JMeter
Revues par les pairs	Pull Request
Audits de code	Spécialistes et développeur, membres nécessaires à l'audit

En ce qui concerne le processus permettant l'application du plan de test, ce dernier va comme suit: Tout d'abord, quelqu'un voulant faire des ajouts au code doit créer une nouvelle branche. Grâce aux outils d'intégration continue définis ci-haut, les tests unitaires et d'intégration vont automatiquement être déclenché à chaque *commit* fait sur la branche dans le dépôt Git (GitHub). Par la suite, lorsqu'une fonctionnalité est terminée, la personne responsable du développement doit créer une *pull request* et assigner des examinateurs afin de faire la revue par les pairs. C'est au moment de la création de cette *pull request* que les tests de charge peuvent être exécutés. Selon les commentaires des examinateurs, les modifications demandées sont faites, puis cette *pull request* est acceptée par un administrateur et la fonctionnalité est fusionnée avec la branche principale, en considérant que les tests de charges, unitaire et d'intégration sont aussi valides. Il est ensuite suggéré de faire des audits de code à chaque sortie de nouvelle version. Cette méthode permet de réduire la taille des audits au fil du temps.

Q2) Déployez une version REST de Weka en utilisant des containeurs Docker. Clonez cette version de Weka dans votre entrepôt et créez des images Docker qui sont nécessaires pour déployer le logiciel. Soumettez un petit manuel avec les commandes pour déployer Weka en Docker. Dans le vidéo de la question Q6, vous devez démontrer votre déploiement en Docker. (5 points)

Weka REST : https://github.com/jguwekarest/jguwekarest
Manuel : Veuillez consulter le fichier PDF appelé Manuel_Q2, dans le dépôt de remise.

Q3) Profilez votre version de Weka en utilisant un outil comme JProfiler. Rapportez les résultats sur la consommation des ressources (CPU, RAM etc.). Rapportez ces résultats par rapport aux différents cas de charge (nombre de requêtes et de données). Préparez au moins 4 scénarios (un de charge réduite, un de charge moyenne, un de charge augmentée, et un de charge augmentée exceptionnelle). Soumettez un petit manuel d'usage du JProfiler pour votre projet en incluant les étapes pour la configuration et l'exécution de l'outil, les données d'entrée (aussi les scenarios spécifiés) et les résultats. (20 points)

D'abord, nous allons commencer par élaborer nos 4 scénarios de cas de charge. À vrai dire, nous avons faites des combinaisons de requêtes pour chaque scénario qui recherche à atteindre les buts de capacités suivants :

Tableau 3.1 - Scénarios pour les cas de charge

Scénarios	GET	POST	Number of Threads	Loop Count	Ramp-Up Period
1 - Réduite	100	100	200	10	100
2 - Moyenne	500	500	1'000	100	100
3 - Augmentée	1'000	1'000	2'000	100	100
4 - Augmentée exceptionnelle	2'000	2'000	4'000	100	100

Tableau 3.2 - Résultats par rapport aux différents cas de charge

Charge	Nombre de requêtes total	CPU	RAM	Throughput
Réduite	200	2%	172.4 MB	13`000

Moyenne	1’000	71%	178.2 MB	59`000
Augmentée	2’000	92%	198.8 MB	107`000
Augmentée exceptionnelle	4’000	93%	161.2 MB	169’000

Manuel d’usage :
Jprofiler est un outil efficace pour évaluer la performance d’une application Java. En effet, JProfiler peut détecter toutes les applications qui s’exécutent sur la machine locale. Cependant, dans notre cas, notre application Weka Rest s’exécute dans un conteneur sur *Docker*. Ainsi, nous devons faire quelques étapes pour avoir *JProfiler* lors de l’exécution.

Premièrement, nous devons spécifier où nous installons *JProfiler* dans notre machine LINUX/UNIX tel que :

```
RUN wget
http://download-keycdn.ej-technologies.com/jprofiler/jprofiler_linux_1
0_0_4.tar.gz -P /tmp/ &&\
      tar -xzf /tmp/jprofiler_linux_10_0_4.tar.gz -C /usr/local && \
      rm /tmp/jprofiler_linux_10_0_4.tar.gz

      EXPOSE 8849
```

Deuxièmement, nous devons dire à notre conteneur d’exécuter le programme de *JProfiler* lors de l’exécution de l’application WEKA. Dans le fichier *docker-compose.yml*, ajouter la commande tel que :

```
CATALINA_OPTS:
'-agentpath:/usr/local/jprofiler10.0.4/bin/linux-x64/libjprofilerti.so
=port=8849'
```

Finalement, nous devons exposer notre port à notre hôte d’un conteneur puisque *JProfiler* s’exécute sur l’hôte et s’attache au conteneur à l’aide du *localhost* tel que :

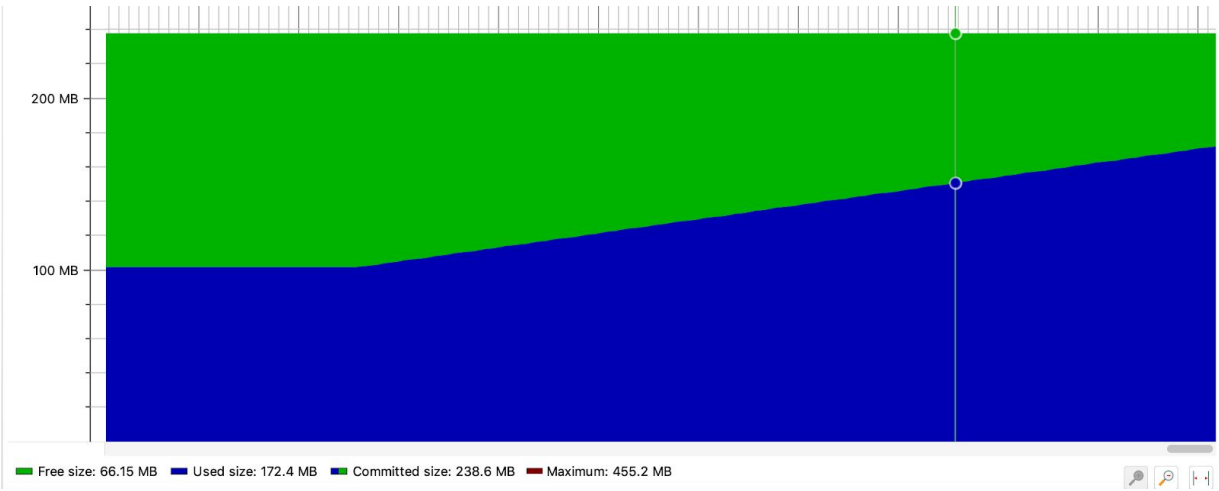
```
ports: '8849:8849'
```

Ressources utilisées:
JProfiler : https://www.ej-technologies.com/products/jprofiler/overview.html
Tutoriel:
https://clairekeum.wordpress.com/2017/10/18/monitoring-a-containerized-app-with-jprofiler/

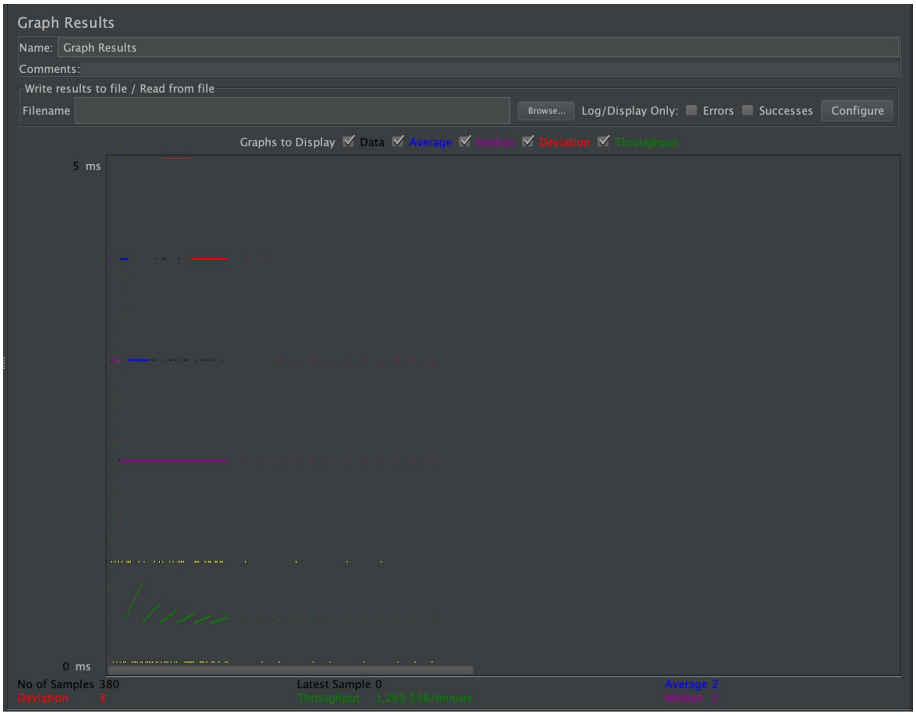
Q4) Préparez et exécutez des load tests pour votre version de Weka. Utilisez les scenarios créés pour la question 3. Utilisez l'outil JMeter pour exécuter les load tests. Rapportez sur les scenarios ou les objectifs ne sont pas possible à atteindre. Soumettez un petit manuel d'usage du JMeter pour votre projet en incluant les étapes pour la configuration et l'exécution de l'outil, les données d'entrée (aussi les scénarios spécifiés) et les résultats (20 points)

Pour les load test, nous avons eu recours à l'application JMeter. En effet, comme mentionné auparavant, nous avons défini 4 groupes de tests pour les loads tests. Chacun de ces load tests comptait un différent nombre de requêtes GET et de requêtes POST. Le but était d'évaluer la performance ainsi que l'utilisation de ressources du conteneur Docker. Les résultats des tests sont ci-dessous.

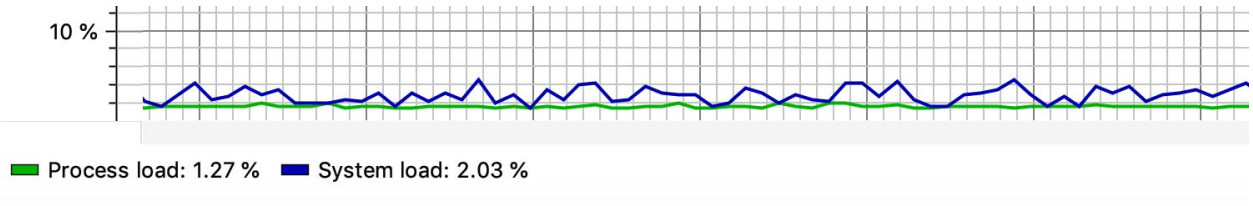
Charge réduite :



Mémoire RAM utilisée

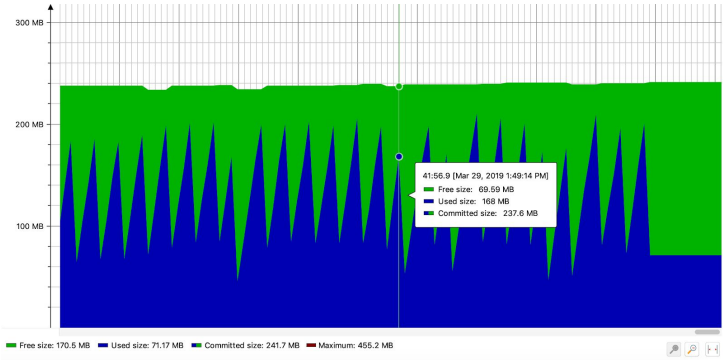


Graph de ressources

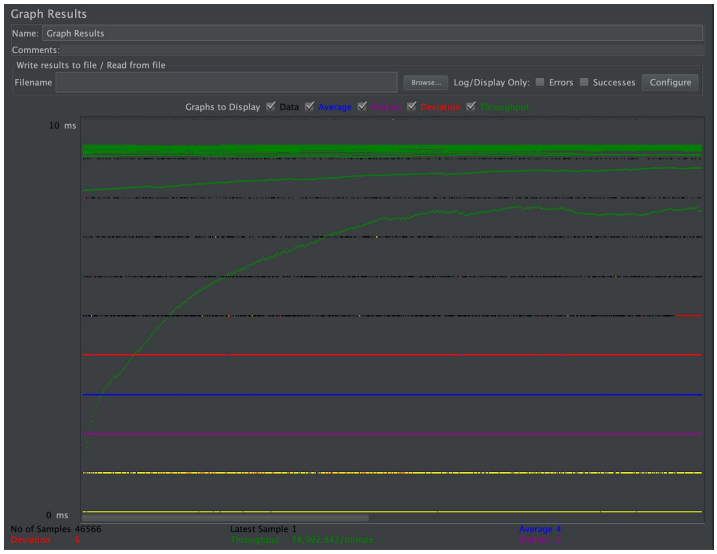


Utilisation de CPU

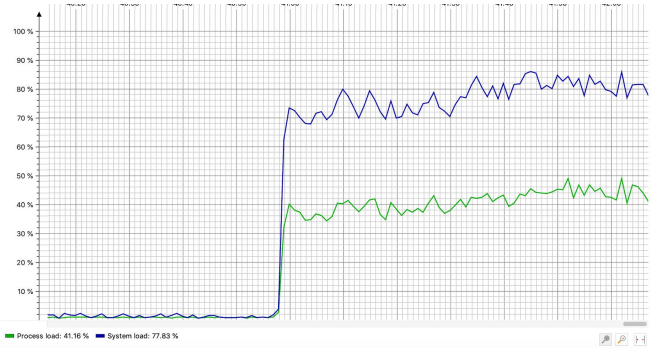
<u>Charge moyenne</u> :



Mémoire RAM utilisée

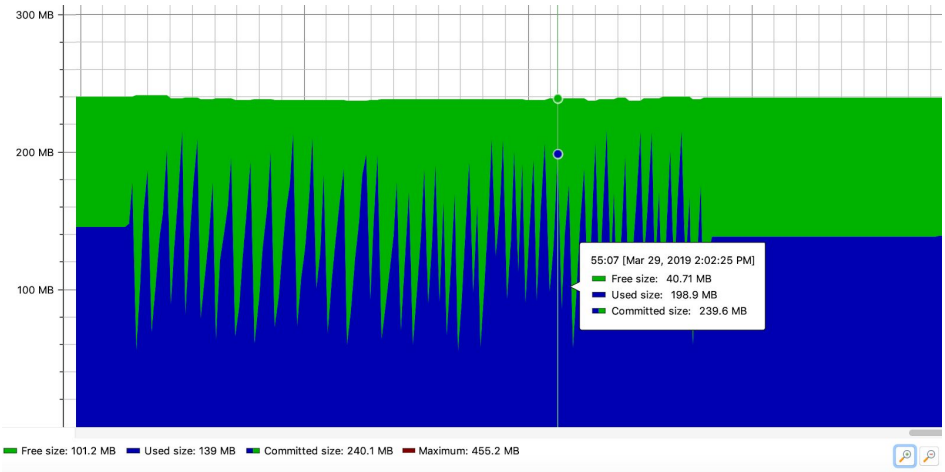


Graph de ressources

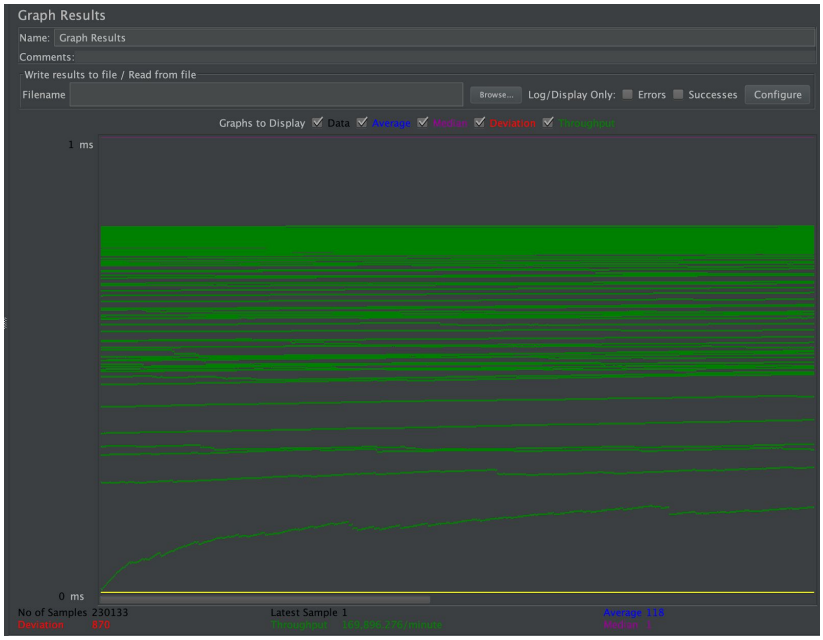


Utilisation de CPU

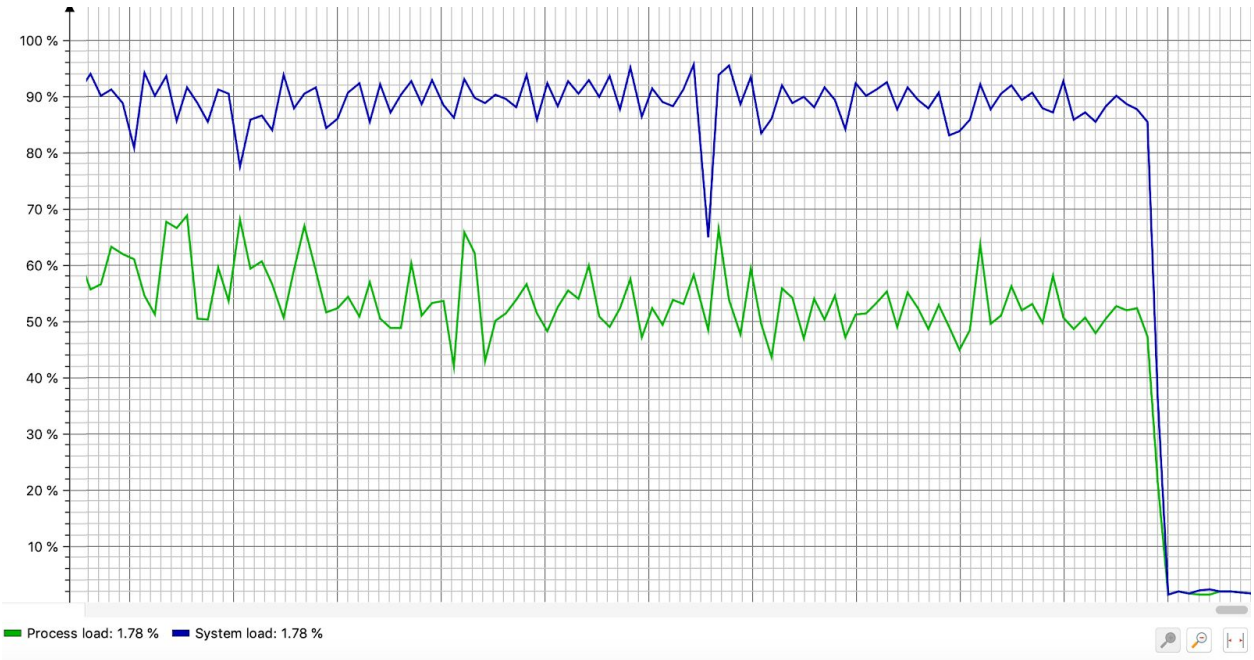
<u>Charge Elevée</u> :



Mémoire RAM utilisée

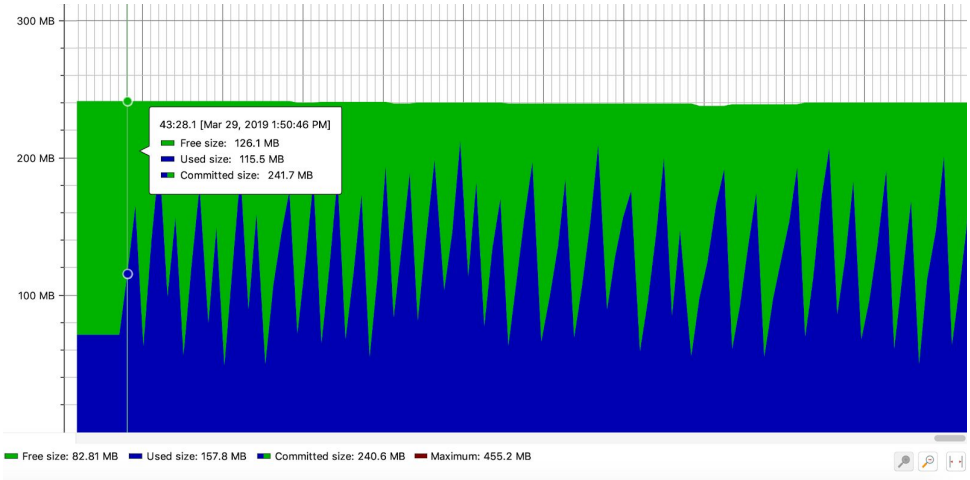


Graph de ressources

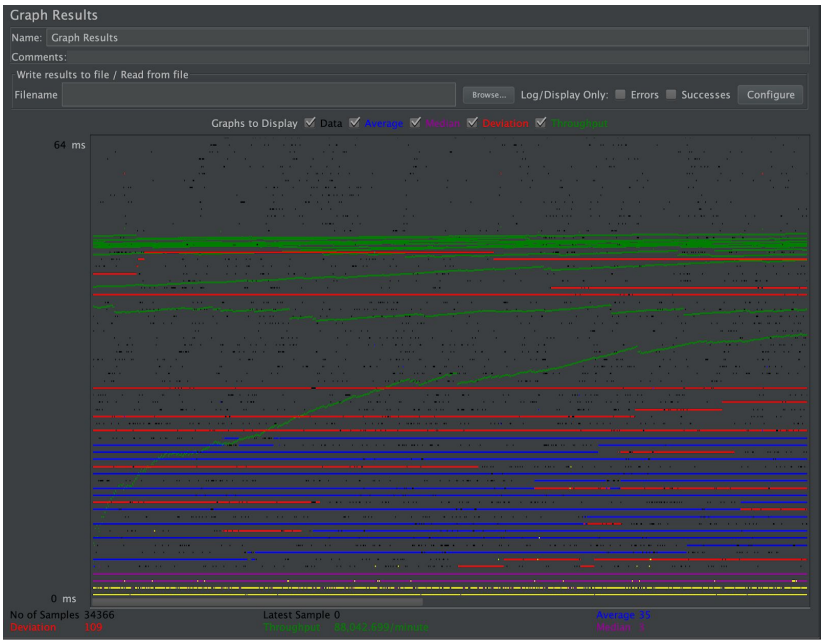


Utilisation de CPU

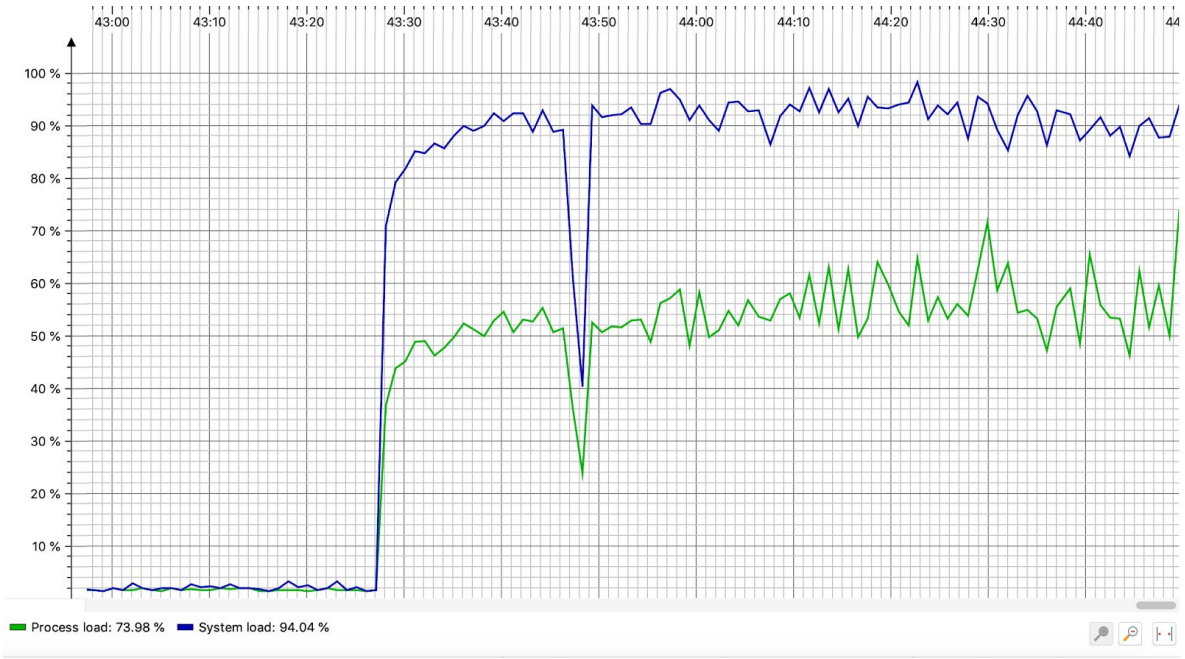
Charge augmentée exceptionnelle :



Mémoire RAM utilisée



Graph de ressources



Utilisation de CPU

Pour l'utilisation de Jmeter, il s'agit simplement de créer un Thread Group pour pouvoir exécuter plusieurs requête. À vrai dire, dans le thread group, nous spécifions trois paramètres, soit le *number of threads*, le *loop count* et le *ramp-up period*. Par ce fait, nous avons réussi à créer quatres scénarios pour comprendre l'utilisation de ressources par le container.

Ressources utiles:
JMeter : http://jmeter.apache.org/usermanual/get-started.html

Q5) Configurez le monitorage pour Docker. Définissez les seuils de CPU entre lesquels les objectifs de performance sont garantis. Implémentez la capacité

d'adapter l'infrastructure Docker de Weka pour répondre à l'augmentation de charge (en ajoutant des containeurs) ou à la réduction de charge (en retirant des containeurs). Utilisez la méthode « scale » de docker-compose pour faire l'adaptation. (20 points)

Pour la question 5, toutes les instructions se trouvent dans le vidéo. Cependant, voici les ressources utilisées afin de parvenir à répondre à la question:

Prometheus: https://prometheus.io/docs/introduction/overview/
CAdvisor: https://github.com/google/cadvisor
DockProm: https://stefanprodan.com/2016/a-monitoring-solution-for-docker-hosts-containers-and-containerized-services/
Docker-compose/scale : https://docs.docker.com/compose/reference/scale/
Docker monitoring : https://rancher.com/comparing-monitoring-options-for-dockerdeployments/

Q6) Vidéo : Démontrez le déploiement de Weka REST en Docker. Démontrez l'exécution du système avec peu de ressources et une charge de travail augmenté. Démontrez les métriques problématiques pour l'utilisation des ressources. Exécuter la commande « scale » pour ajouter des ressources et redémontrez les métriques corrigées. (15 points)

Lien vers la vidéo: https://youtu.be/HzymhWdQES0

Annexe 1: Description des métriques calculable, en lien avec le critère d'efficacité de performance

Métriques liées au comportement par rapport au temps:

Temps de réponse: Se définit comme le temps de bout à bout qu'une tâche prend pour traverser une partie du système. [cours 6]

Latence moyenne: Aussi appelé temps d'attente (*wait time*), ceci correspond au temps moyen requis pour recevoir le premier octet d'une requête, une fois cette dernière envoyée. [types, steps, best]

Temps moyen de chargement: Représente le temps moyen requis pour traiter l'entièreté de chaque requête d'un utilisateur sur le système. Vu la nature de Weka, des requêtes typiques doivent être définie. Les requêtes pourraient même être groupées selon leur catégories, et des temps moyen de chargement pourraient être établis pour chaque groupe, vu la nature changeante de ces dernières, selon l'utilisation de l'application.

Temps de réponse de pointe (*peak response time*): Ceci correspond à la mesure du temps maximal requis pour traiter une requête. Dans le cadre des tests de performance de type *peak load*, il est intéressant d'observer cette métrique. Cette dernière va d'ailleurs variée en fonction de la charge (*load*), car cette dernière peut varier. [linkedIn article]

Requêtes par seconde: Correspond au nombre de requête traité par seconde. Ceci est une application directe de la métrique de taux de traitement, pour une valeur de temps d'une seconde. Cette mesure pourrait être donnée en milliseconde, tout dépendant de l'application et des requêtes, en plus du paramétrage physique de la machine.

Utilisation du CPU (temps): Correspond au temps requis par le CPU pour traiter une requête. Cette mesure, comme bien d'autres, dépendant évidemment de l'architecture physique du système.

Taux de traitement (*throughput*): Mesuré en kilooctets par seconde, le taux de traitement mesure la quantité de bande passante utilisée durant les tests.

Métriques liées à l'utilisation des ressources:

Utilisation de la mémoire: Quantifie la quantité de mémoire nécessaire pour traiter une requête.

Utilisation du CPU (quantité) : Quantifie le pourcentage du CPU utilisé pour traiter une requête.

Utilisation de la RAM: Quantifie la quantité de mémoire nécessaire pour traiter une requête.

Métriques liées à la capacité:

Selon notre interprétation du sous-critère de capacité, ainsi que sa définition, les métriques des sous-critères lié au comportement par rapport au temps, ainsi qu'à l'utilisation des ressources sont valable comme métriques de la capacité. Cependant, ce sont les seuils, donc la valeur des objectifs, qui se verra changer en fonction des limitations maximales déterminées pour notre système.