LOG3430 - Méthodes de test et de validation du logiciel

Laboratoire 5

Simulations de tests de charge

Département de génie informatique et de génie logiciel
École Polytechnique de Montréal

Automne 2024

1 Introduction

Dans ce laboratoire, vous apprendrez ce qu'est un test de performance, comment utiliser un outil de test de performance, comment concevoir un test de performance et exécuter des tests de performance. En particulier, vous effectuerez des tests de charge sur un serveur web.

Les tests de performance sont un type de test permettant de déterminer la vitesse d'un ordinateur, d'un réseau ou d'un périphérique. Ils vérifient la performance des composants d'un système en transmettant différents paramètres dans différents scénarios de charge.

Le test de charge est le processus qui simule la charge réelle d'un utilisateur sur une application ou site Web. Il vérifie le comportement de l'application lors de charges normales et élevées. Ce type de test est appliqué lorsqu'un projet de développement touche à sa fin.

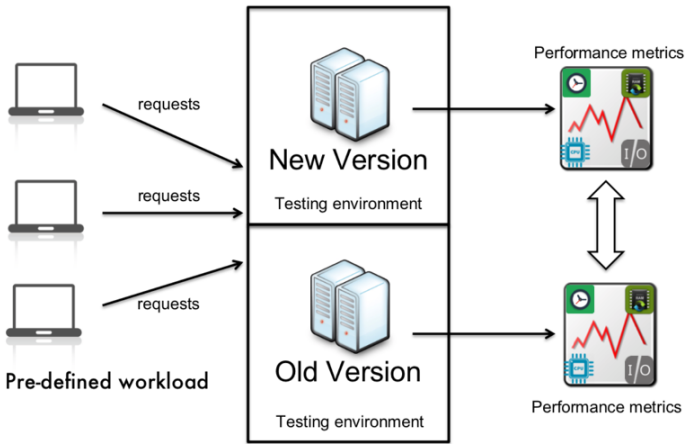
2 Objectifs

Les objectifs généraux de ce laboratoire sont :
— Apprendre à déployer un système pour faire des tests de charge
— Apprendre à configurer un pilote de charge
— Apprendre à exécuter un test de charge
— Pratiquer l'analyse des résultats de ce test de charge

3 Les tests de performance

Les tests de performance/charge consistent en trois processus : (1) concevoir une charge appropriée, (2) exécuter un test de charge et (3) analyser les résultats d'un test de performance/charge.

En pratique, les testeurs configurent l'environnement de test et définissent la charge de travail. Ensuite, les mêmes demandes sont envoyées pour tester le système et les mesures de performance sont collectées. Enfin, les testeurs comparent les deux mesures de performance et identifient la régression des performances (le cas échéant). Ci-dessous, une figure montre le mécanisme des tests de performance/charge :



Les outils

Il existe plusieurs outils pour aider à effectuer des tests de performance/charge.

- **JMeter** - Un outil open source qui peut être utilisé pour les tests de performances et de charge afin d'analyser et de mesurer les performances d'une variété de services.
- **LoadRunner** - Une version de test de performances d'entreprise de Loadrunner et une plate-forme ont permis la normalisation mondiale.
- **ReadLine13** - Une plate-forme de test de charge qui apporte la puissance à faible coût du cloud à JMeter et à d'autres outils de test de charge open source.
- **Locust** - Un outil open source simple et flexible, permettant de créer des tests personnalisés. Sa facilité d'utilisation est renforcée par une interface web interactive et une intégration fluide avec les pipelines CI/CD.

Pour ce TP on va utiliser Locust, pour plus de details consultez : https://locust.io/

Outils Requis

Système d'exploitation : macOS/Windows
Python : version 3.9+

Les tâches

Partie A - *Déployer un serveur web et Locust pour le test de charge*

- **Quelles sont les caractéristiques physiques de votre machine ?**
- Obtenez une copie locale de ce répertoire : `https://github.com/AltafAllahAbbassi/TP5_LOG3430`
- Créez un nouvel environnement et installez les dépendances
- Lancez le serveur web avec la commande : `python server.py`. Visitez `/fast` et joignez une capture d'écran
- Lancez le test de charge existant avec Locust et configurez le premier scénario :
  - Nombre d'utilisateurs : 10 000
  - Montée en charge : 100

  Accédez à l'interface web de Locust et joignez une capture d'écran des statistiques obtenues.
- Analyse des résultats :
  - **Combien de requêtes ont échoué ?**
  - **Quel endpoint a eu le taux de requêtes le plus élevé (requêtes par seconde) ?**
  - **Quel endpoint a montré le temps de réponse maximum le plus élevé, et que pourrait cela indiquer ?**
  - **Comment la performance du serveur pourrait-elle être améliorée en fonction de ces résultats ?**

Partie B - *Utiliser Locust pour simuler une charge de travail*

- Créez un nouvel utilisateur (*Heavy User*) qui effectue des requêtes plus lentement que le *Simple User*. Ajoutez une capture d'écran du code correspondant.
- Exécutez les deux scénarios suivants :
  - Nombre total d'utilisateurs : 1000
  - Durée : 2 minutes
  - Nombre d'utilisateurs par seconde : 10
  - Scénario 1 : 50% des utilisateurs sont des Simple Users et 50% sont des Heavy Users
  - Scénario 2 : 2% des utilisateurs sont des Simple Users et 98% sont des Heavy Users
- **Quelles méthode avez-vous utilisé ? Fournissez le code ou les commandes correspondantes.**
- Joignez le graphique généré ainsi que le ratio des utilisateurs.
- **Comparez les résultats des deux scénarios et expliquez les différences.**

Partie C - *Integration de test charge au pipeline CI/CD*

- Dans votre repo local, créez le répertoire *.github/workflows/* pour stocker votre fichier de workflow. Ce fichier va définir les étapes de déploiement.
- Dans le répertoire *.github/workflows/*, créez un nouveau fichier appelé *main.yml* et ajoutez-y le code suivant :

```
name: Run Locust Tests on Commit
on:
  push:
    branches:
      - main
jobs:
  locust-test:
    runs-on: ubuntu-latest

    steps:
      - name: Check out the repository
        uses: actions/checkout@v2
      - name: Set up Python
        uses: actions/setup-python@v2
        with:
          python-version: '3.10'
      - name: Install dependencies
        run: |
          python -m pip install --upgrade pip
          ............
      - name: Start Flask server
        run: |
          ............ &
          echo "Waiting for Flask server to start"
          sleep 30
```

— Complétez les étapes manquantes et poussez le code. **Quelles sont ces étapes ?** Joignez une capture des résultats du pipeline.
— Ajoutez une étape supplémentaire au pipeline pour lancer les tests de charge après chaque commit. Les tests de Locust doivent être configurés conformément à la partie A et arrêtés après 2 minutes.
Poussez à nouveau le code du pipeline sur GitHub avec un nouveau commit.
Indiquez l'étape que vous avez ajoutée et joignez une capture du fichier *.github/-workflows/main.yaml*.
Ajoutez également une capture d'écran des résultats du pipeline (avec l'ajout des tests de charge).

4 Livrables attendus

Les livrables suivants sont attendus :
— Un rapport pour le laboratoire. Le rapport doit contenir :
  — Vos réponses à la partie (A) : (4 points).
  — Vos réponses à la partie (B) : (3 points).
  — Vos réponses à la partie (C) : (2 points).
  — Qualité du rapport : (1 point).
— Le rapport doit être soumis au format PDF, nommé `matricule1_matricule2_lab4.pdf`, ainsi que le lien vers votre répertoire GitHub, qui doit être PUBLIC.
— L'absence d'une capture d'écran, un répertoire GitHub privé pour une question qui en fait la demande, ou d'un code source ou d'une capture explicitement demandée, entraînera une non-notation de cette question.

5 Information importante

1. Un retard de [0,24h] sera pénalisé de 10%, de [24h, 48h] de 20% et de plus de 48h de 50%.
2. Aucun plagiat n'est toléré. Vous devez soumettre uniquement le code et les rapports de couverture de code réalisé par les membres de votre équipe.