	# SPÉCIFICATIONS TECHNIQUES DES BESOINS ET DES EXIGENCES
Date de création	19/11/2014
Titre	STBE
Auteurs	T. Fleury, S. Labassi, A. Lavaire
Diffusion	T. Fleury, S. Labassi, A. Lavaire, E. Lécharny
Validation	T. Fleury, S. Labassi, A. Lavaire, E. Lécharny

Reformulation de la demande

Objectif du projet

L'objectif du projet est de pouvoir lancer des tests de charge configurables sur des serveurs afin de pouvoir évaluer leurs performances. Ce genre d'opération nécessite généralement l'utilisation de machines puissantes et onéreuses pour jouer le rôle des clients réalisant cette charge. C'est pourquoi ce projet vise à remplacer ces machines par un ensemble de machines beaucoup moins coûteuses telles que des Raspberry Pi.

Exigences du projet

Ce projet d'« injecteur de requêtes » doit donc se baser sur un ensemble de petites machines (processeur limité et faible quantité de RAM). Il va donc falloir être capable de :

- lancer tous les injecteurs de façon synchrone,
- les diriger à distance,
- récupérer les données produites pour les agréger afin de pouvoir les analyser.

Pour faciliter l'utilisation d'un tel système réparti sur un ensemble de machines, il faudra bien évidemment pouvoir déployer les différents "jeux de tests" sans avoir à les installer sur chaque injecteur. Ce déploiement devra pouvoir être piloté depuis une interface graphique. Celle-ci permettra également d'afficher les résultats des injections de requêtes.

En ce qui concerne les protocoles réseaux des serveurs à tester, la priorité est donnée à LDAP (qui correspond au besoin actuel du client). Le système d'injection de requêtes devra pouvoir être étendu plus tard à d'autres protocoles. Ces autres protocoles ne font cependant pas partie des objectifs du projet actuel.

Afin d'être capable de pousser le serveur au maximum de ses capacités, il faut adapter le nombre d'injecteurs à utiliser.

Chaque injecteur permet d'augmenter la charge d'un même pourcentage (sauf dans la cadre d'une limitation de la bande passante, ou si le serveur est déjà surchargé). Grâce à cette régularité, il est possible de déterminer le nombre adéquat d'injecteurs pour atteindre l'objectif (100%). Pour ce faire, il faut néanmoins prendre en compte le "temps de chauffe" du serveur. En effet, des optimisations (notamment de la JVM) peuvent se déclencher après un certain temps d'utilisation du serveur permettant ainsi à ce dernier de pouvoir gérer une plus grande charge réseau. Ainsi, pour adapter le nombre d'injecteurs, il faut se baser sur les capacités du serveur après son "temps de chauffe".

Ainsi, un test d'injection de requêtes doit suivre les étapes suivantes :

- lancement de N injecteurs,
- "chauffage" du serveur pendant un certain temps avec N injecteurs,
- démarrage du test proprement dit avec N injecteurs,
- agrégation des données générées sur les N injecteurs.

En fonction des données relevées par les injecteurs concernant les durées d'exécution des requêtes et des performances du serveur testé (il faudra donc fournir un moyen de les évaluer), l'utilisateur pourra déterminer le nombre d'injecteurs nécessaire pour pousser le serveur à 100% de ses capacités.

L'objectif du projet étant de réaliser une économie au niveau des injecteurs de requêtes, certaines optimisations pourront éventuellement être réalisées au niveau de ces derniers.

Dans le cadre du protocole LDAP, les requêtes envoyées au serveur sont au format binaire et la traduction des requêtes vers le binaire peut être une opération coûteuse en temps processeur. Un système de précompilation de ces requêtes et de leurs réponses prévues pourra alors être mis en place afin de permettre un plus grand envoi de requêtes.

Ce genre d'optimisation n'est cependant pas nécessaire au bon fonctionnement du projet et ne sera traité que si les délais le permettent.

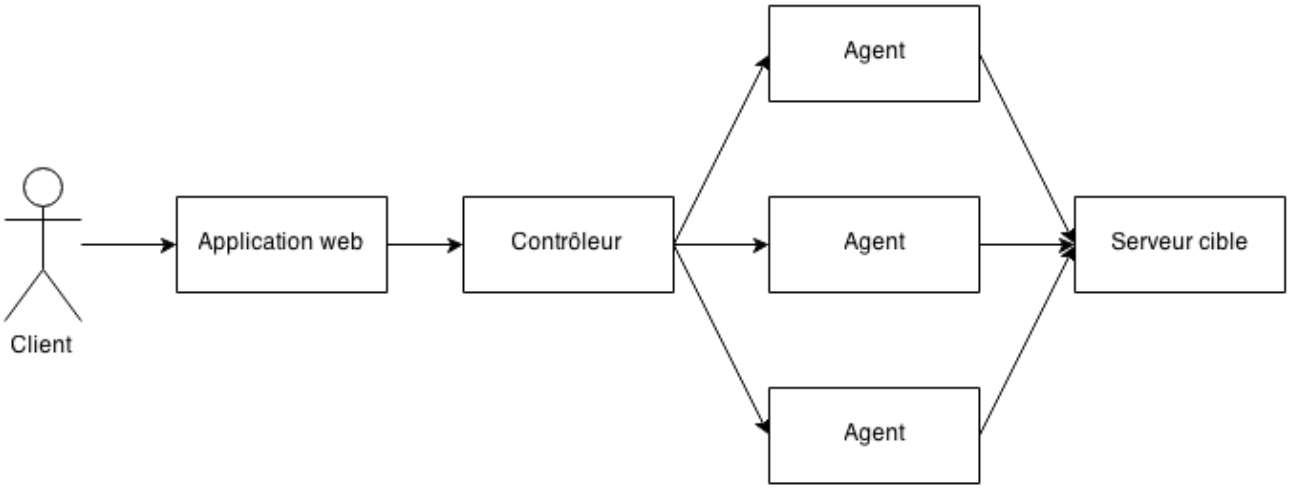
Délai du projet

Le produit doit être livré à la fin du mois de janvier 2015.

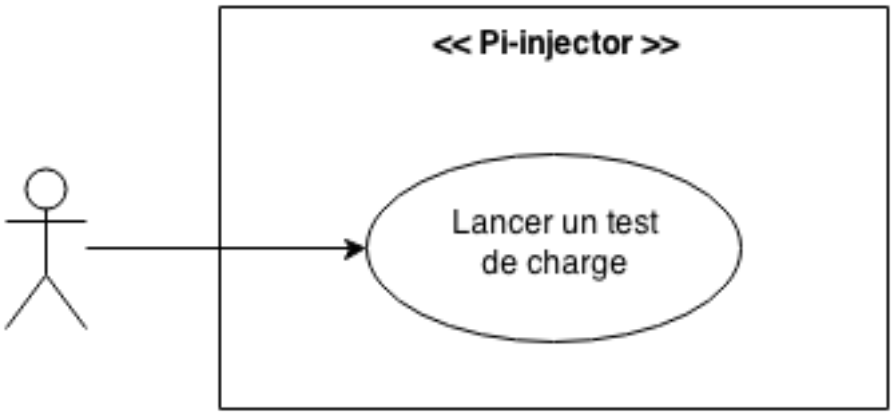
Analyse du problème

Business model

Schéma des différents composants de notre application :



Business use case



Notre projet n'offre qu'une seule fonctionnalité, celle d'effectuer des tests de charge sur un serveur.

L'utilisateur, possesseur d'un serveur à tester, va demander l'exécution d'un test de charge sur ce serveur. Le test consiste à utiliser plusieurs injecteurs (des Raspberry Pi) qui vont envoyer le plus de requêtes possible au serveur afin de le pousser au maximum de sa charge. Les injecteurs établissent des statistiques sur le nombre de requêtes envoyées chaque seconde afin d'établir les capacités du serveur à offrir un service optimal. L'utilisateur reçoit le résultat du test sous forme de tableau récapitulatif ou d'un graphique (optionnel).

Spécifications fonctionnelles

Contrôleur

Le contrôleur va nous permettre d'envoyer des tâches (task) aux différents agents. Chaque tâches va contenir du code Java qui va être exécuté par les agents. Ce code va contenir un scénario (défini dans la partie suivante).

Au niveau du contrôleur, nous allons lui passer en paramètre (via notre interface graphique) le nombre d'injecteurs qui vont être utilisés pour le test de charge (par défaut, nous mettons tous les Raspberry PI disponibles sur le réseau). Il aura également en paramètre le temps de chauffe nécessaire à la configuration ainsi que la durée d'exécution du test (ou le nombre d'itérations du scénario).

Le temps de chauffe donné va nous permettre d'ignorer les données reçues en début de test et de les exclure du résultat final. Lors du commencement de l'exécution des requêtes, il va y avoir du coté du serveur testé, ainsi que du coté des agents, des optimisations qui vont être déclenchées. Cela signifie donc qu'à un certain moment, notre nombre de requêtes par seconde va augmenter, soit parce que le serveur est capable de les traiter plus rapidement, soit parce que les agents parviennent à en envoyer plus. Il est donc possible de ne pas prendre en compte ce phénomène dans l'analyse des résultats obtenus.

Le nombre d'injecteurs va nous permettre, comme son nom l'indique, de préciser combien d'injecteurs vont être utilisés pour le test. En effet, dans la majorité des cas, nous allons vouloir utiliser tous les injecteurs disponibles. Mais dans le cas où le serveur à tester a déjà atteint sa capacité maximale, on pourra diminuer le nombre d'injecteurs à utiliser.

Scénario

Pour chaque test, nous allons vouloir lancer une série de requêtes (pouvant être de natures différentes). Cette série va être notre scénario. Les scénarios vont être renseignés par l'utilisateur via notre interface graphique.

Chaque scénario va contenir l'adresse du serveur cible (IP et port) et dans le cas du serveur LDAP, le DN (Distinguished Name), le nom d'utilisateur et le mot de passe. On pourra ensuite choisir son type de requête et l'ajouter à notre scénario.

Dans une première version, les scénarios déployés sur chaque agent vont tous être identiques. Cela va éviter les incohérences qui pourraient être créées via l'utilisation des scénarios. Si par exemple un scénario ne fait qu'ajouter des utilisateurs et un autre ne fait que les supprimer, il faudra être sûr que le premier scénario est bien exécuté avant le deuxième.

Nous allons enregistrer des informations sur chaque requête qui va être exécutée : l'heure de début de l'exécution, le temps total de l'exécution et le type de la requête. Une fois le test de charge terminé, ces données seront analysées par le contrôleur.

Agent

Les agents vont exécuter les tâches soumises par le contrôleur et contenant le scénario précédemment défini par l'utilisateur. Ces tâches consisteront à envoyer en boucle les requêtes du scénario durant le temps du test, défini lui aussi par l'utilisateur. Lorsque ce temps sera écoulé, les agents retourneront les temps d'exécution des requêtes au contrôleur.

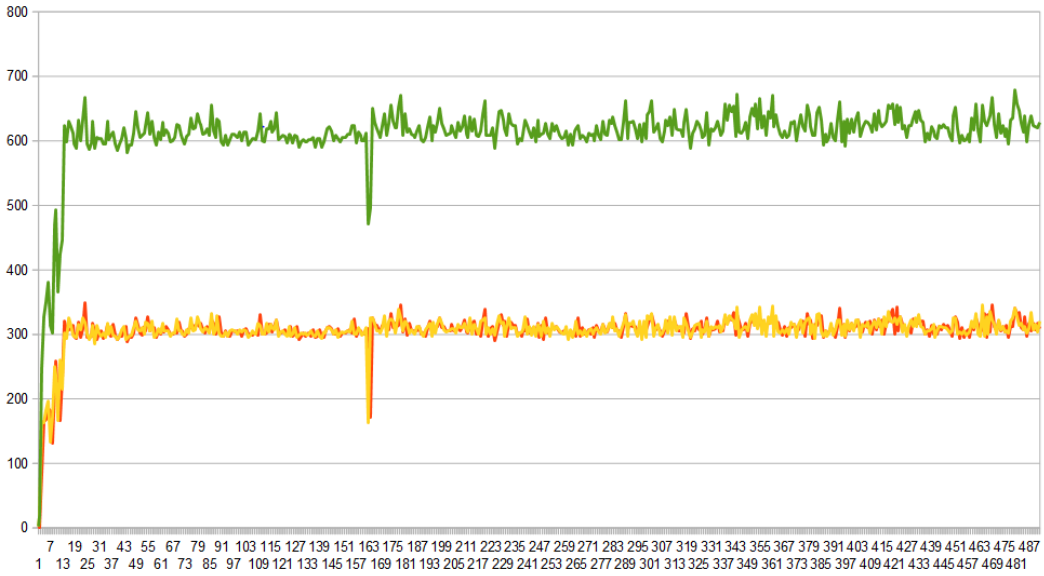
Les traitements effectués par les agents sont donc peu complexes, et ceux-ci seront donc la plupart du temps en attente de la réponse du serveur testé. Afin de maximiser le rendement de nos injecteurs (les utiliser au maximum de leur capacité), une des solutions possibles serait de lancer plusieurs exécutions parallèles d'un scénario sur un même injecteur. Mais il ne faut pas oublier que les injecteurs dont nous disposons ont un processeur mono CPU qui n'est donc pas optimisé pour une exécution parallèle.

Résultat

Comme dit plus haut, chaque agent va renvoyer au contrôleur une liste de résultats, les temps d'exécution des requêtes envoyées par l'injecteur. Il va donc falloir agréger cette liste en un seul résultat.

Le résultat que nous voulons afficher est le nombre de requêtes par seconde pour tous les injecteurs mais aussi pour chaque injecteur. Pour cela, il faut savoir pour chaque injecteur le moment où il a commencé l'exécution du scénario. Nous avons donc un système de temps relatif. Avant son exécution, chaque injecteur va sauvegarder l'heure à laquelle il commence (le référentiel de l'heure est le même pour chaque injecteur car ils le récupèrent via internet ; il ne faut juste pas oublier de configurer chaque injecteur pour lui préciser son fuseau horaire). Au moment où tous les injecteurs vont envoyer leurs résultats au contrôleur, celui-ci va récupérer l'injecteur qui aura commencé en premier son exécution (qui aura donc comme temps initial 0) puis pour tous les autres injecteurs, qui auront comme temps la différence entre leur temps initial et le temps de référence. Une fois cela fait, il ne reste plus qu'à additionner le nombres de requêtes pour chaque injecteurs.

Grâce a ce résultat, nous allons pouvoir tracer un graphique qui aura pour ordonné le nombre de requête par seconde et en abscisse le temps. Il y aura une courbe qui représentera le nombre de requête total puis chaque injecteur aura sa courbe.

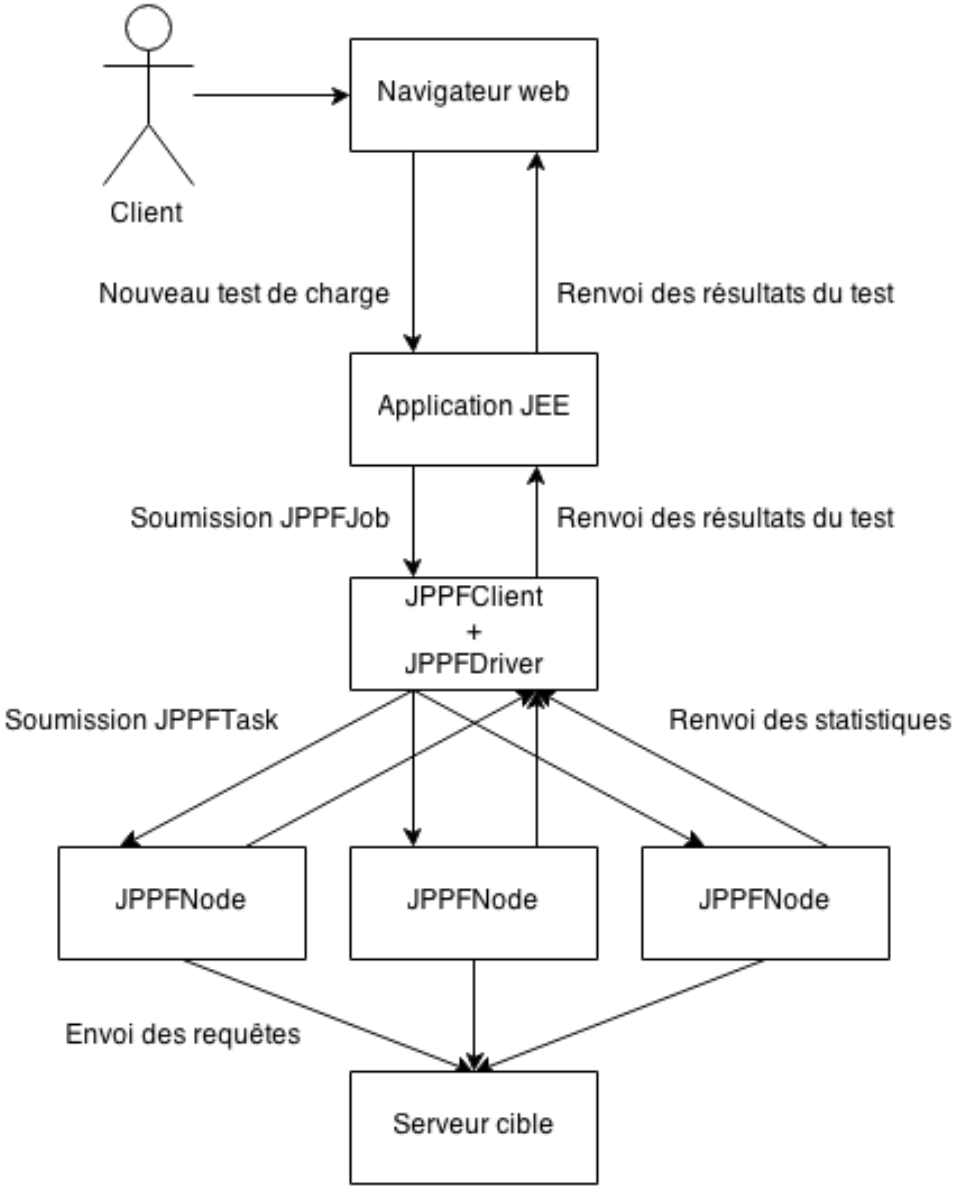


Contraintes techniques

Tous les outils inclus au projet rendu doivent être compatibles avec la licence : Apache Licence 2.0. Le fonctionnement de cette licence est décrit sur le site de la fondation Apache.

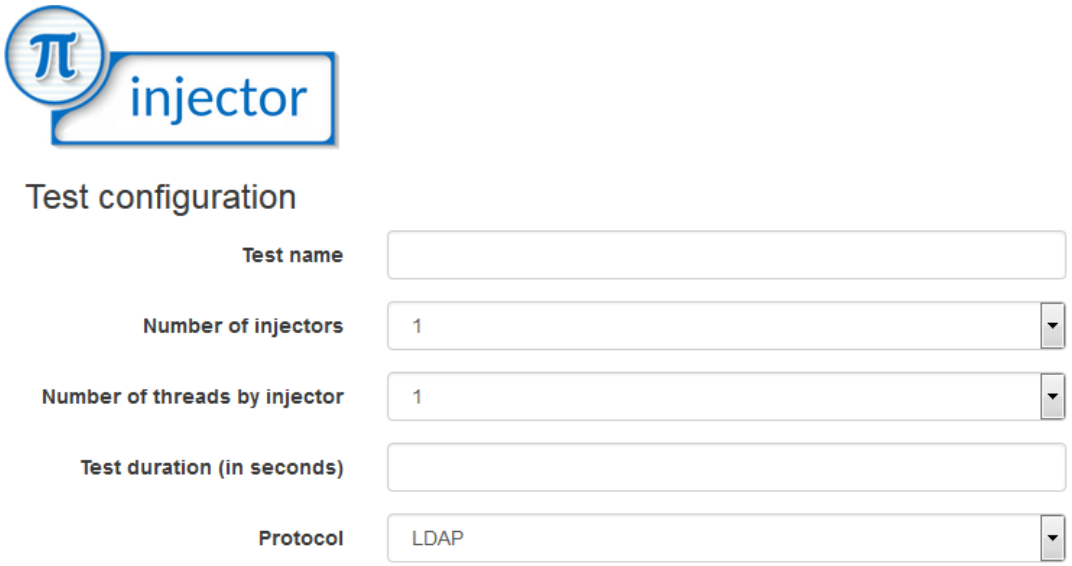
Conception de la solution fonctionnelle

Modèle de l'applicatif

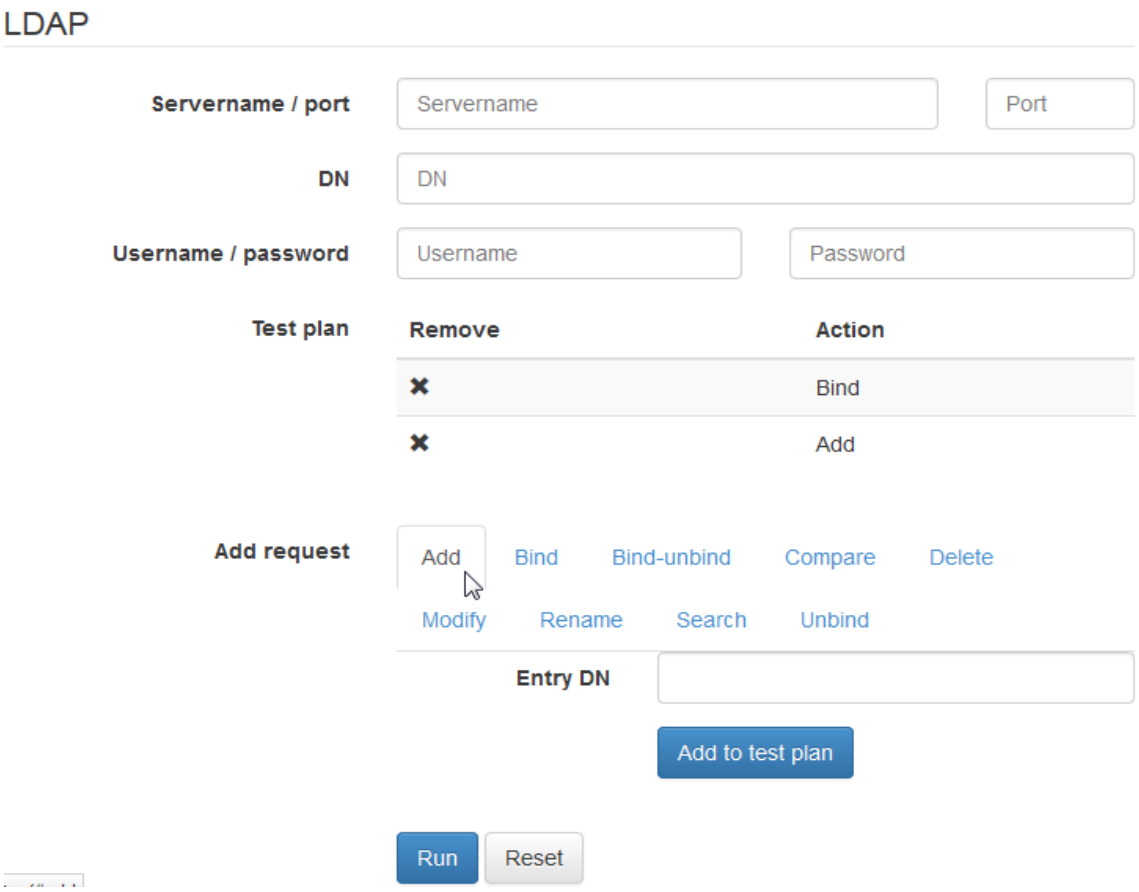


Interface graphique

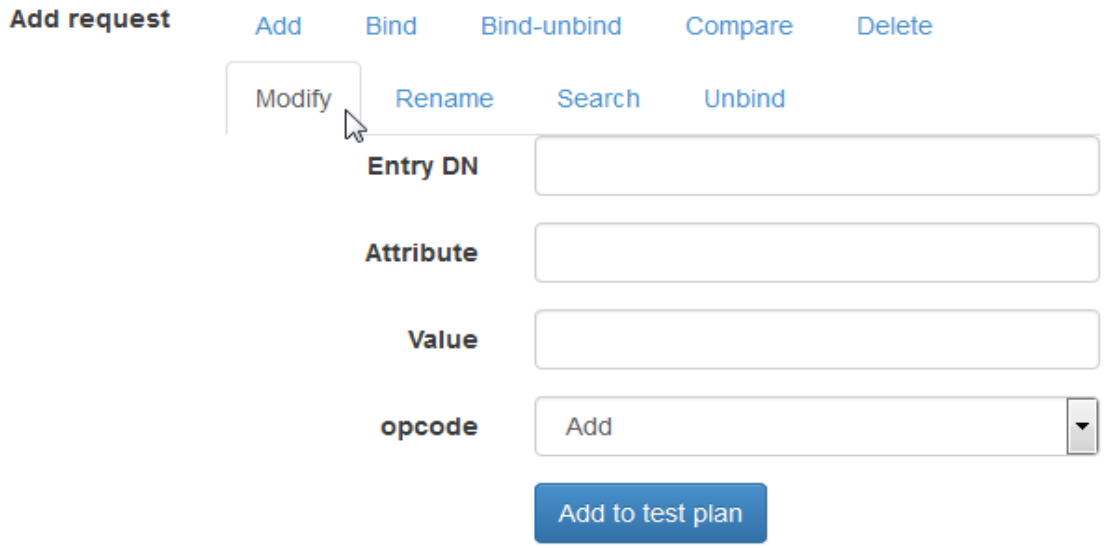
Voici à quoi devrait ressembler l'interface graphique de l'application web, qui permet à l'utilisateur de configurer les tests de charge, les lancer et consulter les résultats.



Voici l'interface de configuration du test de charge. La première partie permet de donner un nom au test (utilisé pour sauvegarder les résultats), le nombre d'injecteurs à utiliser, le nombre d'agents par injecteur, la durée du test et le protocole à utiliser. En fonction de ce dernier paramètre, la deuxième partie de l'interface graphique change, permettant de configurer le scénario à exécuter.



Dans le cas du protocole LDAP, on peut définir plusieurs paramètres propres au serveur. La partie "test plan" va définir le scénario à exécuter. On ajoute les requêtes à exécuter grâce à la partie "add request", où sont proposés les différents types de requêtes possible pour le protocole LDAP.

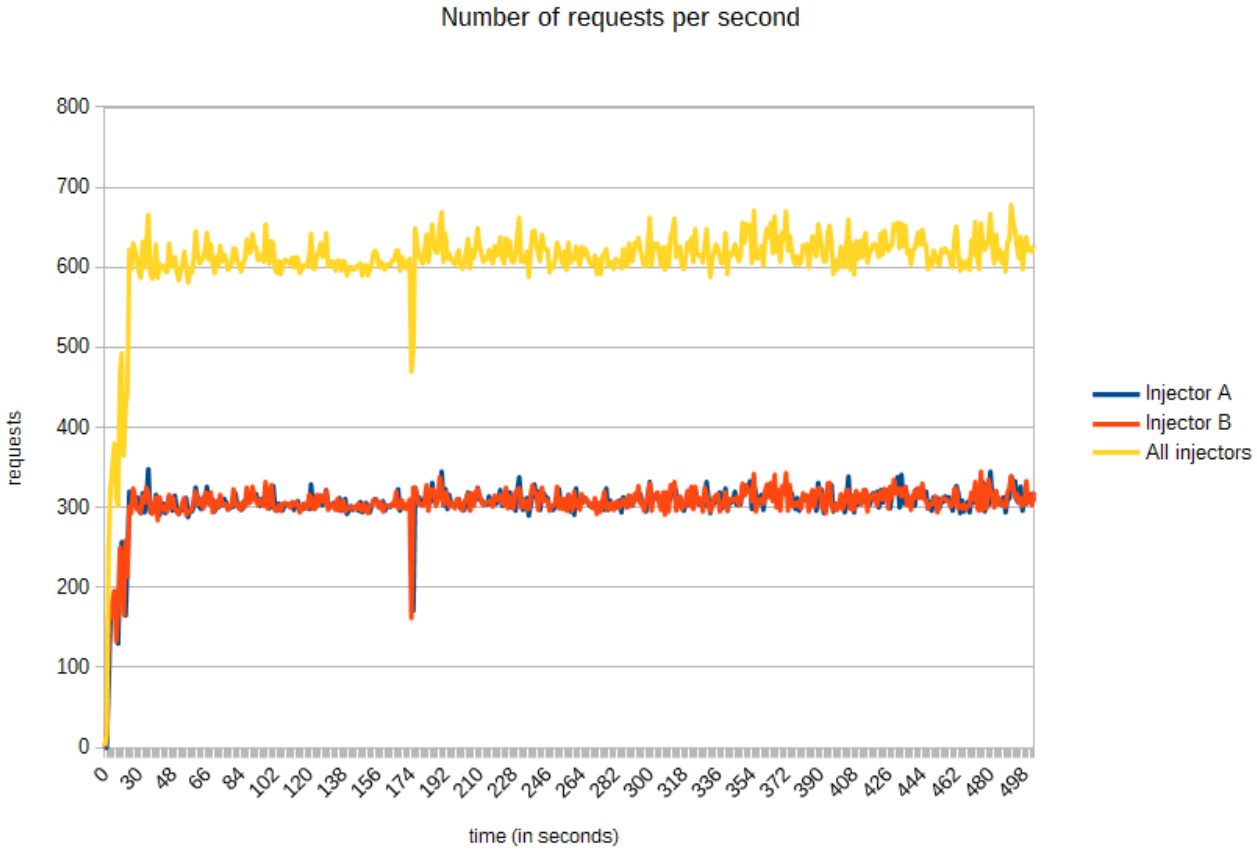


Selon le type de requête, il peut être nécessaire d'ajouter des paramètres, qui s'afficheront lors de la sélection du type de la requête. Le bouton "add to test plan" ajoute la requête à la suite des autres pour former le scénario. Une fois le scénario configuré, l'utilisateur peut lancer l'exécution du scénario.

Voici la page d'accès aux tests effectués. Il est possible de les supprimer et surtout de les consulter.

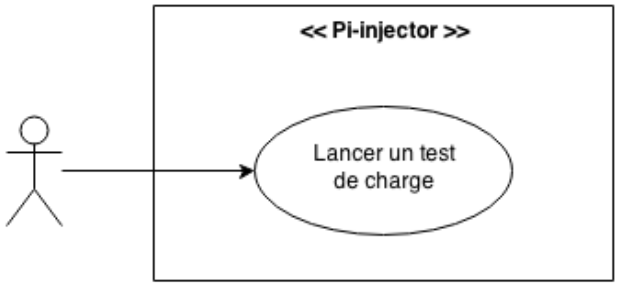
Test LDAP 3

Download CSV



Pour un test de charge, nous affichons un graphique représentant le nombre de requêtes envoyées sur le serveur par seconde. Nous fournissons aussi un fichier CSV, qu'il est possible de télécharger, reprenant les données sous forme de tableau.

Use case applicatif



Lancer un test de charge

- L'utilisateur se connecte à l'application web.
- L'utilisateur configure le test de charge à lancer (adresse du serveur, N injecteurs, type de requêtes pour le scénario...).
- L'utilisateur lance le test de charge.
- Le client JPPF reçoit le scénario à exécuter.
- Le client JPPF envoie ce scénario à N injecteurs.
- Les injecteurs exécutent le scénario et enregistrent pour chaque seconde le nombre de requêtes envoyées.
- Les injecteurs retournent les résultats au client JPPF une fois le scénario terminé.
- Le client JPPF agrège les résultats des injecteurs.
- Le client JPPF retourne le résultat du test de charge sous forme de tableau de données (ou de graphique, optionnel).

Contraintes legacy

Notre application ne s'intègre pas à un projet ou une architecture déjà existante, nous fournissons un logiciel "stand-alone". Néanmoins, l'utilisation de notre solution implique plusieurs contraintes : il faut déployer plusieurs éléments.

Raspberry Pi

Tout d'abord, il faut bien sûr des Rapsberry Pi avec comme OS installé Raspbian (1.3.10). Sur chacune des Raspberry Pi, il faut avoir déployé le dossier JPPF-4.2.4-node qui va servir à notre application. Puis sur chaque Raspberry Pi, il faut lancer le fichier nommé startNode.sh.

Pour de meilleures performances (pour éviter une perte trop importante de temps à cause du réseau), il est conseillé de mettre les Raspberry Pi ainsi que la machine qui va servir de contrôleur sur un même LAN.

Machine contrôleur

Au niveau de la machine contrôleur, il suffit juste de récupérer le dossier JPPF-4.2.4-driver et de lancer le fichier startDriver (.sh ou .bat suivant l'OS installé sur la machine). Il faut aussi lancer le projet via le fichier ANT nommée build.xml (commande run pour l’exécuter).

Que ce soit sur la machine contrôleur ou serveur, il faut Java 8 d'installé (Java 8 ARM pour le Raspberry Pi).

Analyse des risques

Technique

Notre équipe est composée de trois personnes qui sont développeurs. Chacun de nous maîtrise l'environnent de développement utilisé ici et nous avons déjà eu l'occasion de réaliser ensemble plusieurs projets qui ont pu aboutir.

Nous nous sommes engagés à fournir le nombre de jours/homme présent dans notre WBS. Si notre WBS contient une mauvaise estimation des besoins humains, cela risque d'être répercuté sur le contenu optionnel de ce projet mais aussi sur le contenu essentiel, même ci cela est peu probable.

Fonctionnel

Toute notre solution tourne autour du framework JPPF. Le risque que cette solution ne nous permette pas d'accomplir notre projet est moindre mais néanmoins présente.

Nous avons donc une solution alternative nommée JMeter (solution offerte par Apache). Cette solution a fait partie de notre grille de lecture et a rempli une partie des demandes faites par le client. Nous devrons donc modifier cette solution pour la rendre compatible avec la demande reçue.

Il faut aussi prendre en compte la puissance des Raspberry Pi. Les modèles utilisés sont des Raspberry Pi modèle B avec 512Mo de RAM et un CPU de 700MHz. Ce sont donc des machines peu puissantes, chose qu'il faut prendre en compte lors de la conception.

Pour ce projet, de nombreuses configurations, qu'elles soient physiques ou logicielles, sont nécessaires. Il faudra bien prendre en compte une durée importante pour ces tâches non négligeables.

Plan d'action

WBS

- Réalisation d'un injecteur de requêtes [70j/h]
  - Définition des besoins [4j/h]
  - Etablir une grille de lecture des logiciels répondant partiellement à la demande [8j/h]
    - Installation et tests des logiciels [4j/h]
    - Déterminer les fonctionnalités intéressantes de ces logiciels [4j/h]
  - Conception architecturale [7j/h]
    - Architecture basée sur Jmeter [3j/h]
    - Architecture basée sur JPPF [4j/h]
  - Mise en place de l'environnement de test [5j/h]
    - Installation et configuration de Apache Directory Server (serveur LDAP) sur le serveur cible [3j/h]
    - Configuration des agents (Raspberry Pi) [2j/h]
      - Installation et configuration de l'OS Raspbian [1j/h]
      - Installation de Java 8 [1j/h]
  - Mise en place des prototypes [6j/h]
    - Déploiement et test du prototype utilisant le logiciel Jmeter [2j/h]
    - Déploiement et tests d'une application simple basée sur le framework JPPF [4j/h]
  - Développement des fonctionnalités du produit [40j/h]
    - Réalisation du lot 1 : Injection des requêtes depuis les Raspberry Pi [22j/h]
      - Développement de la fonctionnalité d'injection des requêtes LDAP [6j/h]
      - Développement d'un mécanisme permettant la définition des scénarios de requêtes [6j/h]
      - Mise en place d'une politique de déploiement des tâches d'injection sur les différents agents [5j/h]
      - Mise en place d'un système de récupération de d'aggrégation des temps d'exécution des requêtes [6j/h]
    - Réalisation du lot 2 : Interface graphique de contrôle des injections et d'affichage des résultats [15j/h]
      - Développement de l'interface graphique [11j/h]
      - Intégration du lot 2 [4j/h]

- Livraison du produit final [3j/h]