	# SPÉCIFICATIONS TECHNIQUES DE RÉALISATION
Date de création	21/01/2015
Titre	STR
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

Présentation du projet

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 », nommé Pi-Injector, 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) avec l'implémentation de cinq types de requêtes (Bind, Add, Search, Delete et Unbind). 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.

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.

Choix techniques

Identification des logiciels de base utilisés (systèmes, réseaux, compilateur, bibliothèque, framework, composant, etc.), gestionnaire des configurations documentaire, logiciel.

Java 8, Raspbian,

Framework JPPF 4.2.3, Apache DS api, Tomcat 7.0.15 embedded,

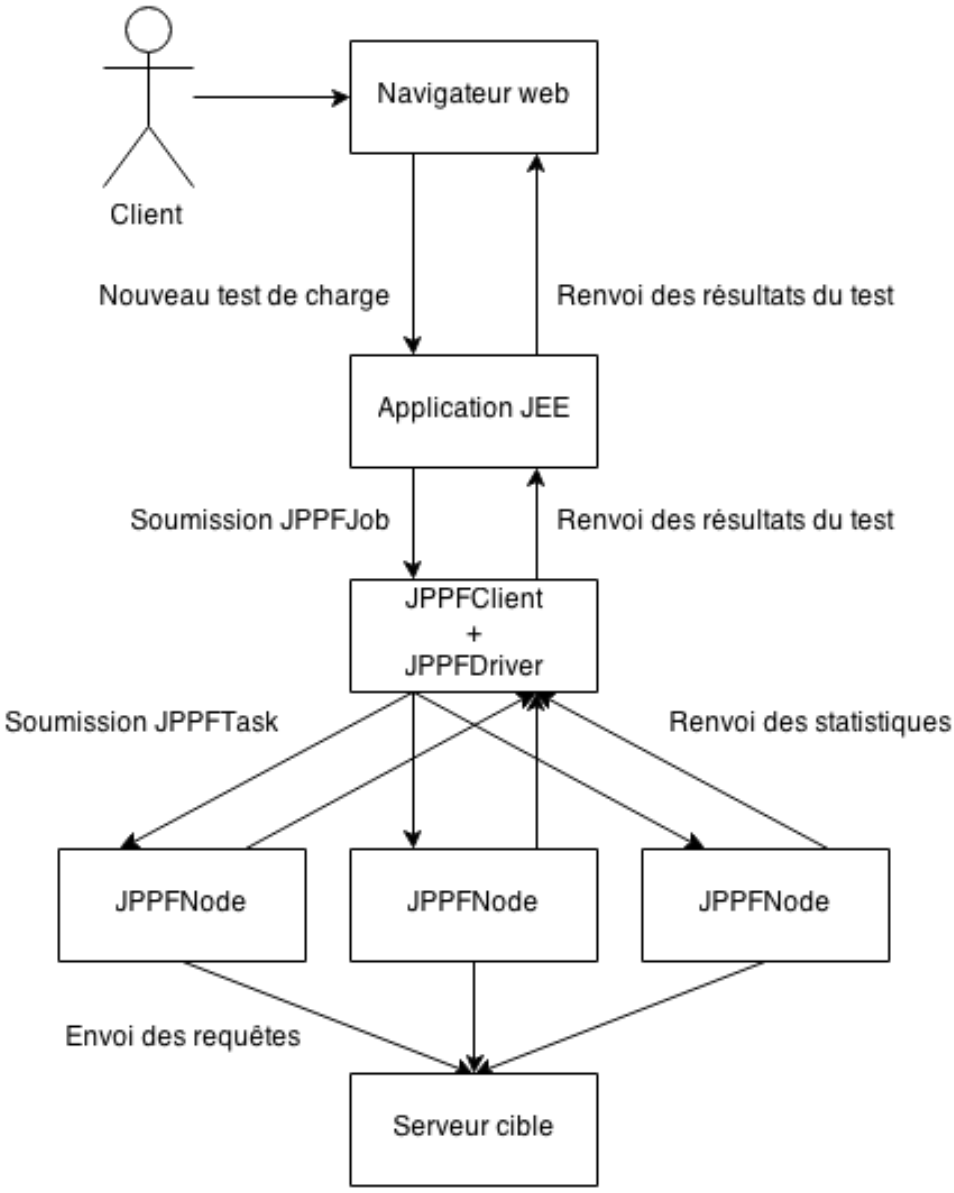
Maven, Git

Au niveau de l'interface web, nous avons utilisé Bootstrap 3.2.0 (licence MIT) pour la mise en forme et jQuery 1.9.0 (licence MIT) pour ses fonctionnalités JavaScript (et parce qu'il est requis pour utiliser Bootstrap). Pour l'affichage des courbes de résultats, nous utilisons Chart.js 1.0.1 (licence MIT).

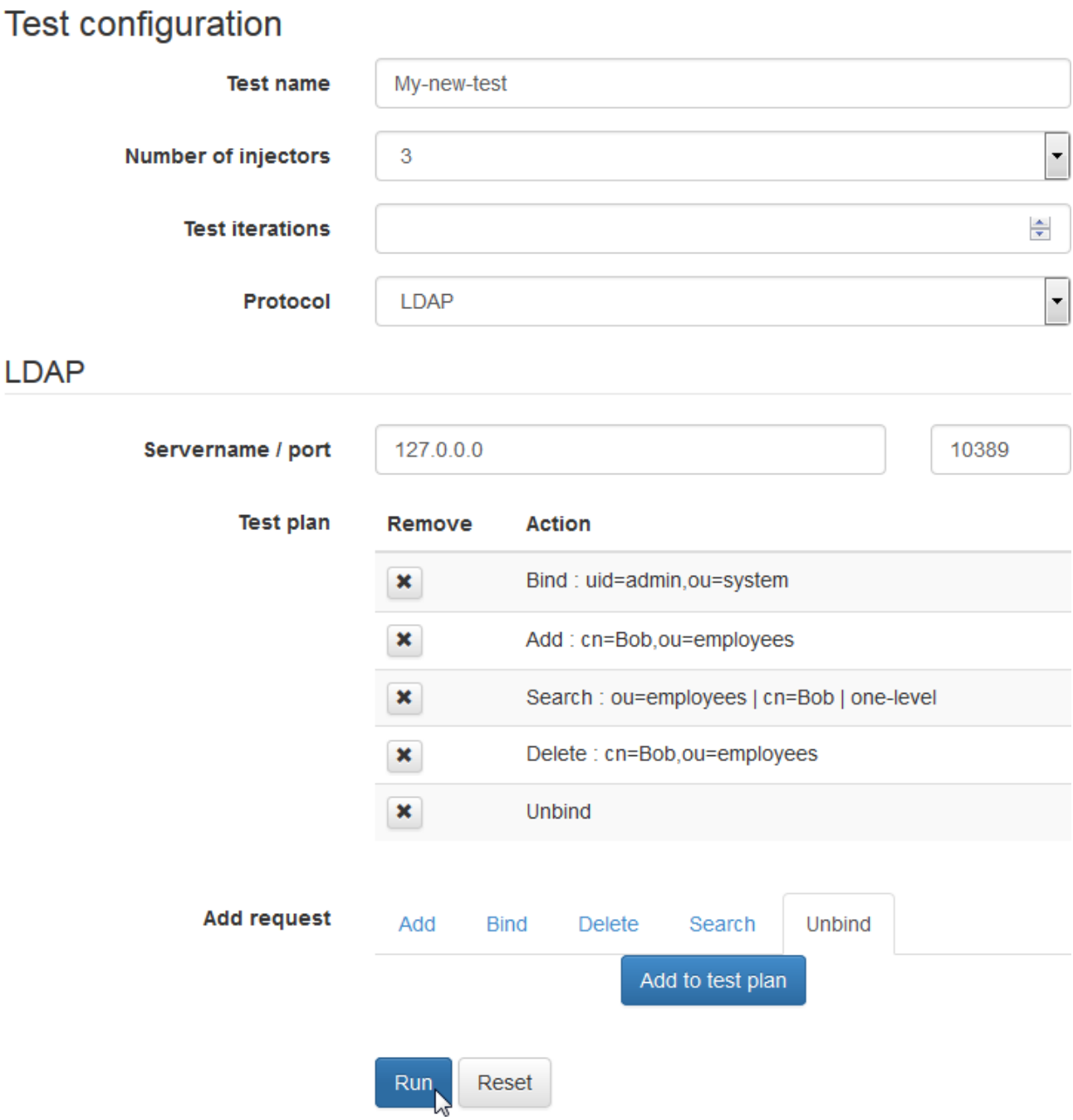
D'autres librairies ont été incluses au projet (commons-io, commons-lang...), mais leur importances et utilisations étant limitées, nous ne nous attarderons pas dessus. Néanmoins, nous nous sommes assurés qu'elles respectaient bien la licence Apache 2.0.

Architecture technique

Voici un schéma représentant l'architecture générale de la solution développée, avec un exemple de création d'un test de charge.



Lorsque les nodes JPPF (sur les Raspberry Pi), le driver JPPF et l'application (sur la même machine) sont déployés, l'interface graphique est disponible par navigateur web à l'adresse « localhost:8080 ». Depuis celle-ci, il est possible de lancer un nouveau test de charge, de voir le taux d'avancement des tests en cours d'exécution et d'afficher sous forme de graphique les résultats des tests précédents.



Pour la création d'un test de charge, l'utilisateur remplit le formulaire disponible et le soumet à l'application.

Celle-ci reçoit les données et créé un script représentant le scénario. Ensuite, ce scénario est envoyé à tous les injecteurs.

Chaque injecteur (Raspberry Pi) va ensuite exécuter le script en boucle (le nombre d'itérations étant défini par l'utilisateur) en enregistrant les statistiques sur les temps d'exécution de chaque requête envoyée au serveur cible.

Une fois l'exécution du test terminée, l'application va agréger les résultats des différents nœuds pour déterminer les données à afficher (le nombre de requêtes par seconde pour chaque injecteur et par tous les injecteurs en même temps). Ces données sont enregistrées sur le disque au format csv.

L'interface graphique permet de lire ces fichiers pour afficher un graphique représentant les courbes.

Plan d'intégration

Détermination du cycle de vie des configurations. Caractérisation des versions livrables de l'applicatif. Séquence de réalisation des modules.

Plan de test

Définition des procédures et des méthodes de test (par exemple, intégration continue). Relation entre tests unitaire et tests globaux. Scénario de pilotage des tests. Définition du tableau de bord et suivi de la réalisation.

Non régression

Définition des tests permettant de vérifier que l'intégration d'un nouveau module ne créé pas de perturbations dans l'exécution des modules déjà intégrés. Ainsi que dans le fonctionnement global de la version ainsi créée notamment en terme de validation de la charge.

Fonctionnel

Scénario de validation des fonctionnalités offertes. Détermination de qui les effectue et qui les valide.