	# 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

Lors de nos premières rencontres avec le client, il nous avait fournit une liste de projets développés en Java qui pouvaient répondre à ses attentes. Parmi cette liste, nous avons choisi de travailler avec le framework JPPF (présenté plus loin) car il répondait parfaitement aux attentes du client. Nous avons donc développé notre application sous Java 8.

Java est un langage portable, c'est-à-dire qu'il est indépendant de toute plate-forme. Il est sûr, la sécurité fait partie intégrante du système d'exécution et du compilateur, et il est multitâche, c'est-à-dire qu'il permet l'utilisation de threads. La JVM utilise elle-même plusieurs threads.

Surtout, tous les membres de notre équipe ont une forte expérience en Java, c'est un langage que nous pratiquons depuis plusieurs années.

JPPF est un framework qui permet à des applications de s'exécuter sur un ensemble de machines appelée « grille distribuée ». Pour ce faire, JPPF distingue trois éléments différents :

- Le client
- Le driver
- Les nœuds

Un nœud est un élément qui va exécuter une « task », qui est un ensemble d'instructions (un peu comme un Runnable est exécuté par un thread dans la technologie Java). Le driver est l'élément auquel les nœuds se connectent. Il va ainsi pouvoir attribuer à ceux-ci les « tasks » qu'ils doivent exécuter suivant un algorithme de répartition qui peut être configuré. Enfin, le client est l'élément qui permet de créer un ensemble de « tasks » réunies au sein d'un « job ». Ce « job » sera soumit au driver pour que les « tasks » contenues soient ensuite exécutées sur les nœuds.

JPPF nous a ainsi permis d'établir une « grille » de Raspberry Pi facilement, tout en ayant un moyen de les administrer à distance via le driver et le client. Par ailleurs, JPPF offre certaines possibilités intéressantes :

- Changement des ports de communication entre les différents éléments via des fichiers de configuration afin de pouvoir passer les routeurs et firewall.
- Connexion automatiques des nœuds au driver grâce à un système de broadcast. Ainsi, lorsqu'un nouveau nœud est démarré, le driver le détecte automatiquement (s'ils sont sur un même LAN).
- Possibilités d'encapsuler les communications en SSL/TLS.

L'API Apache Directory LDAP est une « brique » de développement Java qui fournit des services afin de permettre une utilisation facile du protocole LDAP. Cette API a l'avantage d'être compatible avec tous les serveurs LDAP disponibles sur le marché. Elle a aussi l'avantage d'être développée par notre client (et son équipe) ce qui nous offre la possibilité d'avoir une aide rapidement si besoin.

L'une des exigences du projet était de proposer une interface web pour pouvoir piloter les tests de charge à distance. Pour ce faire, nous avons décidé d'utiliser le serveur web (serveur de servlet / jsp) Apache Tomcat 7.0.15.

En effet, notre projet étant développé avec la technologie Java, notamment avec l'utilisation des « briques » JPPF et Apache Directory LDAP API (dont nous parlerons après), nous avons préféré utiliser un serveur web qui utilise lui aussi cette technologie afin d'homogénéiser notre projet. Ce choix a également été motivé par les connaissances que nous avions déjà de ce serveur (contrairement à Jetty qui fait la même chose).

Cependant, Tomcat est un serveur web qui nécessite d'être lancé pour pouvoir ensuite y déployer des applications web. Il aurait donc fallu demander à nos utilisateurs d'installer eux-même ce serveur pour qu'ils y déploient notre application.

Cette méthode étant trop contraignante, nous avons préféré utiliser la possibilité d'embarquer le serveur Tomcat dans notre application afin que nos utilisateurs n'aient qu'à exécuter un seul fichier (celui de notre application) pour que tout soit lancé d'un coup. Ceci a été possible grâce au plugin Maven « tomcat7-maven-plugin » qui nous génère notamment deux fichiers :

- Un fichier war qui est le point d'entrée de notre application web,
- Un fichier jar qui va lancer le serveur Tomcat et y déployer directement notre application.

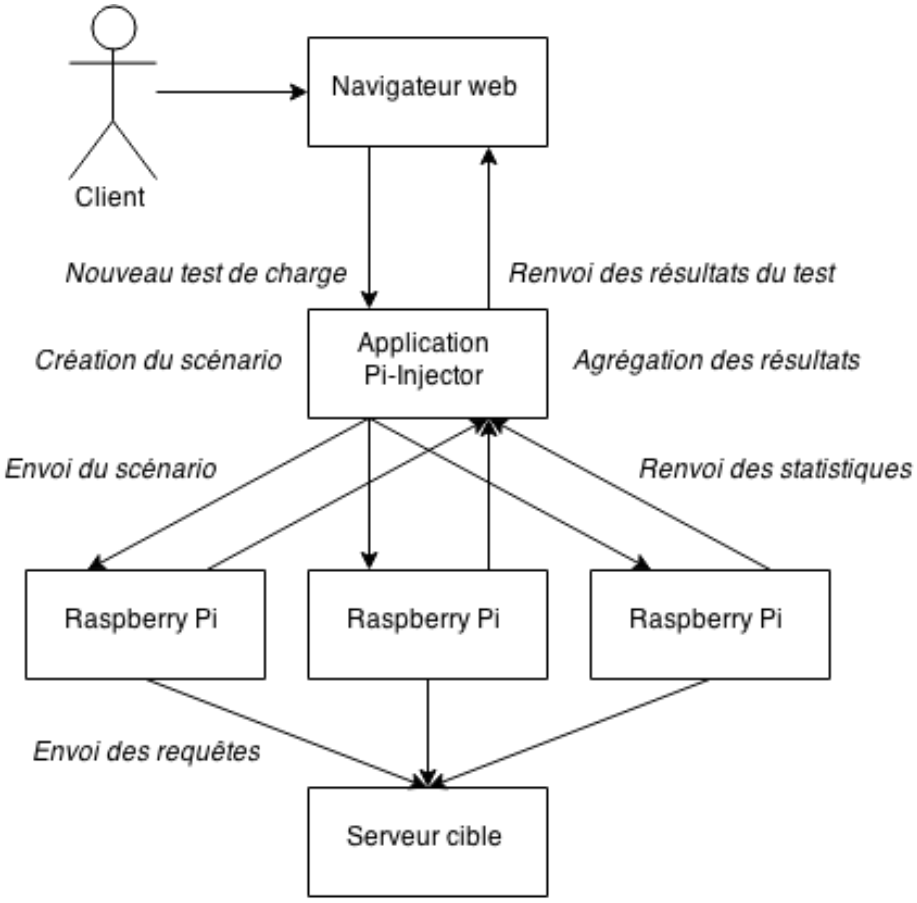
Pour les Raspberry Pi, nous avons choisi d'utiliser Raspbian 1.3.10. C'est un système d'exploitation basé sur la distribution GNU/Linux Debian et il est optimisé pour être installé sur un Raspberry Pi.

La Raspberry Pi est une machine dotée d'une faible puissance de calcul et de peu de RAM. Par conséquent, il est préférable d'installer un système optimisé pour ces machines. De plus, Raspbian étant un dérivé de Debian, nous avons accès à une majeure partie de la documentation de Debian. Enfin, Raspbian est la distribution la plus utilisée pour Raspberry Pi, elle bénéficie donc d'une large communauté, active, ce qui peut être très utile.

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).

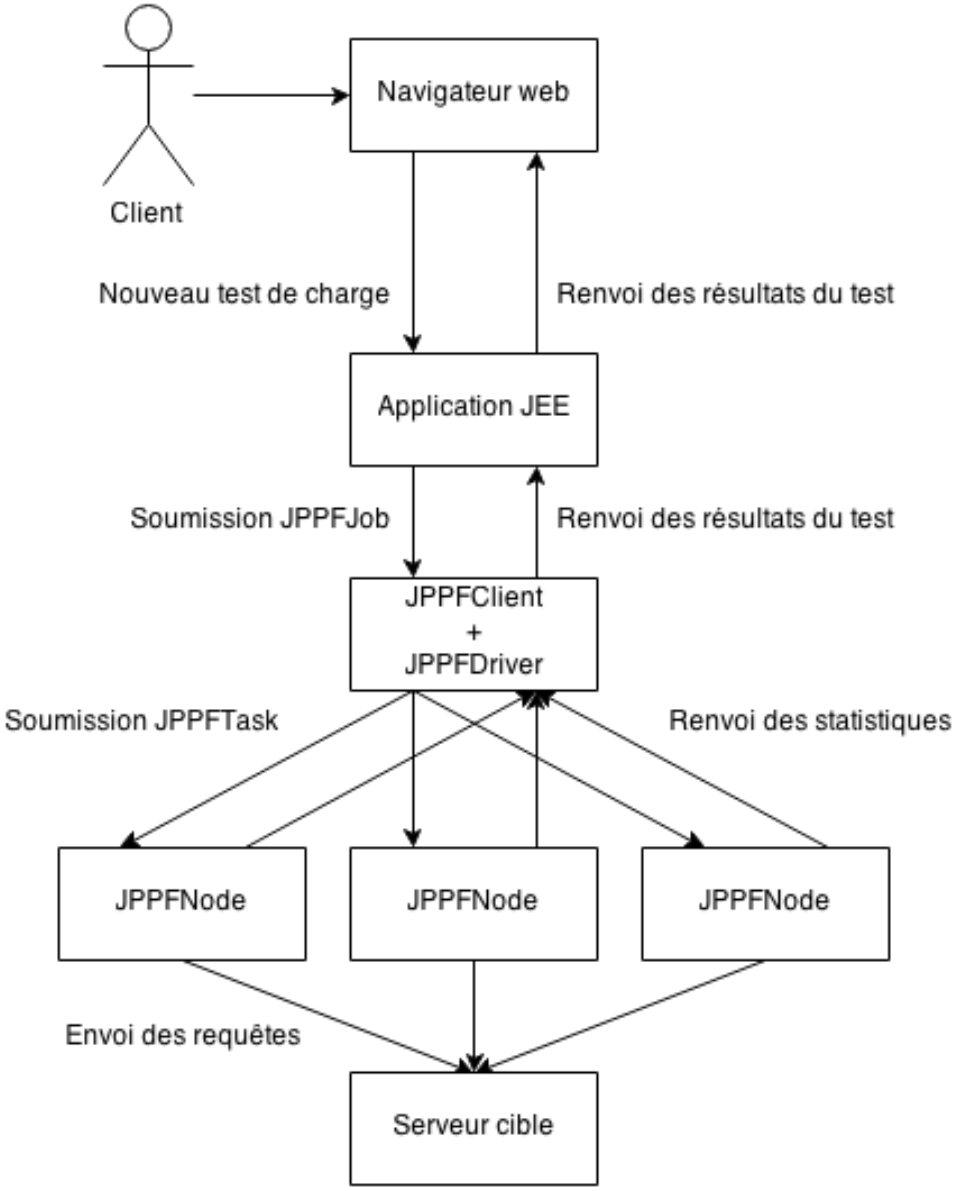
D'autres librairies ont été incluses au projet (commons-io, commons-lang...), mais leur importances et utilisations étant limitées, nous ne nous attarderons pas dessus. Néanmoins, nous nous sommes assurés qu'elles respectaient bien la licence Apache 2.0.

Architecture technique

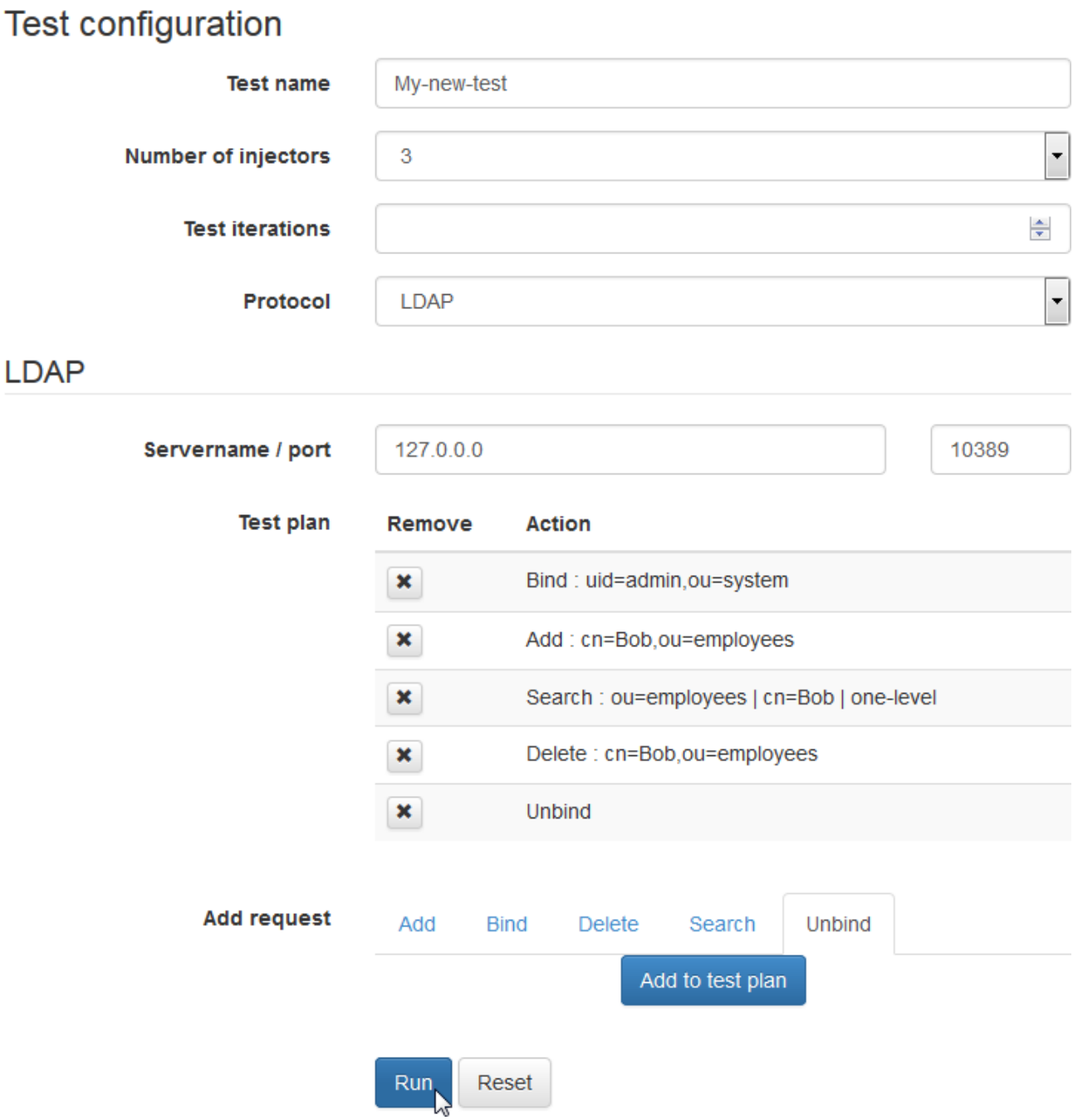


Voici un schéma représentant l'architecture physique de la solution développée et un exemple d'interaction entre les différentes entités lors de la création d'un test de charge.

Sur la page suivante, nous avons un schéma qui illustre l'architecture logicielle avec le même exemple.



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

- **Définition des besoins**
- **Établir une grille de lecture des logiciels répondant partiellement à la demande**
  - Installation et tests des logiciels
  - Déterminer les fonctionnalités intéressantes de ces logiciels
- **Conception architecturale**
  - Architecture basée sur JMeter
  - Architecture basée sur JPPF
- **Mise en place de l'environnement de test**
  - Installation et configuration de Apache Directory Server (serveur LDAP)
  - Configuration des agents (Raspberry Pi)
    - Installation et configuration de l'OS Raspbian
    - Installation de Java 8
- **Mise en place des prototypes**
  - Déploiement et test du prototype utilisant le logiciel JMeter
  - Déploiement et tests d'une application simple basée sur le framework JPPF
- **Développement des fonctionnalités du produit**
  - **Réalisation du lot 1 : Injection des requêtes depuis les Raspberry Pi**
    - Développement de la fonctionnalité d'injection des requêtes LDAP
    - Développement d'un mécanisme permettant la définition des scénarios de requêtes
    - Mise en place d'une politique de déploiement des tâches d'injection sur les différents agents
    - Mise en place d'un système de récupération et d'agrégation des temps d'exécution des requêtes
  - **Réalisation du lot 2 : Interface graphique de contrôle des injections et d'affichage des résultats**
    - Réalisation d'une maquette de l'interface graphique
    - Développement de l'interface graphique
    - Intégration du lot 2
- **Rédaction d'un manuel utilisateur pour lancer les tests et accéder aux résultats**
- **Rédaction d'un manuel développeur pour ajouter des protocoles au logiciel livré**
- **Rédaction d'un manuel d'exploitation pour analyser les performances du serveur testé**
- **Rendu du kit de livraison**