SORBONNE UNIVERSITÉ

CRÉATEURS DE FUTURS DEPUIS 1257

Année universitaire 2017 - 2018

Rapport de PSTL: “Collecte et Analyse de Données Scientifiques”

Présenté par:

HA-VINH Vincent, *3305492, M1 STL*

TOURE Adama, *3605889, M1 SAR*

Encadré par:

THIERRY-MIEG Yann, *Maître de Conférences*

Préface

Nous adressons nos remerciements les plus sincères à monsieur le professeur Yann Thierry-Mieg qui, par son suivi et ses conseils bienveillants, a permis la réalisation de ce projet.

Sommaire

Introduction

Contexte et Problématique

Un travail scientifique de qualité doit pouvoir fournir des expériences reproductibles. Les données produites seront collectées et des représentations visuelles pertinentes sont nécessaires pour qu'un humain puisse les analyser.

Ce projet PSTL a pour but le suivi qualitatif d'outils scientifiques à objectif de publication et de conférence. Pour cela, on envisage le développement d'une plateforme de collecte et de visualisation de données hétérogènes, issues de mesures de performances d'outils de Model Checking.

Le résultat recherché est une page web interactive proposant des critères de sélection et de filtrage des données pour ensuite produire des graphiques intelligibles (diagramme de dispersion, diagramme d'évolution au fil des versions…) .

De plus, dans un souci de reproductibilité des expériences, les technologies employées ne doivent pas être contraignantes en termes de coût.

Analyse

Enoncé du projet

Objectifs :

“L’objectif du stage est de réaliser :

1. Des scripts permettant d’automatiser la collecte des résultats. On dispose déjà d’outils qui fournissent à partir des traces d’exécution des fichiers CSV qui contiennent l’information pertinente. Les scripts à développer (bash à priori) utiliseront l’API Google Sheets (REST/JSON) pour créer des documents Google visibles publiquement en lecture.
2. Une page web écrite en JavaScript qui s’appuie sur l’API Google Charts pour permettre à l’utilisateur de saisir les critères pertinents et de voir le résultat. L’API Google Query (Select … Where …) permet d’extraire les données pertinentes des Sheets créées dans la première partie, afin de proposer les sélections pertinentes dynamiquement (listes déroulantes) et les plots à construire.
3. Mise en ligne du résultat : intégration des scripts de collecte dans les scripts existants d’intégration continue de la plateforme de model-checking ITS-Tools, déploiement de la page web (via GitHub Pages).”

Pré-requis et apports du projet :

“Le principal langage d’implémentation sera JavaScript, en fort appui sur plusieurs API et web services offerts par Google. Ce scénario d’orchestration de web services sera l’occasion de se familiariser avec la façon d’utiliser ces API depuis un programme, et de développer vos compétences Web/JS.
L’outil sera développé en utilisant des outils classiques du développement collaboratif : gestion de versions (github), serveur d’intégration continue (travis-ci), conventions de codage et métriques de qualité de code (Codacy), déploiement de pages statiques (Jekyll), etc…
L’ensemble des contributions du stage sera réalisé sous la forme de code FOSS, et constitué sur github, offrant de la visibilité à ce travail. La transparence du processus de collecte et d’analyse des résultats fait partie des objectifs du stage. On souhaite également qu’il soit relativement extensible (données légèrement différentes, nouvelles visualisations…)
Les contributions si elles sont de qualité seront intégrées dans la plateforme de model-checking distribuée par le lip6 : ITS-Tools.”

extraits de l’énoncé du projet fourni sur
https://www-master.ufr-info-p6.jussieu.fr:8083/site-annuel-courant/pstl

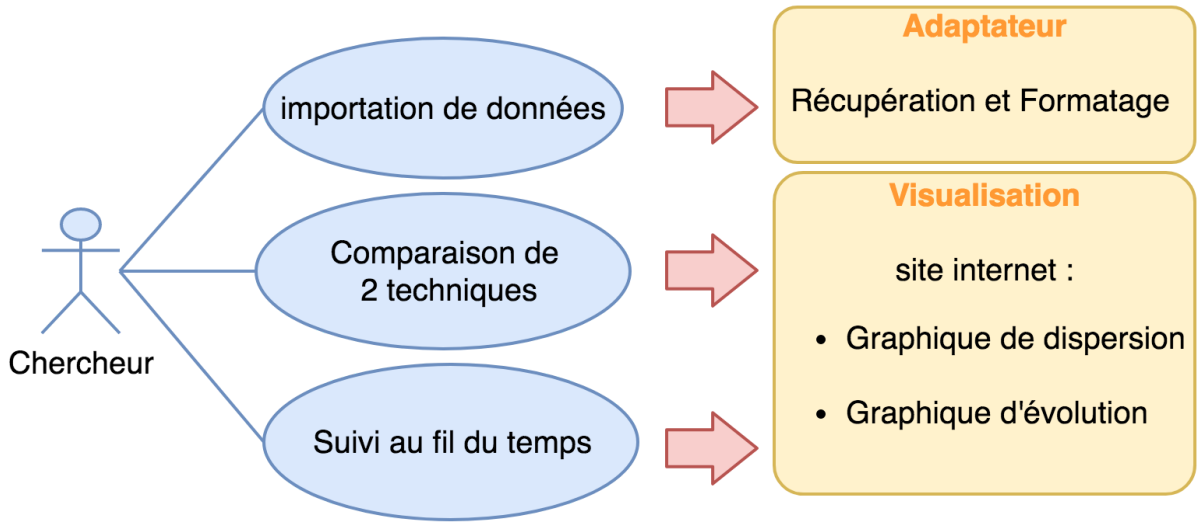
Cas d’utilisation

Après analyse des besoins renseignés dans le cahier des charges, notre étude identifie 3 cas d’utilisation principaux.

Le premier, introduire les données dans le système.
Le but est de monter les données expérimentales sous un format exploitable.

Le second, comparaison de deux techniques.
Le chercheur a des expériences, et essaie des compromis entre tel et tel algorithme, avec des complexités différentes. Il veut pouvoir comparer ses algorithmes pour savoir lequel est le meilleur sur un jeu d’exemples important.
Pour cela, on désire afficher les données sous la forme d’un diagramme de dispersion

Le troisième, suivi au fil du temps.
Cette fonctionnalité sert à pouvoir suivre la progression des performances au fur et à mesure des versions. Après une mise à jour, l’outil du chercheur est-il plus performant ou moins performant ?
Pour cela, on désire afficher les données sous la forme d’un diagramme d’évolution.



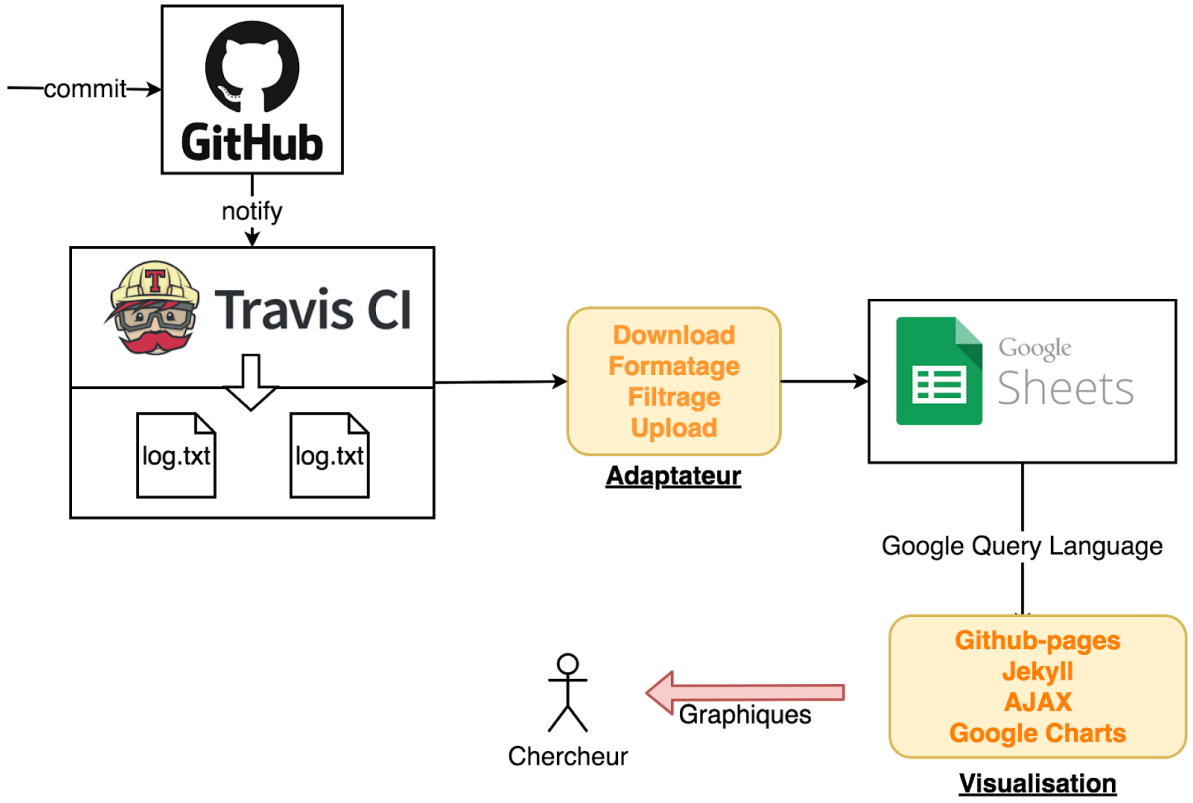
Architecture retenue

En s'appuyant sur les cas d'utilisation précédemment trouvés, on peut diviser notre solution en deux parties distinctes:

La première partie, qu'on nommera Adaptateur, comprendra le cas d'utilisation "importation des données". Cette partie implémentera les phases de :

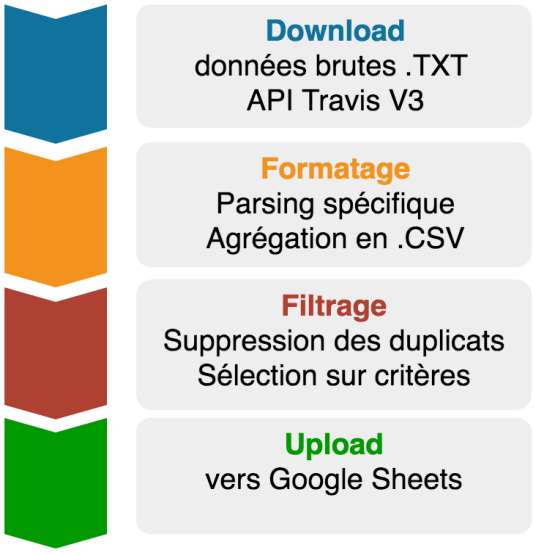
- téléchargement des données depuis leur source, les serveurs de Travis-CI, service d'intégration continue, par l'intermédiaire de l'API Travis-CI V3.
- formatage des données pour ne garder que les informations métier utiles, de filtrage et élimination des doublons selon certains critères définis par le client.
- exportation des données formatées vers le service Google Sheets.

La seconde partie, appelée visualisation, comprendra les cas d'utilisation "comparaison de 2 techniques" et "suivi au fil du temps".
Cette partie prendra la forme d'un site web statique, dont le contenu sera rendu dynamique avec des scripts javascript qui iront questionner la Google Spreadsheet créée à l'étape précédente pour enrichir les choix du formulaire de la page Web statique. Enfin, à la validation du formulaire, les diagrammes seront générés et insérés dans la page HTML du client par javascript grâce à la librairie Google Charts.



Conception

Adaptateur



Download

Chaque fois que le chercheur produit une nouvelle version de son outil de Model Checking, il “commit” (envoie les changements de) cette version sur un dépôt Github. Si le chercheur a configuré son compte Travis-CI pour que celui-ci suive ce dépôt Github, alors Travis-CI est notifié du nouveau commit et commence sa routine d’intégration continue (lancement d’une machine virtuelle qui va effectuer les tests définis par le chercheur et produire des logs).

> “As a continuous integration platform, Travis CI supports your development process by automatically building and testing code changes, providing immediate feedback on the success of the change. Travis CI can also automate other parts of your development process by managing deployments and notifications.”
>
> https://docs.travis-ci.com/user/for-beginners/

Les données du chercheur sont donc les logs produits et hébergés par Travis-CI. Pour les télécharger, on communique avec l’API Travis-CI V3 par requête GET HTTP en indiquant

l'adresse de la ressource désirée. Nous utiliserons la commande curl pour envoyer ces requêtes.
Il faut dans un premier temps récupérer tous les identifiants uniques des logs produits, pour pouvoir ensuite récupérer un à un chaque log disponible à son adresse respective, construite avec son identifiant.

```
curl
"https://api.travis-ci.org/v3/repo/yanntm%2FITS-Tools-pnmcc/builds?sort_by=id&limit=$length&o
ffset=$skipped"
```

L'API répond dans un format JSON. Pour extraire les identifiants de cette réponse, nous avons décidé d'utiliser le parser jq, qui est disponible dans les dépôts officiels de Debian et Ubuntu. Jq est un parser de format JSON, et nous permet de parcourir la réponse de l'API pour récupérer les identifiants avec la commande suivante :

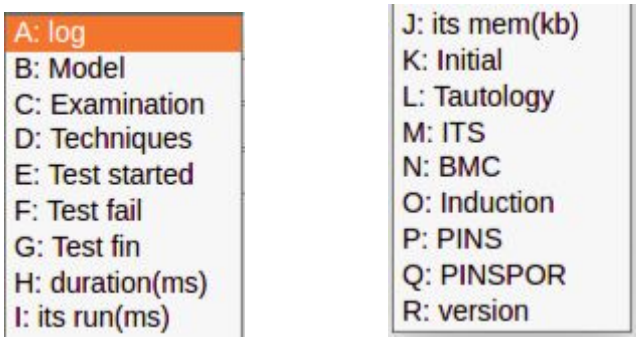
```
jq -r --arg i $i '. | .builds[$i|tonumber].number'
```

Enfin, on envoie une requête GET HTTP sur l'adresse unique de chaque log pour récupérer un fichier .txt par log.

Formatage

Pour le formatage des fichiers .txt, nous utilisons un script écrit en langage perl qui lit chaque fichier log_i.txt et, à l'aide d'expressions régulières, en extrait les informations pertinentes. Ce script traduit donc un fichier .txt entier en quelques lignes de données utiles. Nous allons donc agréger l'ensemble des lignes de données produites en un seul fichier au format .csv, format supporté par Google Sheets (voir étape d'Upload).

Les colonnes du .csv sont les suivantes (peut varier selon la version du script perl utilisé) :



Filtrage

Précédemment, nous avons supposé que pour un commit de l'outil du chercheur, Travis-CI lançait une machine virtuelle qui exécutait une seule fois la batterie de tests.
En fait, on ne peut confirmer totalement cette hypothèse, car il est possible qu'une seule mise à jour de l'outil du chercheur déclenche plusieurs fois l'exécution de la batterie de tests, ou bien que le chercheur lui-même ait besoin d'exécuter la batterie de tests ponctuellement.

Il en résulte l'apparition de multiples lignes-résultats pour un seul et même test (avec les mêmes paramètres mais des mesures de performances différentes. Cette part d'aléatoire non reproductible est due au contrôle limité que le chercheur a sur l'environnement d'exécution des machines virtuelles du côté des serveurs Travis-CI).
Or, à cause de la nature de nos diagrammes de visualisation, nous ne pouvons accepter des données redondantes.
Par exemple, comment afficher sur un diagramme de dispersion la comparaison de 2 techniques pour une expérience, si une des deux techniques possède 5 résultats différents pour cette expérience ?
De plus, la librairie Google Charts utilisée pour créer nos diagrammes ne garantit un résultat cohérent qu'en cas de non redondance des données pour une des deux techniques comparées:

"Note that *dt2* cannot have duplicate keys, but *dt1* can."

description de join(),
https://developers.google.com/chart/interactive/docs/reference

De ce fait, nous avons besoin de filtrer nos données expérimentales pour supprimer les lignes en doublons.
Note : nous utilisons le terme doublons pour parler de lignes qui possèdent la (les) même(s) valeur(s) sur certaine(s) colonne(s) du .csv . Ces lignes ne sont pas des copies exactes entre elles.

Notre approche du problème est d'utiliser la commande sort, une des commandes standard GNU et très bien optimisée, pour trier le .csv.
En indiquant certains champs (option -k), on fait apparaître la ligne à conserver avant les autres doublons, puis dans un second appel à sort (option -u), on ne garde que la première apparition d'une ligne en cas de doublons.

L'algorithme de tri en entier est le suivant :

Le 1er sort trie
- en premier les colonnes clefs [model,examination,techniques,version]
- puis à valeurs égales dans ces colonnes, trie par ordre croissant les Test Fail
- puis à valeurs égales pour Test Fail, trie par ordre décroissant les Test Passed
- puis à valeurs égales pour Test passed, trie par ordre décroissant la duration.

Le 2e sort trie les colonnes clefs [model,examination,techniques,version]
avec l'option -u, à valeurs égales dans ces colonnes clefs, sort ne garde que la première ligne rencontrée (qui est la ligne filtrée selon le premier sort (fail min, passed max, duration max)).

Note : la nécessité d'un 2e sort est due au fait que l'option -u s'applique avant que le sort ne trie les données, donc avant que la ligne à conserver soit remontée en première position des lignes doublons.

Upload

Pour cette étape, notre objectif était d'implémenter à la suite du filtrage dans le script bash des instructions d'Update pour concaténer les lignes de données parsées et filtrées à une Google Sheet disponible en lecture publique sur le compte Google dédié au projet.

Cette étape pouvant être effectuée manuellement, sa priorité n'était pas haute dans notre planning des tâches à faire. Nous n'avons malheureusement pas pu y revenir avant la soumission du projet.

Néanmoins, le procédé à implémenter serait le suivant :

- activer l'API Google Sheets depuis le gestionnaire des APIs Google (Google API Console).
- Créer un jeton Oauth2 d'authentification de notre projet auprès de l'API du compte Google dédié.
- avec la commande curl, envoyer une requête POST HTTP à l'API Google Sheets, à l'adresse de la Google Sheet à remplir, en passant dans le corps du message les valeurs des lignes à envoyer et quelques paramètres encapsuler dans un message JSON.
  (cf. https://developers.google.com/sheets/api/samples/writing#append_values)

- Probablement, gérer le cas où la Google Sheet est pleine (Actuellement, les Google Sheets ont une limite de 2 millions de cellules, soit environ 100 000 lignes de logs dans notre cas).

Visualisation

Nous passons maintenant à la seconde partie du projet, la visualisation des données qui ont été précédemment chargées sur Google Sheets.

Site statique

Pour construire notre plateforme web de visualisation, nous allons nous servir d'un service web proposé par Github, à savoir Github-Pages.
Ce service peut s'activer dans les options de chaque dépôt Github et permet de générer un site web statique à partir des fichiers présents dans le dépôt. Le site web est mis à jour à chaque nouveau commit sur le dépôt.

Github-Pages utilise le générateur de site statique Jekyll, qui permet de s'abstraire de certains aspects front-end des sites web.
Dans notre cas, nous avons produit notre page web et son aspect graphique en écrivant un fichier au format markdown, contenant nos formulaires HTML. Ce fichier index.md a ensuite été traduit par Jekyll en fichier index.html bien formé et le rendu graphique a été généré à partir de thèmes et de règles CSS préétablis.

Il a cependant été nécessaire de beaucoup se documenter sur les bonnes méthodes d'utilisation de Jekyll et de ses fichiers de configuration. Notamment, pour tester les fonctions et interfaces du site web sans devoir commit sur notre dépôt, il faut pouvoir lancer un serveur Jekyll sur sa machine locale. Il faut donc installer Jekyll, qui est écrit en Ruby et qui nécessite tout l'environnement d'exécution d'un outil Ruby, qui comprend de nombreuses dépendances.

Aspect dynamique

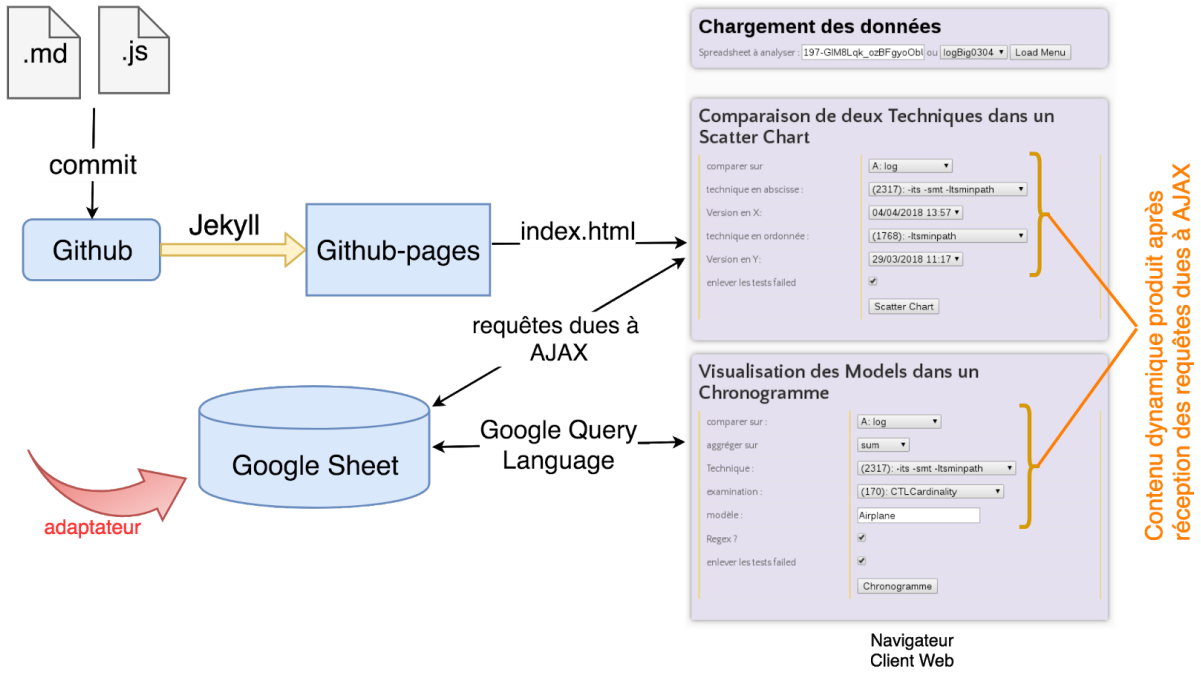
A ce stade, nous avons un site web statique (même index.html envoyé à tous les clients) et une Google Sheet qui est notre source de données.
Rappelons que Github-Pages, bien que pratique, est un service qui crée un site web n'ayant pas de côté serveur pouvant exécuter des scripts. Le site se contente d'envoyer les pages html qu'il héberge à un client lui demandant par requête Http. Ceci est communément appelé le web 1.0.

Il nous faut maintenant rendre le contenu des formulaires de notre page web dynamique.
On va utiliser le principe AJAX, *Asynchronous Javascript and XML*, pour modifier directement le contenu de nos formulaires depuis le navigateur du client.

Ainsi, au chargement de la page web dans le navigateur client, des fonctions javascript vont se lancer. Elles envoient des requêtes asynchrones sur la Google Sheet et à la réception d'une réponse, vont afficher les résultats dans les formulaires.
Cette étape correspond au terme XML du sigle AJAX pour des raisons historiques. Aujourd'hui, le langage XML est souvent remplacé par JSON.
Mais de toute façon, nous n'avons pas à gérer cette compatibilité des langages de communication entre le client et le serveur. Nous utiliserons ici la librairie javascript Google Charts qui est une librairie de code permettant de tracer des diagrammes à partir de données existantes. Cette librairie permet entre autres d'envoyer des requêtes vers une Google Sheet et de recevoir les données résultats à l'aide des classes Query et QueryResponse. Cela permet de faire abstraction du protocole de communication entre le navigateur client et le serveur qui serait ici le cloud Google hébergeant notre Google Sheet.

Au chargement de la page, nous allons questionner notre Google Sheet pour récupérer les intitulés des colonnes (1e ligne de la Google Sheet), les noms des techniques (cellules de la colonne "Techniques"), les dates des versions (cellules de la colonne "version"). et à la réception des réponses, le script javascript enrichira la page web du client avec les données reçues.

La page du navigateur client possède donc maintenant des formulaires entièrement personnalisés en fonction de la Google Sheet source.



Google Query Language

Une fois la page web et ses formulaires chargés, le chercheur renseigne les critères qu'il désire et lance la génération des diagrammes.
De la même manière que précédemment, le code javascript va envoyer à la Google Sheet des requêtes correspondant aux critères choisis.

A noter, le langage de requête supporté par la librairie Google Charts est le Google Query Language, proche du SQL mais avec des différences qui ont un impact important.
Par exemple :

- Il n'y a pas de clause FROM, et pas de possibilité de nommer des variables pendant la requête.
- Il n'y a pas de terme distinct.

Le Google Query Language ne laisse en fait pas la possibilité de créer des requêtes nécessitant plus d'un parcours de l'ensemble des lignes de la Google Sheet. Il est probable que la raison derrière cette politique d'implémentation soit d'éviter la surcharge de calcul des serveurs Google alloués au service de requête sur Google Sheet.

Dans le cas du diagramme de dispersion, nous devons donc envoyer deux requêtes différentes pour récupérer les lignes correspondant à une technique et une version choisie, pour chacune des deux techniques. Nous les joindrons ensuite sur les colonnes "Model" et "Examination" avec la fonction Google Charts join() qui elle s'exécutera sur le navigateur client.

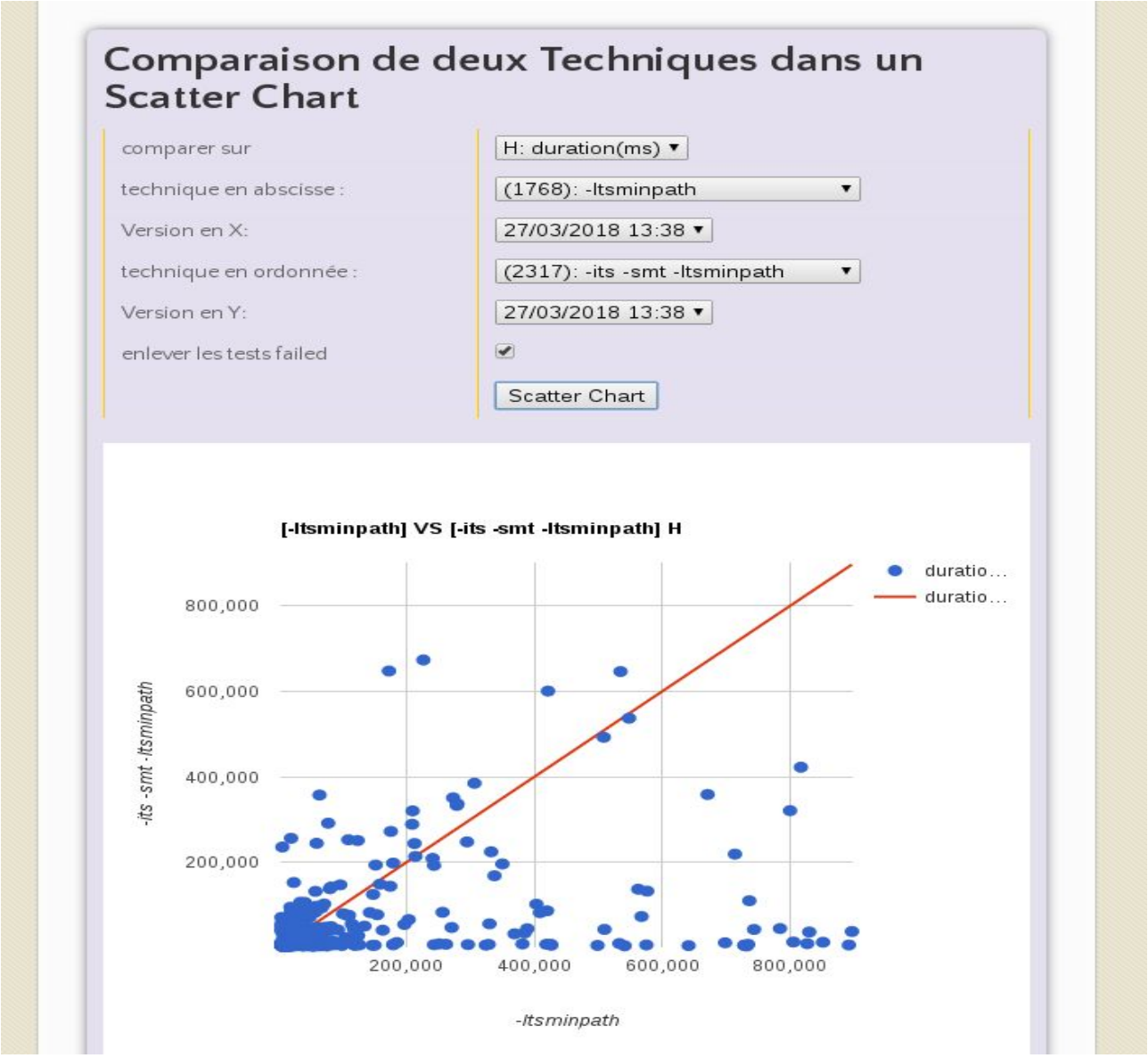
Google Charts

Enfin, à la réception des données en réponse aux requêtes précédentes, le code javascript va faire appel aux fonctions de création de diagrammes de la librairie Google Charts.
Nous tenons à préciser que cette étape, peu détaillée dans ce rapport, a pris une importante partie du temps de développement en raison d'un besoin de documentation très important concernant les nombreuses fonctionnalités et subtilités de la librairie Google Charts.

Résultats

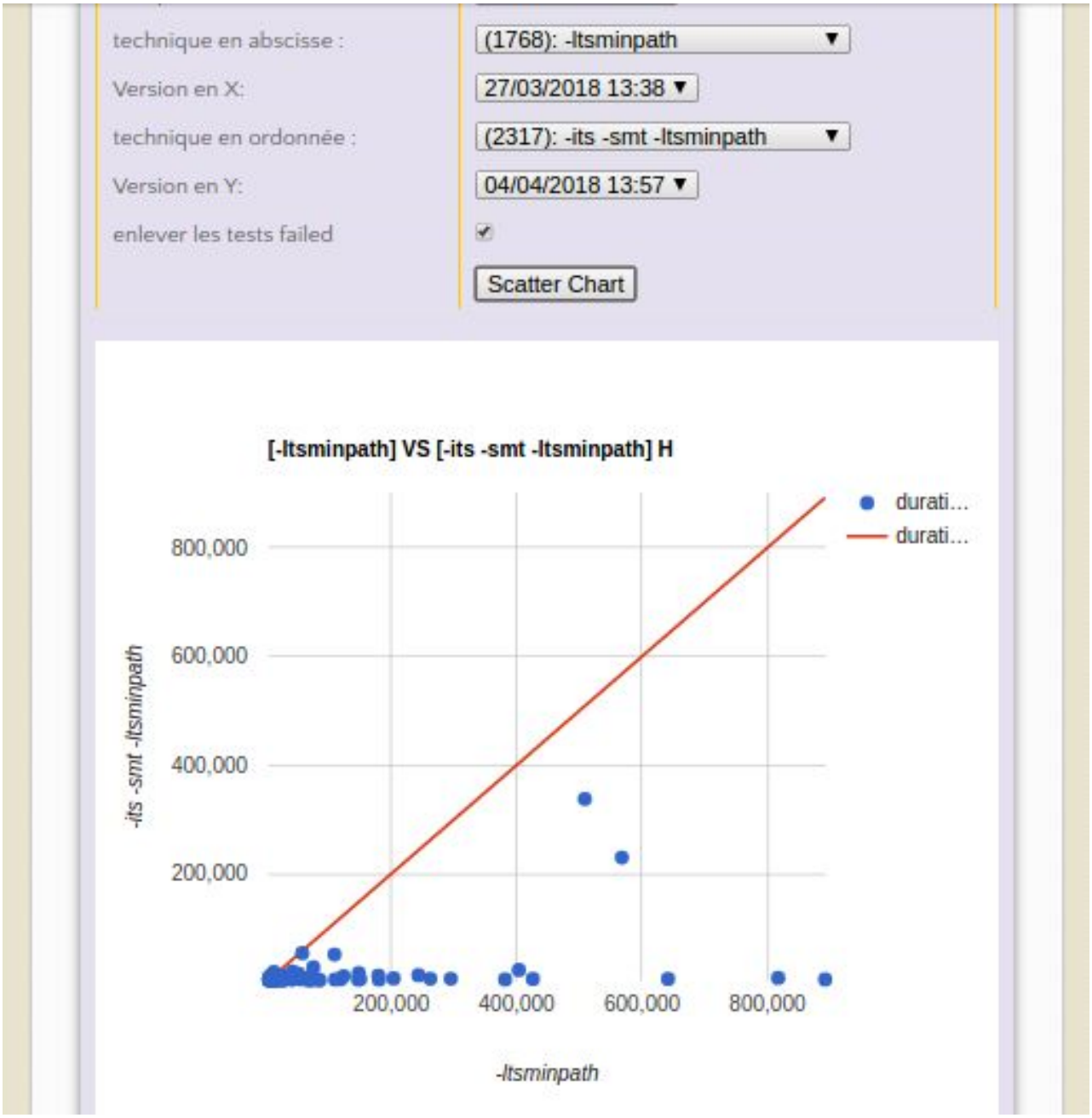
A l'heure actuelle, notre plateforme est fonctionnelle.
Le chercheur peut s'y rendre et sélectionner les critères voulus dans les formulaires.

Un exemple du diagramme de dispersion :



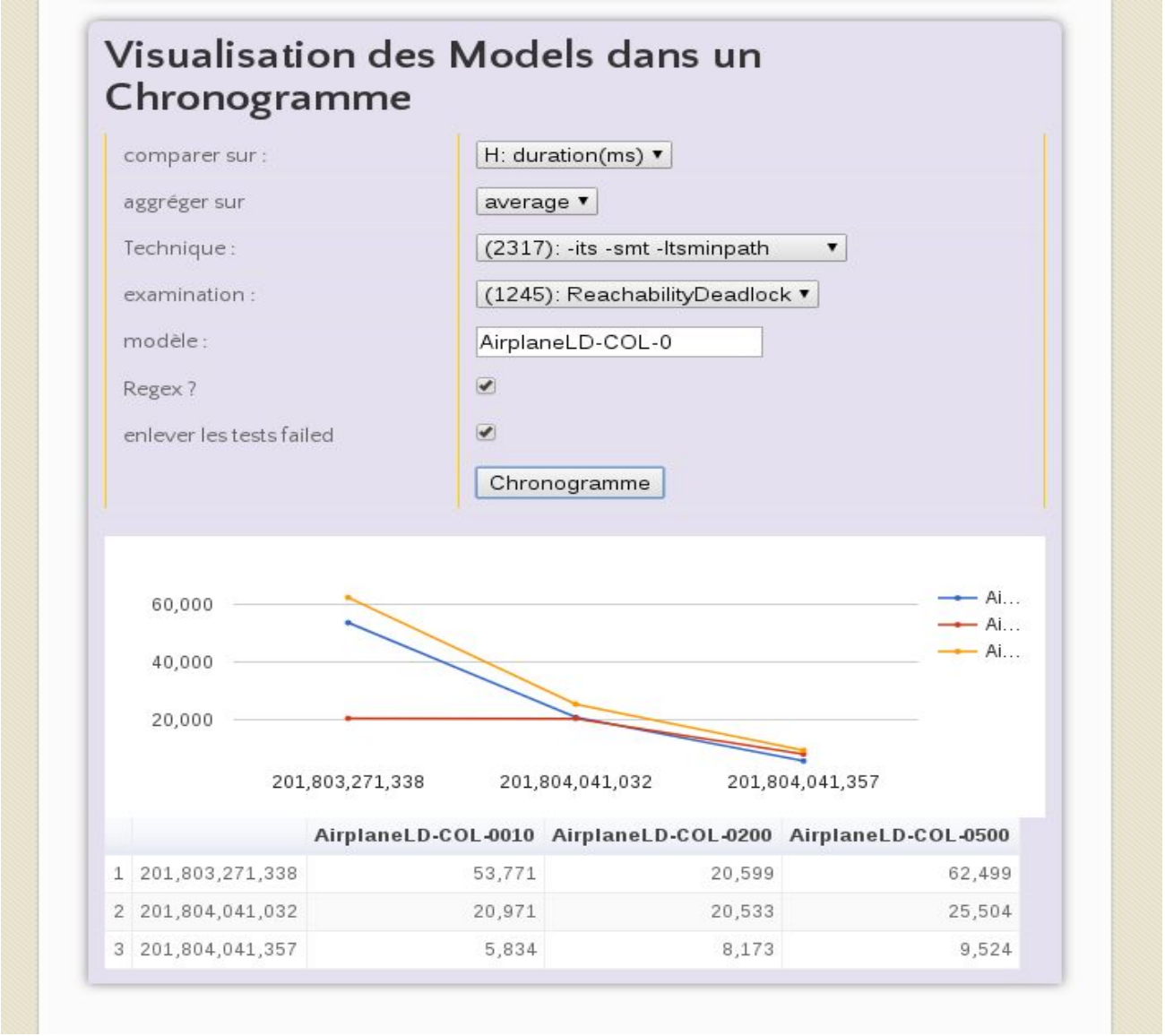
Chaque point représente une expérience. Ici on constate qu'il y a globalement moins de points du côté de -its -smt -ltsminpath, laissant supposer une meilleure performance générale. Mais le résultat reste ambigu.

Un second exemple, cette fois avec une version plus récente de -its -smt -ltsminpath :



Ici, on peut constater que l'ensemble des points est en-dessous de la diagonale, du côté de -ltsminpath.
Donc cette version de -its -smt -ltsminpath bat à plates coutures l'ancienne version de -ltsminpath en termes de performances.

Un exemple de diagramme d’évolution :



		AirplaneLD-COL-0010	AirplaneLD-COL-0200	AirplaneLD-COL-0500
1	201,803,271,338	53,771	20,599	62,499
2	201,804,041,032	20,971	20,533	25,504
3	201,804,041,357	5,834	8,173	9,524

On constate une nette amélioration d’exécution au fil des versions.
L’axe des abscisses représente les dates des versions.
En bas on a le résultat du tableau.

Conclusion

Nous pouvons dire que ce projet qui mixe de nombreuses technologies n'est pas une application traditionnelle mais une application conçue comme un assemblage de web services avec lesquels on interagit en perl, en javascript, en shell au moyen de nombreux outils, ce qui nécessite beaucoup de documentation et d'apprentissage.

Il faut souligner que ce projet, mené à bien, est en ligne à l'adresse https://vincenthavinh.github.io/PSTL_1718/ .
Si l'on peut encore améliorer son aspect, ce projet joue son rôle et remplit les 3 cas d'utilisation qu'on avait cernés.

Personnellement, en m'obligeant à un travail de documentation, ce projet m'a permis de découvrir des techniques orientées vers l'entreprise, ce qui m'a beaucoup intéressé. Il m'a également beaucoup plu par son originalité qui le rendait différent de projets plus traditionnels ou plus académiques.

Bibliographie

http://ddd.lip6.fr/
https://travis-ci.org/yanntm/ITS-Tools-pnmcc/
https://developers.google.com/sheets/api/reference/rest/
https://developers.google.com/chart/interactive/docs/gallery/
https://developers.google.com/chart/interactive/docs/querylanguage/