Analyse et tri de stacktraces pour la complétion du buckets

Antonin DUREY, Jihad EL FAKAWI

Novembre 2016

Table des matières

Introduction

De nos jours, les crashs de logiciels ou de programmes sont quotidiens et les grosses applications doivent faire face à des milliers de rapport de crashs. L'envoi de ces rapports aux développeurs est néanmoins essentiel pour permettre à ceux-ci de corriger les problèmes et failles. Pour simplifier l'analyse de ces rapports, ils sont triés par bucket. Un bucket rassemble ainsi une série de rapports dont la source du problème est très certainement la même. Grâce à cela, lorsque les développeurs analyserons les rapports contenus dans chacun des bucket, ils disposeront d'une grande quantité d'information pour corriger le problème.

Dans le cadre du module OPL de notre Master 2 IAGL, nous avons été amenés à réaliser un projet qui porte sur l'analyse de crash. Sur ce projet, nous avons implémenté le tri de rapports d'erreurs dans des buckets.

Pour trier ces rapports, nous avons implémenté plusieurs algorithmes. Dans un premier temps, nous les avons triés de manière aléatoire. Dans un second temps, nous avons attribué un certain nombre de points aux buckets en fonction des points communs entre le rapport à trier et les rapports contenus dans le bucket. Le bucket choisi est alors celui dont le score est le plus élevé. Le troisième algorithme est une variante du second, il calcule la moyenne des points par nombre de lignes d'erreurs et par nombre de rapports dans un bucket.

L'évaluation de nos résultats a été facilité par l'infrastructure mise en place. En effet, nous pouvions soumettre nos résultat, c'est-à-dire l'attribution des rapports d'erreurs à un bucket précis. Grâce à cela, nous avons constaté que notre troisième et dernier algorithme est plus performant que le second, lui même plus performant que le premier. Néanmoins, ces 2 dernières solutions présentent l'inconvénient de prendre un temps conséquent lors du tri - plusieurs dizaines de secondes avec les données fournies.

Ce rapport présente ces travaux en 3 parties. La première est consacré au travail technique qui a été effectué. La seconde se concentre sur l'évaluation des résultats et des algorithmes. Enfin, la troisième et dernière partie traite des améliorations et des évolutions possibles du projet.

1 Travail technique

1.1 But

1.2 Overview

1.3 Algorithmes

Nous avons implémenté 4 algorithmes différents permettant d'analyser des stacktraces et de les incorporer à des buckets déjà existants.

Le premier est un algorithme aléatoire qui à chaque stacktrace à trier, associe un numéro de bucket alétaoire parmis ceux existants.

Algorithm 1: Algorithme d'attribution de buckets aléatoire

Data: les buckets existants nommés $buckets$ et la liste des stacktraces à trier nommée $stacktracesToBeAttributed$
Result: une liste avec pour chaque stacktrace à trier le numéro du bucket auquel la stacktrace est associée
initialization
for $stacktrace$ *in* $stacktracesToBeAttributed$ **do**
  `addToResults(`$stacktrace$`, getRandomBucketId(`$buckets$`))`
end

Le second algorithme parse les buckets et stacktrace existants. C'est à dire qu'il stocke les informations ligne par ligne pour chaque stacktrace. Cela permet d'obtenir la librairie ou le fichier, la méthode et la ligne ou l'erreur s'est produite.

Dans un second temps, il analyse, pour chaque stacktrace à trier, toutes ses lignes ainsi que toutes les lignes des stacktraces appartenant aux buckets, et attribue des points en fonction de la similitude entre les lignes. Ainsi, l'algorithme accumule des points de similitude par bucket. Pour chaque stacktrace, le bucket choisit est celui qui a le plus de points. C'est cette seconde partie qui est présenté ci-dessous :

Algorithm 2: Algorithme d'attribution de buckets basé sur le calcul de points par rapport à la similitude des lignes des stacktraces

```
Data: les buckets existants nommés buckets et la liste des stacktraces à trier nommée
      stacktracesToBeAttributed
Result: une liste avec pour chaque stacktrace à trier le numéro du bucket auquel la
        stacktrace est associée
initialization
for stacktraceToBeAttributed in stacktracesToBeAttributed do
    initialize(bucketsWithValue)
    for bucket in buckets do
        points ← 0
        for stacktraceFromBucket in bucket do
            for lineFromToBeAttributed in stacktraceToBeAttributed do
                for lineFromBucket in stacktraceFromBucket do
                    points ← points + getPointsFromLibSimilitude(lineFromBucket,
                      lineFromToBeAttributed)
                    points ← points + getPointsFromFileSimilitude(lineFromBucket,
                      lineFromToBeAttributed)
                    points ← points +
                      getPointsFromMethodSimilitude(lineFromBucket,
                      lineFromToBeAttributed)
                    points ← points + getPointsFromLineSimilitude(lineFromBucket,
                      lineFromToBeAttributed)
                end
            end
        end
        putIntoMap(bucketsWithValue, bucket, points)
    end
    chosenBucket ← getBucketWithHighestValue(bucketsWithValue)
    addToResults(map, chosenBucket)
end
```

Le troisième algorithme est une variante du second. Au lieu de calculer la somme des points, on calcule la moyenne des points en fonctionne du nombre de lignes de la stacktrace et du nombre de stacktraces dans un bucket. Cela donne donc un nombre de points moyen par ligne et par bucket. La fin de l'agorithme est la même : le bucket avec le plus de points est désigné.

Le quatrième et dernier algorithme et très différent. Il suppose davantage de stockage d'éléments. En effet, lors de la création des stacktraces, l'algorithme va stocker les librairies, fichiers et méthodes dans une structure sous forme de maps imbriquées. Ainsi, lors de l'attribution d'une liste, il n'y a plus besoin de parcourir chacune des lignes, mais seulement de regarder cette structure pour voir quels

sont les éléments en commun.

Algorithm 3: Algorithme d'attribution de buckets basé sur le calcul de points par rapport au nombre de librairies, fichiers et fonctions en commun

```
Data: les buckets existants nommés buckets et la liste des stacktraces à trier nommée
      stacktracesToBeAttributed
Result: une liste avec pour chaque stacktrace à trier le numéro du bucket auquel la
        stacktrace est associée
initialization
for stacktraceToBeAttributed in stacktracesToBeAttributed do
    initialize(bucketsWithValue)
    for bucket in buckets do
        points ← 0
        for stacktraceFromBucket in bucket do
            /* La comparaison des stacktraces s'effectue maintenant grâce à la structure
            contenant les librairies, fichiers et fonctions présentes dans chaque
            stacktrace. Le parse de ces données a été effectué précédemment lors de la
            création de la stacktrace */
            for lib in getLibsUsed(stacktracesToBeAttributed) do
                if stacktraceUsedLib(stacktraceFromBucket, lib) then
                    points ← points + 1
                    for file in getFilesUsed(stacktracesToBeAttributed, lib) do
                        if stacktraceUsedFileFromLib(stacktraceFromBucket, lib, file)
                         then
                            points ← points + 5
                            for function in
                             getFunctionsUsed(stacktracesToBeAttributed, lib, file) do
                                if
                                 stacktraceUsedFunctionFromLibAndFile(stacktraceFromBucket,
                                 lib, file, function) then
                                    points ← points + 25
                                end
                            end
                        end
                    end
                end
            end
        end
        putIntoMap(bucketsWithValue, bucket, points)
    end
    chosenBucket ← getBucketWithHighestValue(bucketsWithValue)
    addToResults(map, chosenBucket)
end
```