Compte Rendu de Projet de recherche de L3

N°3 : Mise au point & Changements

Florian REYNIER & Mathis CARISTAN

16/02/2016

Résumé de la réunion

Au cours de la réunion, plusieurs points ont été abordés. Dans un premier temps, nous avons discuté d'un paramètre de notre fonction de hachage. Puis nous avons étudié les résultats préliminaires que nous avions obtenu. Nous avons evoqué les changement entrepris dans notre fonction xxHash. Enfin, nous avons soulevé un problème que nous avons rencontré, et de la manière de le résoudre.

1 Le paramètre de notre fonction de hachage.

La taille des sous-chaînes s_i dans notre fonction a été le premier point de discussion. Étant donné que nous avons travaillé avec une table de taille 2^{16}, il était nécessaire de prendre des sous-chaînes d'au moins 16 bits de long. Dans le cas contraire, le nombre de sous-chaînes possibles aurait été inférieur à la taille de la table, impliquant que des valeurs de la table n'aurait jamais été associées à des clefs. En revanche, il était moins évident de justifier pourquoi la taille des sous-chaînes n'était pas supérieure à celle de la taille. Sur ce point, il convient d'étudier un instant le comportement de la fonction pour des mots courts. Prenons par exemple les mots "`ABC`", "`BCA`", "`DEFG`" et "`FGDE`". Nous remarquons immédiatement que les deux premiers sont une permutation l'un de l'autre, de même que pour les deux derniers. Chaque lettre étant codée sur 5bits, les deux premiers correpondent à une chaîne de 15bits, tandis que les deux autres une chaîne de 20bits. Pour la suite de la fonction, chaque mot est découpé en chaînes de 16bits de long (complétées par des bits à 0 si besoin). Dès lors, il apparaît que `ABC` et `BCA` (ainsi que toutes autres les autres permutations) correpondent à la même valeur de hachage. Bien que cela ne soit pas le cas pour `DEFG` et `FGDE`, un découpage en chaînes plus longues (20bits ou plus) entraînerait le même comportement. Une solution pour éviter ce comportement (quand on testera la fonction sur 32bits par exemple) pourrait d'appliquer une permutation des séquences correspondant aux lettres selon leur place dans le mot (les 5bits de la première lettre ne sont pas permutés, ceux de la deuxième lettre sont permutés cycliquement d'un "cran" etc.)

2 Résultats préliminaires et changement de langage

Les premiers résultats de notre fonction de hachage sont excessivement mauvais. Bien que nous ne nous attendions pas à ce que cette fonction révolutionne le hachage, les résultats obtenus semblent particulièrement mauvais (sur un texte de 200 000 mots, et une table de taille 65 000, le taux d'occupation est inférieur à 10%, et un nombre de collision moyen d'environ 70, avec jusqu'à 2800 collisions pour la valeur 0). Ceci nous laisse supposer de la présence d'une erreur dans le code de la fonction de hachage. Cette supposition est d'autant plus crédible du fait de la méthode employée dans le code. Étant donnée que le type char n'existe pas en OCaml, et que le langage ne permet pas de manipuler un entier en passant par sa représentation binaire, nous avons choisi de représenter les lettres et mots par des listes de booléens.

Cette observation, ainsi que l'étude d'autres fonctions de hachage, souvent basées sur les manipulations de bits, nous a mené à la conclusion que l'OCaml n'était sans doutes pas le langage le plus adapté pour l'écriture de fonctions de hachage. En conséquence, nous avons choisi de modifier le langage du projet. Nous comptons écrire les fonctions suivantes, et ré-ecrire notre fonction de hachage, en C, et les interfacer avec de l'OCaml pour leur utilisation. Ainsi, nous continuerons à utiliser (et apprendre) l'OCaml, mais en écrivant de manière plus simple le cœur des fonctions.

3 Adaptation de la fonction xxHash

La fonction xxHash est en général utilisée pour calculer des checksum, après discussion il a donc été décidé de modifier cette fonction afin de non pas calculer un hash pour tout le fichier, mais un hash par mot dans le fichier. Cette modification a nécessité de changer du code en profondeur, en effet la fonction réalisait le hash du fichier en lisant celuis ci par bloc de 64ko, il a donc fallu changer de fonction de lecture de fichier afin de pouvoir récupérer le flux non pas de bloc en bloc fixe mais de mots en mots. Il a également été décidé de rediriger la sortie des hash vers un fichier.

4 Problème

Lorsque nous avons essayé de créer un histogramme des résultats de notre fonction de hachage, nous nous sommes retrouvés confrontés à un léger problème. Le logiciel que nous utilisons (LibreOffice Calc) ne permet pas une "belle" présentation sous forme d'histogramme d'un échantillon d'environ 65000 données. Pour résoudre le problème, nous allons essayer de refaire l'histogramme sur un autre logiciel, et dans le cas ou cela ne fonctionnerait pas non plus, nous travaillerons à la place sur un échantillon plus réduit de données (une table de 2^{10} voire moins si nécessaire), pour permettre une représentation graphique des résultats. Ces travaux sur des tables de taille réduites ne devraient cependant servir que pour illustrer les propriétés des fonctions, le réel travail d'analyse se fera toujours sur des tables de taille plus importante (2^{16} et 2^{32}).

Objectifs

Nos objectifs prioritaires sont en premier lieu de pouvoir réaliser deux histogrammes clairs et lisibles, pour cela nous avons choisi de réduire le nombre de points afin de pallier le problème du nombre de points. Par la suite il sera intéressant de commencer à implanter de nouvelles fonction et notre choix s'est porté sur sdbm, de réputation peu efficace, murmurhash un candidat serieux à xxHash, et djb2 qui présente un interêt de par sa conception.