Équipe: Nugget

March 18, 2017

1 Projet

Aujourd'hui, notre projet permet d'évaluer un tableur dont l'ensemble des formules tient en RAM, ainsi que d'effectuer des mises à jour sur celles-ci. Comme demandé dans la spécification, les formules dans la zone d'intéraction spécifiée par l'utilisateur sont évaluées en priorité par rapport au reste de la feuille de calcul. Dans ce document nous expliquons en détail les problèmes rencontrés et les solutions que nous proposons.

1.1 Approche Naïve: Dépendances de formules et Évaluation

Lorsque l'on se propose d'évaluer une feuille de calcul on se heurt rapidemment à un premier problème, certaines formules ont besoin du résultat d'autres formules pour pouvoir s'évaluer. Afin de résoudre les dépendances entre cellules nous nous sommes dirigés vers une représentation de la feuille sous forme de graphe orienté. Cela nous permettait de manipuler facilement les dépendances par construction: la relation de dépendance est représentée par par une arrête dirigée, que nous notons $D(c_1, c_2)$ pour dire que la cellule c_1 dépends de c_2 (i.e, c_2 doit être évaluée avant c_1). L'ordre des calculs est donc induit par le graphe et via une méthode par point fixe, l'évaluation consistait alors à récupérer un ensemble des cellules évaluables C vérifiant que $\forall c \in C,\ \not\exists\ c'$ tel que $D(c, c')$. Lorsqu'il n'est plus possible d'extraire un tel ensemble non vide, l'algorithme s'arrête. Nous récupérons donc un ensemble de cellules évaluées et un ensemble résidu ne contenant que des cycles de formules interdépendantes. Les cellules impactées par une mise à jour sont facilemment déterminables par extraction de la composante connexe associée.

1.2 Représentation mémoire et Évaluation de grandes feuilles de calcul

La précédente méthode ne permettait pas de travailler sur des *gros* fichiers: notre représentation sous forme de graphe étant trop volumineuses pour permettre de réprésenter des grandes feuilles de calculs en RAM. Nous avons donc choisit de nous doter d'une structure de stockage sur disque et non sur RAM (`toolkit/FileInterface.ml`). Afin de pouvoir manipuler cette structure facilemment nous avons serialisé les données de manière à ce que chaque cellule occupe un espace d'une taille fixée. Le module `FileInterface` offre une couche d'abstraction qui réponds aux problèmes de **sérialization**. Il a aussi fallu se doter d'une nouvelle approche pour l'évaluation.

The computation wheel

Cette solution consiste à *faire tourner une "roue"* (soit un ensemble de formules F). Lorsque la roue tourne, chaque formule $f \in F$ tente de s'évaluer **partiellement**. Si elle s'évalue **entièrement**, elle est supprimée de la roue, sinon elle y reste. Après avoir effectuer un tour complet, si l'état de la roue n'a pas changé (ie. aucune formule n'as pû s'évaluer) nous considérons être dans une situation bloquante car l'ensemble des formules dans la roue représente un cycle et nous nous arrêtons.

Un point important est que nous utilisons de l'évaluation partielle de formule, c'est à dire que nous ne tentons pas de les réduire entièrement, mais utilisons une approche pas-à-pas du calcul.

Un modèle de calcul

Face à la spécification du défi 3, il s'est averé nécessaire de pouvoir prioriser certains calculs par rapport aux autres. Nous nous sommes donc doté d'une abstraction au dessus du calcul nous permettant d'exprimer une sémantique pas-à-pas de ce dernier. Cela nous offre la possibilité de manipuler le flot d'execution et

de simuler une execution concurrente en entrelaçant manuellement les calculs. On peut alors controler l'ordonnancement des *processus* ainsi créés et donner une priorité plus forte à l'évaluation de certains d'entre eux, arreter arbitrairement un calcul en cours, le mettre en attente, etc...

Relacher les contraintes

Nous avons vu précédemment que nous nous placons dans un cadre où la terminaison de certains calculs dépend du résultat d'autres évaluations. Cela impose de mettre en attente ces calculs pendant un temps indéterminé dans l'espoir d'obtenir les informations nécessaires à leur progression, cette situation menant le plus souvent à un point où ces calculs ne sont plus en capacité d'avancer, bloquant à leur tour d'autres calculs. Pour autant cette contrainte peut etre assouplie en exploitant les résultats partiels d'une évaluation qui peuvent fournir suffisamment d'informations pour progresser malgré que le calcul observé n'est pas terminé, voir ne terminera jamais. En ce sens il semble intéressant de se tourner vers une autre vision du calcul, celle d'un flot de données passant par des processus de transformation de manière continue plutot que comme une séquence d'instructions bloquantes.