Sémantique et extension du langage C2QL pour la composition de techniques protégeant la confidentialité dans le nuage

Santiago Bautista

Juillet 2017

Résumé

Des applications de tout genre manipulent des personnelles de ses utilisateurs et utilisent le cloud pour s'exécuter ou s'héberger. Différentes techniques existent pour protéger la confidentialité de ces données. Pendant ce stage on a étudié la sémantique et prouvé les propriétés algébriques d'un langage permettant de décrire efficacement la composition de plusieurs de ces techniques, comme la fragmentation et le chiffrement.

Mots clés : privacy, cloud-computing, semantics, proof of correctness, algebraic laws, fragmentation, optimisation

Table des matières

1 Introduction

De plus en plus de logiciels sont développés pour être exécutés dans le cloud, et ses logiciels, de quelque sorte qu'ils soient (messagerie, gestion d'agenda personnel, commande de pizza ou reconnaissance vocale) traitent des données personnelles, qu'ils doivent donc protéger.

Une des propriétés qui doit être garantie dans la protection des données personnelles est la confidentialité. Différentes techniques existent pour protéger la confidentialité des données, comme par exemple le chiffrement et la fragmentation.

Dans sa thèse de 2016, Ronan Cherrueau montre que chacune de ces techniques a des avantages et des inconvénients, mais qu'*en composant les différentes techniques ensemble* on peut profiter de tous les avantages de ces techniques en éliminant la plupart des inconvénients. Il développe donc un langage, nommé C2QL (pour *Cryptographic Compositions for Query Language*), qui permet de décrire une telle composition de techniques de sécurisation des données pour en vérifier la correction et raisonner plus facilement.

Le langage se présente comme un ensemble de fonctions que l'on peut composer entre elles. Parmi ces fonctions, il y a les fonctions classiques pour faire des requêtes dans des bases de données, telles que la projection et la sélection, tout comme des fonctions décrivant la protection des données, comme le chiffrement ou la fragmentation.

Un des intérêts de ce langage est que, pour décider comment protéger les données des utilisateurs, le développeur peut suivre un processus simple en trois étapes. D'abord, le développeur écrit les requêtes *en ne tenant compte* ni du fait que le programme s'exécute dans le nuage, ni des mécanismes pour le protéger. Puis, il compose sa requête avec les fonctions de protection nécessaires (qui dépendent du problème en particulier, des contraintes de confidentialité spécifiques) pour avoir une requête sécurisée. Finalement, le développeur utilise des lois de commutation entre les différentes fonctions pour optimiser sa requête sécurisée.

Par conséquent, disposer de lois qui indiquent à quelles conditions les différentes fonctions du langage commutent est très important.

Or, si dans sa thèse R. Cherrueau donne la plupart de ces lois, il n'a pas eu le temps de les démontrer, ni de contempler tous les cas de figure.

C'est pourquoi, pendant ce stage, j'ai complété l'ensemble de lois fournies (section 3.2) dans la thèse de Ronan et j'ai formalisé la sémantique des différentes fonctions (section 3.1) pour ensuite démontrer la correction de ces lois (section 3.3).

Une description du contexte dans lequel s'inscrit ce stage est donnée à la section 2, et une discussion sur les aspects qui n'ont pas été traités est donnée à la section 4.

2 Contexte

2.1 De l'importance de composer les techniques de protection

De plus en plus d'applications, en particulier les applications web et les applications pour téléphone portable, cherchent à utiliser le nuage, soit pour stocker du code ou des données, soit pour faire des calculs, voir les deux à la fois.

En effet, le nuage peut offrir des services (que ce soit sous forme d'infrastructure, de plateforme ou de logiciel) disposant d'une forte disponibilité et faciles à redimensionner.

Autrement dit, pouvoir utiliser le nuage est devenu un enjeu de la conception logicielle, à cause des avantages de disponibilité et redimensionnement que cela offre.

Mais ce n'est pas le seul enjeux de la conception logicielle.

Vu que la plupart de ces applications manipulent des données personnelles, garantir la *confidentialité* des données est également un enjeux de ces applications là ; tout comme le sont les *performances* pour garantir une meilleure expérience à l'utilisateur de l'application.

Dans sa thèse, R. Cherrueau s'intéresse à trois techniques particulières utilisées dans le développement logiciel et regarde comment elles interagissent avec les trois enjeux cités plus haut. Ces trois techniques, qu'on va décrire brièvement, sont le *chiffrement*, la *fragmentation verticale* et l'*exécution* de l'application *chez l'utilisateur*.

Le chiffrement Lorsqu'il est bien utilisé, le chiffrement permet de garantir la confidentialité des données de l'utilisateur. De plus, dans certains cas, des calculs peuvent être faits sur les données chiffrées. On appelle chiffrement homomorphe un chiffrement avec lequel on peut effectuer des calculs avec les données chiffrées. Les chiffrement homomorphes totaux, comme celui de Gentry (**référence à ajouter**) sont pour l'instant trop contraignants pour pouvoir être utilisés dans la plupart des applications, mais les chiffrements homomorphes partiels, c'est à dire les chiffrement avec lesquels on peut effectuer *certaines* opérations sur les données chiffrées, peuvent se révéler très utiles. C'est le cas des chiffrements déterministes (dont les chiffrements symétriques) qui sont des chiffrement homomorphes partiels, permettant le test d'égalité.

Dans tous les cas, le chiffrement implique un surcoût en terme de calculs, donc diminue les performances. Dans certains cas, il améliore la confidentialité et permet l'utilisation du nuage.

L'exécution côté client Si le programme était exécuté entièrement par la machine de l'utilisateur, cela serait à la fois bon pour la confidentialité (car les données ne seraient pas du tout exposées aux risques liés à l'utilisation du nuage et du réseau) et pour les performances, puisque, à moins de traiter une trop grande quantité de données dans un calcul hautement parallélisable, les performances du cloud sont moins bonnes que celles des machines des utilisateurs. Par contre, l'exécution côté client ne permet pas de profiter des avantages du cloud.

La fragmentation verticale consiste à séparer les différentes données que manipule le programme entre deux clouds n'ayant aucun rapport entre eux (à deux endroits géographiques différents, gérés par des entités différentes, etc. . .). Ceci permet de protéger celles des données personnelles qui sont constituées d'une *association* de deux données. Par exemple, dans une application stockant un ensemble de rendez-vous, l'association $(\text{date}, \text{lieu})$ doit être protégée, car une personne malveillante ayant accès à ces deux informations là à la fois pourrait suivre l'utilisateur de l'application.

La fragmentation verticale contribue à protéger la confidentialité, mais d'une façon souvent moins forte que celles du chiffrement ou de l'exécution côté client. Par contre, chacun des fragments de données qu'elles génère peuvent être opérés séparément, en introduisant ainsi, lorsque le programme le permet, une dose de parallélisation supplémentaire qui peut

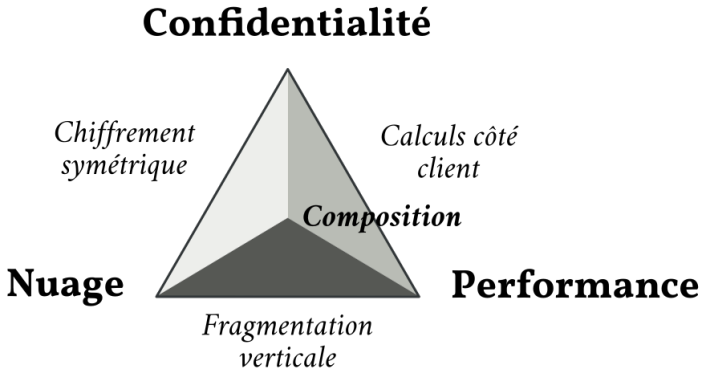

FIGURE 1 – Enjeux et techniques dans le cloud-computing

(image provenant de la thèse de Ronan Cherrueau)

améliorer les performances et permettre de tirer encore plus d'avantages du nuage.

La figure 1 résume comment ces trois techniques là interagissent avec les trois enjeux cités plus haut.

Dans sa thèse, Ronan Cherrueau montre qu'en composant ces différentes techniques on peut à la fois profiter des avantages du nuage, protéger la confidentialité et améliorer les performances d'un programme.

Pour pouvoir décrire comment s'effectue une telle composition, pour pouvoir vérifier la correction d'une telle composition et pour pouvoir raisonner dessus, Cherrueau a introduit un langage : C2QL.

2.2 Un langage pour décrire la composition : C2QL

Le langage C2QL donne une façon d'exprimer comment les techniques mentionnées ci-dessus se composent avec les fonctions classiques de l'algèbre relationnelle.

Les données manipulées sont donc représentées sous forme de tables, ou relations, c'est à dire un ensemble de lignes contenant des valeurs pour chacun(e) des différent(e)s attributs ou colonnes considéré(e)s.

Dans l'exemple de l'application stockant des rendez-vous, cette table contiendrait autant de lignes que de rendez-vous stockés dans l'application et (par exemple) trois colonnes ou attributs : nom de l'utilisateur ayant stocké le rendez-vous, date du rendez-vous et lieu du rendez-vous.

Les opérateurs empruntés à l'algèbre relationnelle présents dans ce langage sont

- La projection, notée π qui consiste à ne considérer que certains des attributs de la table.
- La sélection, notée σ qui consiste, pour une table donnée, à ne considérer que les lignes satisfiant un certain prédicat.
- L

3 Contribution

3.1 Établir des définitions

3.2 Compléter les propriétés

3.3 Prouver les propriétés

3.4 Optimiser les requêtes

4 Travail futur

5 Conclusion

Sémantique de C2QL

Santiago Bautista

Juin 2017

Le but de ce document est de donner une définition formelle des fonctions dont est composé le langage C2QL.

Préambule

Définition 1 *On appelle* nom d'attribut *toute chaîne de caractères.*

Ici, pour simplifier, on appelle chaîne de caractères tout mot sur l'alphabet

$$\Sigma = \{a, \ldots, z\} \cup \{A, \ldots, Z\} \cup \{0, \ldots, 9\}$$

Vu que le nom d'attribut « id » joue un rôle particulier, on appelle, par opposition, nom d'attribut régulier *tout nom d'attribut autre que « id ».*

Définition 2 *On appelle* schéma relationnel *tout ensemble de noms d'attributs réguliers.*

Définitions générales

Définition 3 *On appellera* valeur *tout élément d'un certain ensemble $\mathcal{V}$, que l'on suppose non-vide, infini, dénombrable, et stable par formation de n-uplets (i.e. $\forall k \in \mathbf{N}, \mathcal{V}^k \subset \mathcal{V}$).*

Définition 4 *On appelle* relation *de schéma relationnel Δ tout ensemble de fonctions de $\Delta \cup \{id\}$ dans $\mathcal{V}$.*

Chacun des éléments de la relation (chacune de ces fonctions) est appelé(e) ligne.

Pour chaque ligne l de la relation et chaque α de Δ, $l(\alpha)$ est appelé attribut *de* nom *α pour la ligne l.*

L'image de id est appelée identifiant de la ligne*, et elle est, au sein de chaque relation, unique pour chaque ligne.*

Définition 5 *On appelle* S *l'ensemble des schémas relationnels possibles. Autrement dit, on pose* $\mathrm{S} = \mathcal{P}\left(\Sigma^* \setminus \{id\}\right)$.

On appelle T *l'ensemble des relations possibles,*

et on introduit la fonction sch *de* T *dans* S *qui à une relation associe son schéma relationnel.*

Projections et sélections

Définition 6 *Pour tout ensemble δ de noms d'attributs réguliers, on appelle* projection sur les attributs *δ la fonction suivante :*

$$\begin{array}{rccc} \pi_\delta : & \mathrm{T} & \to & \mathrm{T} \\ & r & \mapsto & \{l|_{(\delta \cap \mathrm{sch}(r)) \cup \{id\}} / l \in r\} \end{array}$$

6