Le contenu de ce document provient de l'article figurant dans l'adresse suivante :

https://blog.ippon.fr/2016/04/07/modelisation-cassandra-stocker-des-fichiers/

Stocker des fichiers

Enregistrer des fichiers est un besoin courant. Les réseaux sociaux enregistrent les photos de leurs utilisateurs. Les catalogues de produits stockent leurs images. Les applications de messagerie ou de gestion de dossiers permettent d'attacher des pièces jointes. À chaque fois, vous aurez le choix entre conserver les fichiers dans la base de données applicative ou dans un système spécialisé à part. Il s'agit d'un choix d'architecture qui dépend des besoins et des volumes de chaque cas.

Stocker dans la base de données permet de simplifier l'architecture du système en évitant d'ajouter un composant technique supplémentaire. Cela facilite aussi la gestion des sauvegardes puisque toutes les données sont présentes au même endroit. Avec un seul système à sauvegarder et à restaurer, la cohérence des données après restauration est garantie. Ce choix est adapté aux applications qui conservent peu de petits fichiers.

Disposer d'un système spécifique, c'est avoir la garantie de conserver les fichiers dans un système adapté. On évite ainsi de perturber les performances de la base de données avec les données binaires. Et l'on peut utiliser des mécanismes de stockage moins chers. Ce choix sera généralement mis en œuvre quand le nombre ou la taille des fichiers devient important.

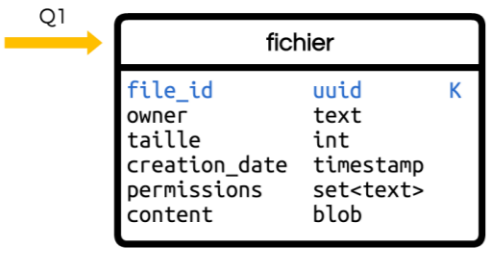
Première modélisation

Nous allons commencer par le cas le plus simple à modéliser. Il s'agit tout simplement de créer une table pour conserver l'intégralité des données. Cette table unique est identifiée par l'identifiant du fichier. Elle est interrogée à partir de celui-ci

Elle contient en plus des métadonnées, ou données descriptives, qui dépendent de votre application. Dans l'exemple, les métadonnées sont :

- la taille du fichier,
- le propriétaire,
- la date de création,
- un ensemble de permissions.

Le contenu y est enregistré à côté des métadonnées descriptives.



Q1: Lire un fichier à partir de son identifiant

Elle présente des avantages majeurs :

- Le modèle est simple et facile à comprendre.
- Une seule requête suffit pour récupérer le fichier et ses métadonnées.

Elle présente aussi un inconvénient vital :

- Ce modèle n'est viable que si la taille des fichiers est petite et bornée. En effet, Cassandra est sensible aux partitions de trop grandes tailles qui saturent sa mémoire et sollicitent le garbage collector.

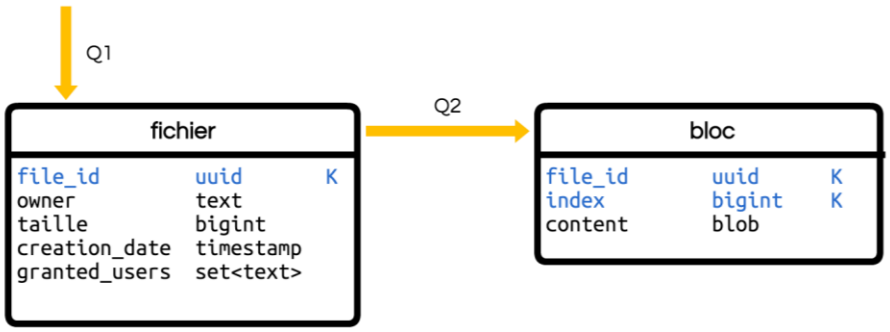
Ainsi, vous pouvez envisager cette solution lorsque vos fichiers ont une taille moyenne de 50 Ko et maximale de 5 à 10 Mo.

Découper les fichiers

Dans le cas général, il sera préférable de découper les fichiers en blocs qui seront stockés dans des partitions différentes. Pour cela, deux tables seront utilisées :

1. La première contient les métadonnées du document.
2. Et l'autre le contenu.

Le choix de la taille d'un bloc est l'objet d'un compromis entre le nombre de blocs, et donc le nombre de requêtes nécessaires pour la lecture et l'écriture du contenu, et la fluidité de leur gestion. Une taille de 64 Kio semble raisonnable de nos jours. C'est d'ailleurs, la taille des blocs de MongoDB GridFS.



Q1: Lire la description d'un fichier à partir de son identifiant
Q2: Lire le contenu d'un fichier bloc par bloc

On obtient alors le modèle suivant:

Cette fois-ci, on a :

1. une table contenant les métadonnées,
2. une table contenant les blocs de données.

Indiquer la taille du fichier est devenue nécessaire, car elle permet au lecteur d'en déduire le nombre de blocs à lire.

Chaque bloc est identifié par l'ID du fichier et un index représentant sa position dans le fichier. La clé primaire est identique à la clé de partition pour s'assurer de la maitrise de la taille de cette dernière.

Le modèle s'organise autour de deux requêtes:

- **Q1** vérifie l'existence du fichier et lit les métadonnées, dont la taille.
- **Q2** lit un bloc de données.

Content Addressable Storage

La modélisation actuelle fonctionne correctement. Cependant, il est possible d'optimiser l'espace consommé. En effet, jusqu'à présent, un fichier déposé par des chemins différents sera enregistré plusieurs fois. Ce sera le cas si plusieurs personnes téléversent le même document. Imaginez le nombre de copies d'un courriel ou de ses pièces jointes dans un système d'archivage de la messagerie d'une grande entreprise!

Pour éviter le gaspillage, nous allons tenter de dédoublonner les fichiers et ne conserver qu'une copie du contenu.

Une solution naïve consisterait à chercher les fichiers de même contenus. Évidemment, on ne cherche pas le contenu lui-même, mais un condensat, ou une empreinte, qui permet de trouver le doublon. Si cette solution fonctionne en principe, elle n'est pas adaptée à Cassandra puisqu'elle impose une lecture avant l'écriture (read-before-write). En plus, il faudra prévoir une vue matérialisée pour permettre cette lecture.

On peut faire mieux en utilisant directement l'empreinte comme identifiant. Au lieu d'être arbitraire, l'identifiant devient intrinsèque au contenu. Sa valeur est déterministe et indépendante du contexte. Cette pratique est suffisamment courante pour avoir un nom : *Content Addressable Storage*, ou stockage indexé par le contenu. Et vous l'utilisez tous les jours, puisque *git* qui identifie ses objets par leur SHA-1 repose sur un CAS. Cependant, le CAS présente des contraintes. Par exemple, le contenu d'un fichier ne peut pas changer puisqu'un changement de contenu change l'identifiant. De plus, dans le cadre d'une gestion de fichiers, deux fichiers de même contenu sont identiques.

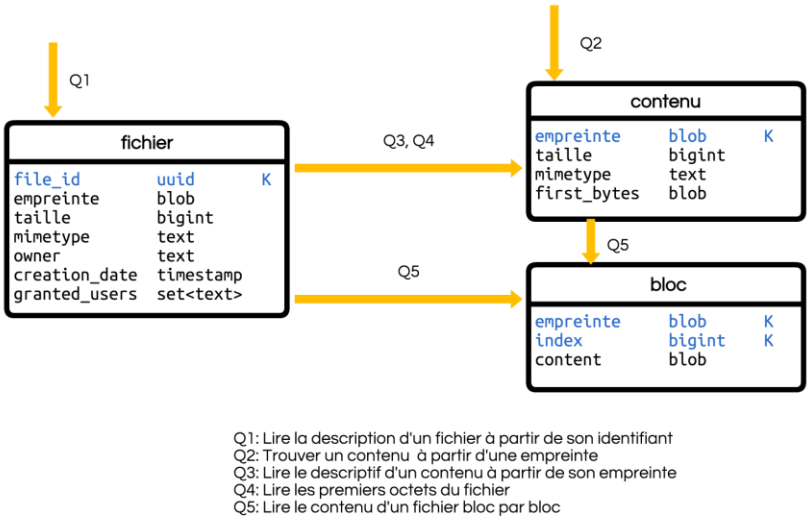
Dans le cas pratique, on utilisera le CAS pour la conservation du contenu et de sa description intrinsèque. On utilisera un stockage traditionnel pour la description contextuelle du fichier.

Ainsi, on a:

1. une table *fichier* identifiée par un uuid contenant :
    - les métadonnées contextuelles (propriétaire, date de création, permissions) ;

- le lien vers le contenu et des champs dénormalisés ;

2. une table *contenu* identifiée par l'empreinte et contenant les informations descriptives du contenu ;
3. une table *bloc* avec le contenu découpé en bloc.



Une optimisation utile permet de stocker les premiers octets, souvent le 1er Kio, dans les métadonnées (champ first_bytes). Ce peut être utile dans le cas où il est fréquent de ne lire que les entêtes (pour déterminer le type du fichier par exemple) ou lorsque l'application gère de nombreux petits fichiers.

Ce modèle fonctionne bien pour l'écriture sans modification des contenus. Dans le cas ou un fichier peut être supprimé ou que son contenu peut être modifié, il faudra ajouter un mécanisme de collection des contenus inutilisés. Il s'appuiera sur un champ permettant de savoir si un contenu est utilisé.