Polycopié pour le cours de MongoDB

Romain Tavenard

Chapitre 1

Introduction

Ce document est une tentative de polycopié associé au module de MongoDB pour la première année de Master Mathématiques Appliquées - Statistiques de l'Université de Rennes 2. Il est distribué librement (sous licence CC BY-NC-SA plus précisément) et se veut évolutif, n'hésitez donc pas à faire vos remarques à son auteur dont vous trouverez le contact sur sa page web.

Durant la lecture de ce polycopié, vous trouverez des blocs de code tels que celui-ci :

```
> db.users.find({"name": "Tavenard"})
```

Nous prendrons notamment l'habitude de précéder les requêtes du symbole > et d'afficher le résultat obtenu.

Ce polycopié s'inspire largement du très bon livre (disponible à la BU de l'Université de Rennes 2 et en PDF sur le web) suivant :

> K. Chodorow. MongoDB, The Definitive Guide. O'REILLY

1.1 Rappels : Bases de données relationnelles

Les bases de données que vous avez manipulées jusqu'à présent (*cf.* module du premier semestre de M1) reposent sur le modèle relationnel. Rappelons ici quelques unes des propriétés de ce modèle pour mieux comprendre l'apport des modèles dits **NoSQL** que nous étudierons dans ce cours.

Tout d'abord, le modèle relationnel est très structuré : les bases de données sont organisées en tables, à chaque attribut est associé un type unique et on peut spécifier un certain nombre de contraintes d'intégrité (contrainte d'unicité pour les clés primaires, contrainte d'existence pour les clés étrangères, *etc.*). On peut

par exemple disposer d'une base contenant une table `etudiant` qui stocke les données suivantes :

idetudiant	nom	prenom	ville
1	Perrier	Jean	Rennes
2	Martin	Aline	Mulhouse

On utilise (la plupart du temps), pour interroger ces bases, le langage SQL (d'où le qualificatif NoSQL associé aux bases ne reposant pas sur ce modèle) comme dans l'exemple suivant :

```
SELECT nom, prenom FROM etudiant WHERE ville = "Rennes"
```

Lorsqu'on manipule des bases de données relationnelles, on cherche à limiter tant que faire se peut la redondance des données. Pour cela, on sépare les données dans différentes tables, et c'est au moment de formuler une requête que l'on spécifiera le moyen de faire le lien entre les différentes tables par le biais d'une **jointure**. Reprenons l'exemple précédent et supposons maintenant que notre base de données contient une seconde table `cours` avec le contenu suivant :

idetudiant	matiere	note
1	NoSQL	17
1	Machine Learning	12
2	NoSQL	10

Pour obtenir les noms des étudiants et leurs notes dans la matière `"NoSQL"`, on exécutera la requête suivante :

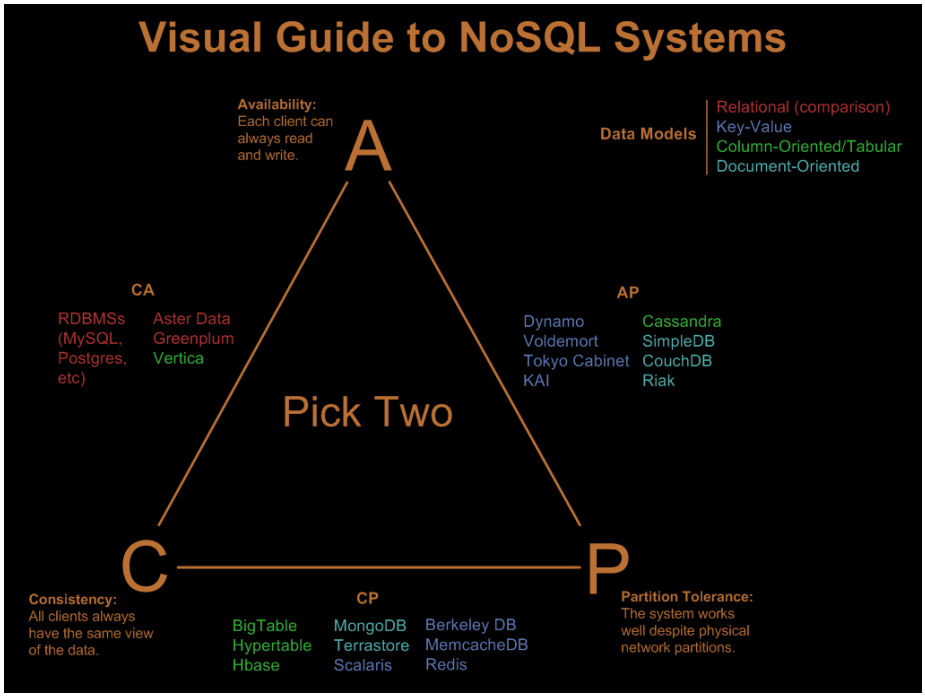
```
SELECT nom, note FROM etudiant NATURAL JOIN cours
WHERE cours.matiere = "NoSQL"
```

Toutefois, pour des raisons qui dépassent le scope de ce cours, les bases de données relationnelles posent un certain nombre de problèmes lorsqu'il s'agit de les distribuer sur plusieurs serveurs. Cela devient problématique lorsqu'il s'agit de manipuler de gros volumes de données ou de répondre à un nombre important de requêtes dans un temps limité.

En pratique, le théorème CAP (Brewer, 2002) stipule qu'il est impossible pour un système distribué de garantir simultanément les 3 propriétés suivantes :

- Cohérence (*Consistency*) : tous les noeuds voient les mêmes données à chaque instant ;
- Disponibilité (*Availability*) : toute requête est traitée en un temps fini ;
- Tolérance à la partition (*Partition Tolerence*) : le système est robuste à une organisation des noeuds en grappes peu connectées entre elles.

Ainsi, les différents SGBD (Systèmes de Gestion de Bases de Données) ne peuvent que se contenter de garantir deux de ces trois propriétés à la fois. Le graphique suivant (issu de http://blog.nahurst.com/visual-guide-to-nosql-systems) illustre les choix faits par un certain nombre de systèmes actuels (dont MongoDB) :



1.2 Exécution de MongoDB sous UNIX

TODO lancement terminal