Préparation à l'examen de *NoSQL*

Wéry Benoît

23 novembre 2017

Chapitre 1

Vrai ou Faux

1. *Dans une entreprise, les données vivent souvent plus longtemps que les logiciels.* Vrai

   Une entreprise *utilise* des logiciels et *stocke* des données en veillant à garder le plus d'indépendance possible entre ces deux éléments. Ainsi, un changement d'implémentation de la structure des données ne devrait pas affecter le bon fonctionnement du logiciel.

   Là où les données doivent persister dans le temps, un logiciel peut être amené à être remplacé (par obsolescance, optimisation,...)

2. *Dans une entreprise, les logiciels vivent souvent plus longtemps que les données.* Faux

3. *Le passage du relationnel au NoSQL se fait généralement au profit d'une diminution des garanties relatives à la consistence des données.* ???

4. *Le passage du relationnel au NoSQL se fait généralement au profit d'une diminution de la quantité de données stockables.* Faux

   ... stockables VS stockées...

   Le NoSQL a émmergé d'un besoin de stocker de nouveaux formats de données (ex : stocker directement des objets plutôt que de devoir retrouver leur informations indépendantes pour le reconstruire), malgré la *puissance* et la *stabilité* des base de données relationnelles, celles-ci ne sont plus suffisantes dans certaines applications.

5. *Le passage du relationnel au NoSQL se fait généralement au profit de l'abandon de la possibilité de lire des données de manière concurrente.* Faux

   Le modèle relationnel permet d'accéder de manière concurrente aux données (en R/W) via l'utilisation de **transactions**, les données sont stockées de façon consistante et persistante.

   Le modèle NoSQL, quant à lui, permet un accès concurrent aux données MAIS elles ne sont pas garanties consistantes. De plus, les données d'une BDD étant stockées sur des serveurs différents (contrairement au relationnel -> 1machine/BDD), il se peut que des requêtes identiques fournissent des résultats différents. ... A VERIFIER

6. *Le passage du relationnel au NoSQL se fait généralement au profit d'une diminution des garanties relatives à la persistance des données.* ???

   "la gestion de la persistance des données réfère au mécanisme responsable de la sauvegarde et de la restauration des données. Ces mécanismes font en sorte qu'un programme puisse se terminer sans que ses données et son état d'exécution ne soient perdus". A VERIFIER

7. *Tout comme pour le relationnel, l'organisation des données en NoSQL suit un modèle mathématique rigoureux.* Faux

   Là où le relationnel se base sur des modèles mathématiques standards pour définir la structure de la BDD, le NoSQL ne suit pas de schéma (ex : possibilité d'ajout de champs sans contrôle).

   De cette façon, on obtient une structure **flexible** avec une BDD qui n'est pas figée dans le temps, ce qui est utile dans le cas de *prototypages*.

8. *Le NoSQL est particulièrement adapté à des traitements de données de type OLTP* Faux

   Rappel : Online Transactional Processing (OLTP)... modèle qui utilise des données dans un but purement transactionnel -> gestion

   Une transaction doit faire appel à des données consistantes, ce qui n'est pas garantit dans le cas du NoSQL ... BONNE RAISON ?

9. *Le NoSQL est particulièrement adapté à des traitements de données de type OLAP.* Vrai

   Rappel : Online Analytical Processing (OLAP)... modèle qui utilise les données dans un but d'analyse et de prédictions -> statistiques

10. *Toutes les bases de données de type NoSQL satisfont les propriétés ACID.* Faux

    Rappel : proriétés ACID...
    - **A**tomicity : une transaction fait tout ou rien
    - **C**onsistency : la BDD change d'un état valide vers un autre état valide, les données sont constamment à jour
    - **I**solation : une transaction doit être exécutée sans avoir connaissance de l'existance des autres
    - **D**urability : une transcation validée et confirmée est sotckée, durable dans le temps

    De telles conditions correspondent au modèle relationnel.

    Un autre ensemble de conditions, le modèle BASE, permet de gérer les *pertes de consistance* en maintenant la fiabilité et correspond donc à l'approche NoSQL :
    - **B**asically **A**vailable : le système renverra toujours une réponse, même si la donnée n'est pas à jour ou qu'il s'agit d'un message d'erreur.
    - **S**oft state : l'état change constamment au cours du temps (même lorsqu'il n'y a pas d'inputs) pour essayer de se stabiliser entre les serveurs de la BDD
    - **E**ventual consistency : le système finira tôt ou tard par être consistant

11. *Un système distribué peut toujours garantir la consistence des données sur tous ses nœuds.* Faux

    Les données peuvent être différentes d'un noeud à l'autre en fonction des mises à jour qui ont été faites ou non.

12. *Le mouvement de l'Open Data consiste à fournir librement des données récoltées pour permettre à la communauté de les analyser.* Vrai

    Il s'agit de données collectées par de grands groupes (Nasa, Nsa, Union Européenne,...) et qui sont rendues publiques. On peut considérer cela comme une situation "win/win" puisque les utilisateurs ont librement accès à une quantité gigantesque d'informations et qu'ils peuvent eux-mêmes en faire des analyses, utiles dans ce cas-ci pour les groupes qui fournissent les données.

13. *Le mouvement de l'Open Data à créer des logiciels open source permettant d'analyser des données massives (big data).* ???

    ??? sens phrase ???

    Le **Big Data** consiste en l'**augmentation du volume de données** manipulées (ex : données scientifiques, réseaux sociaux, opérateurs tél., indicateurs écono et socio,...).

    Il y un réel *défi* afin de gérer et traiter cette énorme quantité de données et pour y parvenir, il faut se tourner vers d'autres modèles de BDD que le relationnel car il n'est plus adéquat.

14. *Google File System (GFS) est un moteur de base de données NoSQL.* Vrai

    GFS est un **système de fichiers distribué** optimisé pour fonctionner sur un *cluster*. Il est utilisé pour gérer des fichiers de taille importante (découpés en blocs et stockés sur différentes machines) qui sont rarement supprimés ou réécrits. La plupart du temps, les accès portent sur la lecture de larges zones ou des ajouts en fin de fichiers. Pour chaque fichier, il en existe une/des copie(s) pour subdvenir aux éventuelles pannes de serveurs.

15. *Apache Hadoop est une implémentation propriétaire de MapReduce, commercialisée par Oracle.* Faux

    Il s'agit d'une implémentation **libre** de MapReduce (en Java)

    Rappel : MapReduce... il s'agit d'un *modèle de programmation* dont le but est de traiter et générer de grandes quantités de données. Il se base sur des algorithmes parllèles et distribués sur un cluster. Une implémentation de MapReduce se base sur deux fonctions :

    (a) Map effectue un **traitement sur une liste** (tri, filtre,...)

    (b) Reduce regroupe les données en un résultat.

16. *L'intégration des données dans une seule base pour les partager entre plusieurs applications permet d'obtenir les meilleures garanties en terme de préservation de l'intégrité des données.*

17. *Limiter l'accès aux bases de données d'une entreprise à une seule application qui offre une API aux autres permet de rendre un changement de leur structure plus facile.*

18. *Une base de données NoSQL (clé-valeur, document et colonne) stocke des agrégats que l'on peut comparer aux tables du modèle relationnel* Vrai

    Là où le modèle relationnel utilise des tables avec des rangées pour stocker les données, le modèle NoQSL utilise des "agrégats" qui sont des **unités d'information** qui peuvent être traîtées, stockées et échangées de façon atomique.

Ces unités de données sont **plus complexes** et **plus structurées** que leurs homologues du modèles relationnel, qui sont de simples tuples. Les possibilités de traîtement qui en résultent sont donc plus complètes.

19. *Une collection de paires clé-valeur peut être assimilée à une table relationnelle à deux colonnes dont la colonne représentant les clés est la clé primaire de la table.* Vrai

    Les paires "clé-valeur" sont identifiables au moyen de la clé. L'avantage remarquable de ce système est que **la clé n'a pas de forme stricte** ! !

    La clé est un *string* [...]

    De plus, la clé n'a pas conscience du format de la valeur qu'elle refère car celle-ci est de type **blob (= Binary Large OBject)**. Autrement dit, la valeur peut être quelconque (ex : *sting, JSON, XML, objet, ...*), c'est à l'application de gérer le format de chaque valeur. De ce fait, le modèle n'a ni schéma, ni structure pour le stockage et il est donc impossible de récupérer une partie de la valeur (-> tout ou rien).

    Les avantages de ce modèle sont : efficacité de recherche et la simplicité

20. *Supprimer la valeur associée à une clé est l'une des trois opérations de base que l'on peut réaliser sur une base de données clé-valeur.* Vrai/ Faux ? ? ?

    Les trois opérations de base sont :

    (a) **Récupérer** une valeur à partir de sa clé [Read(key)]

    (b) **Définir** la valeur d'une clé [Create/Update(key, value)]

    (c) **Supprimer** une clé [Delete(key)]

    On ne sait pas supprimer la valeur d'une clé sans supprimer la clé elle-même.

21. *Dans une base de données clé-valeur, il est généralement prévu de rechercher toutes les clés dont les valeurs satisfont une certaine propriété.* Faux

    La clé étant de type *string*, elle est utilisé pour définir des formats/patterns utilisés pour des recherches efficaces.

22. *Il est possible d'imposer des contraintes sur les domaines des valeurs des paires clé-valeur d'une base de données clé-valeur.*

    COMPLETER [...]

23. *Distribuer les données sur un cluster de machines fait partie des éléments mis en place dans le monde NoSQL.* Vrai

    Le modèle NoSQL se prête bien à la distribution des données (agrégats) sur différentes machines physiques (= clusters, architecture en noeuds).

Intérêts du cluster...

- gestion de grandes quantités de données, *scale out* : la capicité de stockage peut être facilement augmentée en ajoutant de nouvelles machines et ce "à chaud"
- meilleur trafic R/W : la répartition de la charge permet de fluidifier le trafic des requêtes
- résister à des ralentissements, pannes réseaux : les machines peuvent être inscrites chez différents Fournisseurs d'Accès Internet (FAI) et le réseau est de ce fait moins sensible à une éventuelle panne chez un FAI ou une saturation

24. *Il est possible de faire du sharding de données pour une base de données se trouvant sur une machine unique.* Vrai ? (via machine virtuelle) -> mais complètement inutile

Le sharding est une technique de distribution des données qui consiste à répartir la charge entre différents serveurs de façon horizontale (= [...]).

En sharding, les données **ne sont pas répliquées** et chaque serveur est accessible en **R/W**. Pour optimiser la distribution, on veillera à rassembler sur un même noeud les données qui sont *accédées ensemble* (probabilité d'accès commun et localisation géographique). C'est le client qui a connaissance de la répartition des données et qui est donc responsable d'interroger la bonne machine.

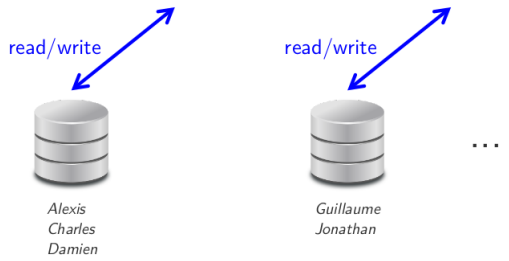

FIGURE 1.1 – Distribution des données : modèle sharding [1]

Avantages :

- fraction de la DB pour accélérer son traîtement et la rendre plus facile à gérer
- avantages cluster (résistance aux pannes, *scale out*, ...)

Inconvénient :

- définir bonne répartition des données -> risque de surcharge d'un serveur
- pas de résilience (ni en lecture, ni en écriture) -> si un serveur tombe, une partie de la DB n'est plus du tout accessible

25. *Le sharding permet de récupérer les données en cas de corruption grâce à un stockage redondant de ces dernières sur plusieurs serveurs pouvant être physiquement à des endroits différents.* Faux

En sharding les données ne sont pas répliquées contrairement aux modèles de réplications *master/slave* ou *peer-to-peer*.

26. *Réplication de données et sharding sont incompatibles.* Faux

Il est possible de combiner les techniques de *sharding* et *replication* ce qui offre les avantages de résilience (R, W ou les deux) et de répartition de la DB.

On peut combiner le sharding avec :
- le *master/slave* : plusieurs maîtres (responsable chacun d'une partie de la DB) ou rôle mixtes (esclave pour certaines données et maîtres pour d'autres)
- le *peer-to-peer* : fraction de la DB (sharding) et réplication sur N noeuds

27. *La réplication master-slave offre la propriété de résilience à la lecture.* Vrai

Dans le modèle de réplication master/slave, on distingue deux types de noeuds

(a) le **master** : responsable des données et de leur mise à jour, il est accessible en R/W

(b) les **slaves** : répliques complètes (!!) du maître, accessibles seulement en lecture

Les requêtes des clients sont redirigées suivant que ce soit pour de la lecture ou de l'écriture. Comme le master est le seul accessible en lecture, il doit communiquer ses modifications aux esclaves pour les mettre à jour.

Ce modèle possède la propriété de *read resilience*, càd que si le maître crash, le système sera toujours accessible en lecture via les esclaves. De plus, il est possible d'élir (manuellement ou automatiquement) un esclave comme nouveau maître si celui-ci tombe.

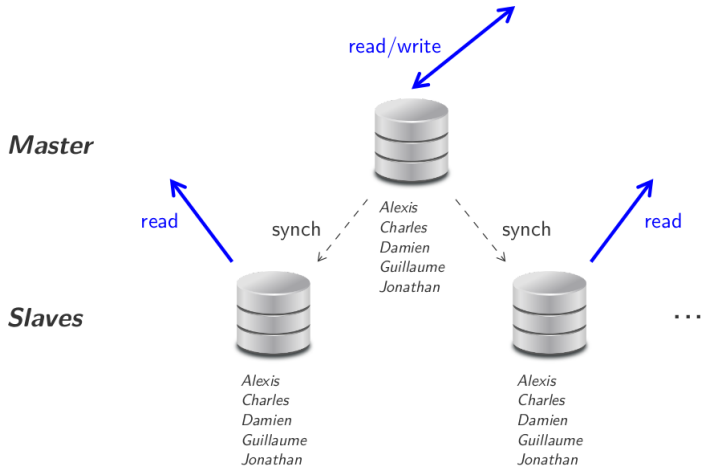

FIGURE 1.2 – Distribution des données : modèle de réplication master/slave [1]

Avantages :
- robustesse en cas de crash et *read resilience*
- fluidification du trafic pour les requêtes de lecture
- avantages cluster (résistance aux pannes, *scale out*, ...)

Inconvénient :
- pas adapté lorsqu'il y a beaucoup d'écritures -> surcharge maître
- nécessite plus d'espace de stockage puisque chaque noeud est une copie complète du maître
- les valeurs lues par plusieurs utilisateurs peuvent être différentes par inconsistence -> la cohérence des données n'est pas garantie, puiqu'il faut que le maître ait synchronisé ses modifications avec les esclaves

28. *En utilisant une réplication master-slave, les données deviennent complètement inaccessibles une fois que le master tombe.* Faux

Par la propriété de *read resilience*, les données seront toujours accessibles en lecture si le maître crash (avec risque de perte d'information si l'esclave n'était pas à jour par rapport au maître). Elles ne seront disponibles en écriture **que si** un esclave est désigné pour remplacer le maître (-> pas *write resilience*)

29. *La consistence des données est plus compliquées à garantir avec une réplication master-slave qu'avec une réplication peer-to-peer.* Vrai

Dans le modèle de réplication *peer-to-peer*, les noeuds sont tous égaux, ils sont **accessibles en lecture et en écriture**. A chaque écriture sur un noeud, tous les autres doivent être mis à jour, ce qui peut créer des conflits d'écriture concurrente si une même donnée est modifiée à deux endroits différents en même temps.

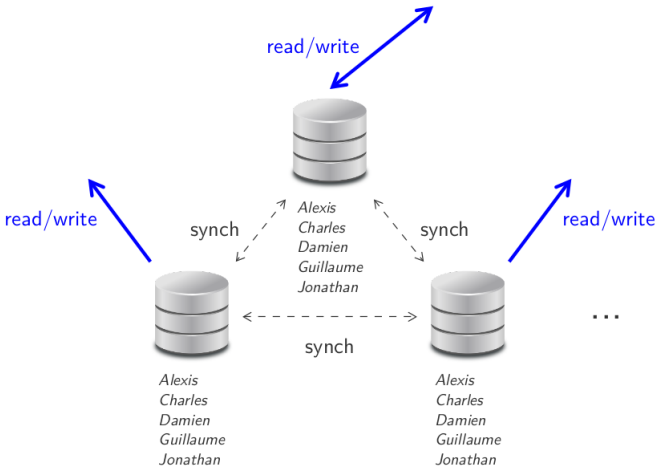

FIGURE 1.3 – Distribution des données : modèle de réplication peer-to-peer [1]

Avantages :
- robustesse en cas de crash et *complete resilience*
- fluidification du trafic pour les requêtes de lecture ET d'écriture

Inconvénient :
- pas adapté lorsqu'il y a beaucoup d'écritures -> lourdeur synchronisations
- nécessite plus d'espace de stockage puisque chaque noeud contient toute la DB
- risques de conflits d'écritures concurrentes
- peu évoluable -> l'ajout d'un nouveau noeud impose de le connecter à TOUS les autres noeuds existants

30. *La consistence des données est plus compliquées à garantir avec une réplication peer-to-peer qu'avec une réplication master-slave.* Faux

En master/slave, les modifications peuvent **seulement** être effectuées via le maître. Il y a donc des risques d'inconsistence des données le temps que les esclaves soient mis à jour MAIS les modifications seront partout les mêmes.

En peer-to-peer, des modifications peuvent être faites **sur n'importe quel noeud** qui communique ensuite à tous les autres ses modifications. Des écritures concurrentes sur différents noeuds peuvent donc engendrer des problèmes d'inconsistence de données qui sont plus difficile à vérifier. (Rem : une solution serait de synchroniser tous les noeuds sur une même date/heure et d'utiliser un serveur dédié pour stocker les informations de mise à jour des données. Ainsi, on peut facilement déterminer la plus récente)

31. *En utilisant une réplication master-slave, une lecture sur le master assurera toujours d'obtenir les données les plus récentes* Vrai

Puisque le master est **le seul noeud accessible en écriture**, ses données sont les plus à jour (contrairement aux esclaves qui doivent être synchronisés).

32. *Les buckets de Riak permettent de segmenter les données en plusieurs collections d'agrégats.*

Riak est un **moteur NoSQL décentralisé/distribué** (= Système de gestion de base de données SGBD) orienté clé-valeur. Il est très bon pour monter en charge, càd augmenter le volume de stockage, et a une haute tolérance aux pannex (peut enlever facilement un noeud en panne sans perte d'intégrité des données).

Ce SGBD stocke les clés dans des ***buckets***, qui agissent comme des **espaces de noms** pour les clés. Autrement dit deux clés peuvent porter le même noms si elles se trouvent dans des buckets différents.

Dans le modèle clé-valeur, lorsqu'on fait une requête sur une clé, toute la valeur est retournée et il n'est pas possible de cibler certaines parties (partie gauche [**?**]). Le système de buckets remédie d'une certaine façon à ce problème en permettant la **segmentation des données** dans des buckets séparés (partie droite [**?**]).

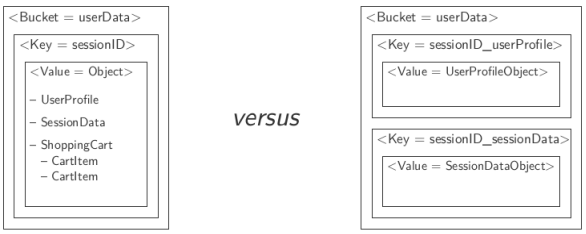

FIGURE 1.4 – Riak - système de *buckets* [1]

Avantages :
- pratique lorsqu'on sait que les données d'une même valeur ne seront jamais utilisées en même temps -> évite de récupérer l'ensemble du contenu de la valeur

33. *On ne peut pas stocker des arbres binaires comme valeurs avec Redis.*

34. *Redis garantit la persistance de données.*

35. *Les bases de données orientée colonnes optimisent le stockage disque pour des tables qui contiennent de nombreuses lignes.*

36. *Une base de données orientée colonnes est très adaptée lorsqu'on a plus d'opérations d'écriture que de lecture.*

37. *Une base de données orientée colonnes est très adaptée lorsqu'on a plus d'opérations de lecture que d'écriture.*

38. *Une base de données orientée colonnes est un map à deux niveaux.*

39. *Dans une base de données orientée colonnes, les familles de colonnes sont de préférence définies une fois pour toute lors de la création de la table.*

40. *L'avantage de l'utilisation de colonnes plutôt que de lignes est d'offrir une vitesse d'écriture plus grande de nouveaux enregistrements.*

41. *L'avantage de l'utilisation de colonnes plutôt que de lignes est d'offrir un meilleur taux de compression des données stockées.*

42. *L'avantage de l'utilisation de colonnes plutôt que de lignes est d'offrir de meilleures performances lors de la lecture de tous les enregistrements d'une table.*

43. *Une base HBase peut servir d'input/output de MapReduce (Hadoop)*

44. *Une base HBase peut servir de fichiers avec GFS (Google File System)*

45. *Une base de données orientée graphe stocke deux collections d'agrégats appelés nœuds et arêtes.*

46. *La suppression d'un nœud dans une base de données orientée graphe implique la suppression de toutes les relations partant et arrivant sur ce nœud.*

47. *Il est impossible de stocker une liste de personnes dans une base de données orientée graphe*

48. *SPARQL est un langage de requêtes générique permettant d'interroger n'importe quelle base de données NoSQL.*

49. *Gremlin est un langage de requêtes générique permettant de décrire des traversées de graphe.*

50. *Neo4j supporte les transactions ACID.*

51. *Les bases de données orientée graphe sont très adaptée pour le sharding.*

52. *OrientDB offre la possibilité d'utiliser le sharding de données.*

53. *Une approche pessimiste de la consistence des données consiste à se limiter à un serveur unique pour le stockage des données.*

54. *Garantir la consistence de lecture empêchera tout conflit de type write-write.*

55. *Garantir la consistence de mise à jour empêchera tout conflit de type read-write.*

56. *Garantir la consistence de réplication est impossible avec un système peer-to-peer.*

57. *Si mes données sont répliquées sur quatre nœuds, avec $W = 2$, il suffit de lire deux nœuds pour lire l'information la plus à jour.*

58. *Si mes données sont répliquées sur quatre nœuds, il suffit d'impliquer $W = 2$ nœuds dans l'écriture pour assurer une consistence des données.*

59. *L'utilisation d'un timestamp comme version stamp est moins lourd à déployer que d'utiliser un GUID (Globally Unique Identifier).*

60. *L'utilisation d'un GUID (Globally Unique Identifier) comme version stamp est moins lourd à déployer que d'utiliser un timestamp.*

61. *L'utilisation d'un GUID (Globally Unique Identifier) comme version stamp permet de retrouver la version la plus récente d'une donnée.*

62. *Le write optimiste est très cher à implémenter dans un modèle clé-valeur.*

63. *Dans une base de données orientée colonnes, les données transitent par plusieurs espaces de stockage avant leur destination finale permanent.*

64. *Il est possible d'utiliser de la réplication master-slave avec une base de données orientée graphe pour rendre les lectures plus performantes.*

65. *Les bases de données orientée documents permettent d'effectuer des transactions atomiques au niveau d'un document.*

66. *Les bases de données orientée documents permettent d'effectuer des transactions atomiques au niveau d'une collection.*

67. *On peut changer le modèle d'une base de données NoSQL entre clés-valeurs, colonnes et documents, tout en garantissant exactement le même ensemble de propriétés.*

68. *Le passage vers le NoSQL permet de se passer des ORMs.*

69. *Le NoSQL est très adapté pour stocker des données très uniformes.*

70. *Le passage du relationnel au NoSQL rend les calculs à effectuer sur les données plus lents suite à un éventuel cout de transfert des données au sein du cluster.*

71. *L'opération Map s'applique à une collection de documents et renvoie un collection modifiée.*

72. *L'opération Reduce produit un résultat unique.*

73. *ElasticSearch est une base de données NoSQL.*

74. *ElasticSearch possède des caractéristiques similaires aux bases de données NoSQL.*

75. *Une migration de données en relationnel ou en NoSQL implique toujours un stockage physique de données redondantes.*

Bibliographie

[1] Sébastien Combéfis. *NoSQL - Séance 2 : Modèle clé-valeur - Riak, memcached, Redis.* Ecole Centrale des Arts et Métiers, Bruxelles, 2017.