

Hibernate vs. the Cloud Computing

ARMAGEDDON



©1998 Fox and its related entities. Photo/Merrick Morton

www.charlotte.com/justgo/movies/

Qui suis-je?

- Julien Dubois
- Co-auteur de « Spring par la pratique »
- Ancien de SpringSource
- Directeur du consulting chez Ippon Technologies
- Suivez-moi sur Twitter : @juliendubois



<http://creativecommons.org/licenses/by-nc-nd/3.0/>

Nous allons parler de

- Bases de données relationnelles
- Hibernate
- Cache
- NoSQL

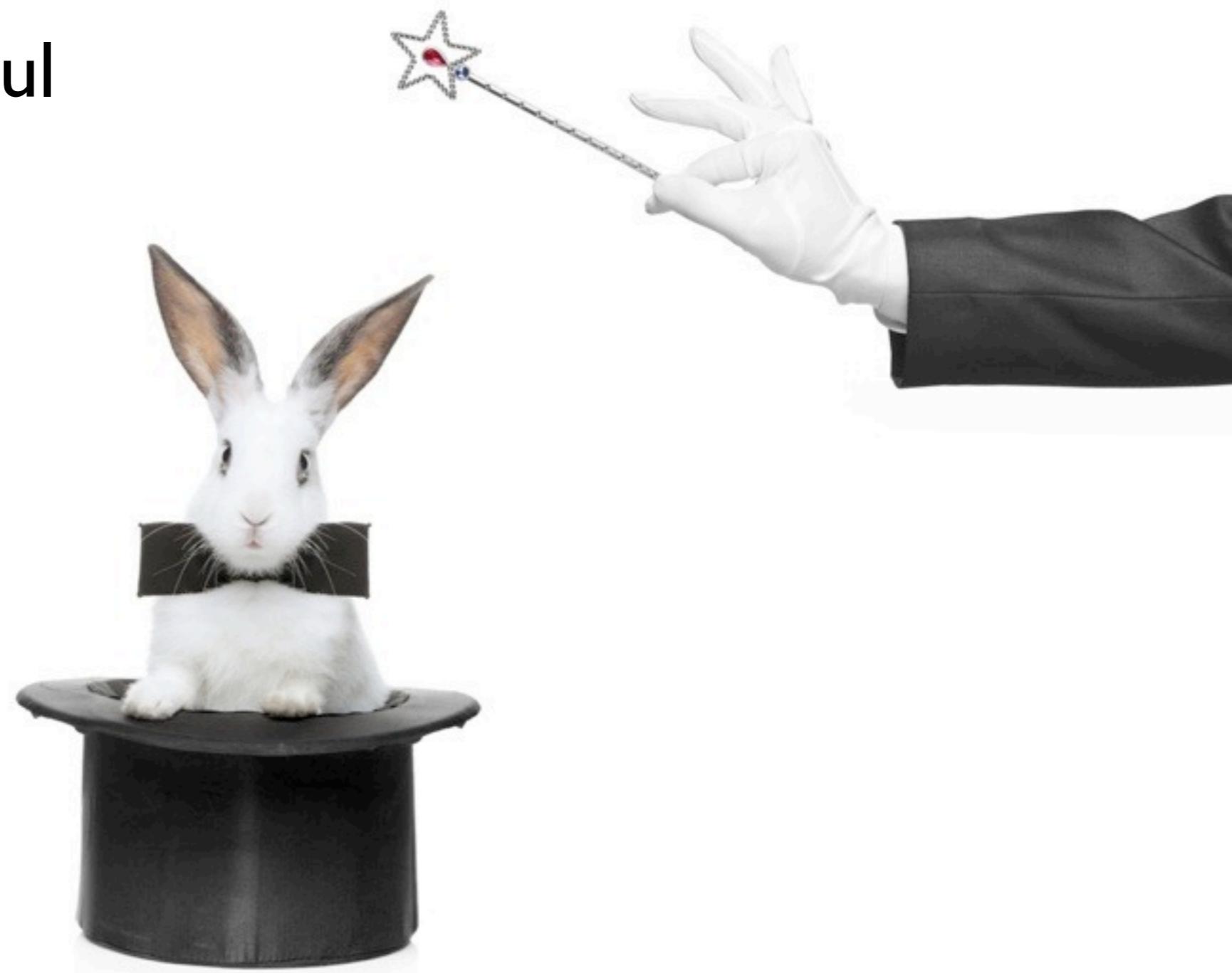


Commençons par deux idées fausses...



Idée #1: Hibernate est magique

- Il optimise tout seul les requêtes SQL
- Plus besoin d'être compétent!



Idée #2: Le cloud est magique

- Le cloud permet d'ajouter automatiquement des machines en fonction des besoins
- Il permet de tenir ainsi les pics de charge sans rien avoir à faire



La réalité est différente

- Ne croyez pas 01 Informatique
 - Votre base de données ne monte pas en charge
 - Et Hibernate n'y peut rien...



Les données sont votre problème

- Dans 90% des applications, la base de données est le principal point de contention
- Ces données sont stockées dans une base relationnelle :
 - Transactions
 - Sécurité
 - Intégrité
- Mais ces qualités ont un coût : vous ne pouvez pas monter en charge facilement



Exemple: quels amis viennent au BreizhCamp?

Problème

- Vous avez une table «utilisateurs» avec 1 million de lignes
- Chaque utilisateur est ami avec un certain nombre d'autres utilisateurs (many-to-many)
- Chaque utilisateur peut aller à des événements (many-to-many)
- Quand un utilisateur va sur la page d'un événement, on veut lui présenter la liste de ses amis qui y vont

Solution «Oracle simple»

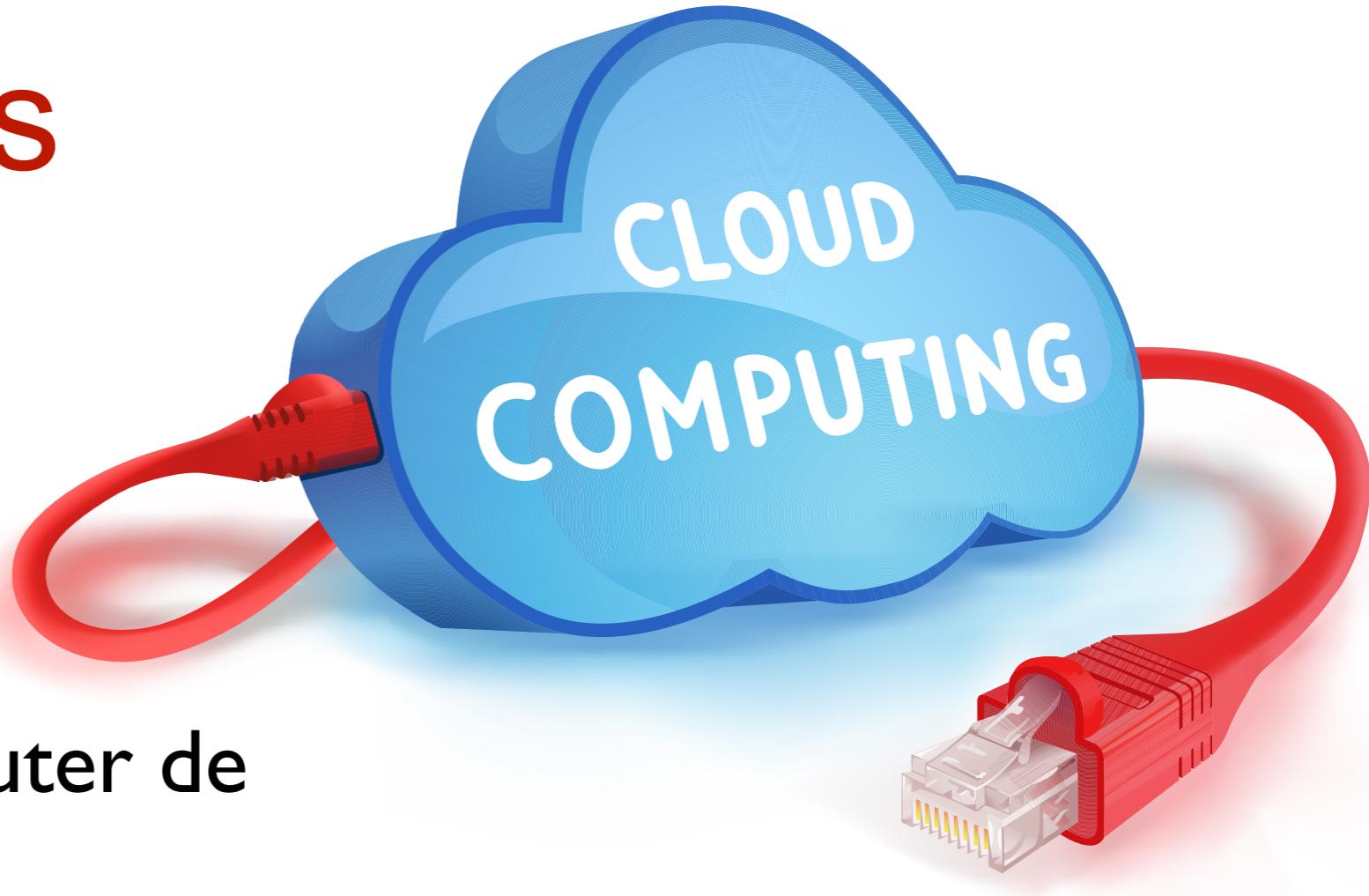
- Deux jointures: une pour sélectionner l'événement, une autre pour sélectionner l'utilisateur en cours

Que penser de cette solution?

- Très sécurisée: transactions, intégrité...
- Très simple à mettre en place avec Hibernate
- Ne monte pas en charge si on a beaucoup d'utilisateurs dans la base
- Va poser problème s'il y a beaucoup d'écritures
- Ne monte pas en charge si on a beaucoup de lectures
- Tout est sur la même machine (Single Point of Failure)

Vos données dans les nuages

- Le problème, c'est qu'une base de données relationnelle n'est pas scalable
 - On ne peut pas rajouter de nœuds « à chaud »
 - Sa scalabilité n'est pas du tout linéaire : pour maintenir ses transactions et son intégrité, elle est forcée de locker des ressources
 - Sur une vraie base de données (Oracle), ce lock est au niveau d'une ligne



Cela a même été prouvé

- Le théorème de CAP (ou théorème de Brewer)
- Votre système ne peut avoir que 2 des 3 caractéristiques suivantes
 - Cohérent (Consistent)
 - Disponible (Available)
 - Tolérant aux pannes réseau (Partition Tolerance)
- Conséquence : votre base de données relationnelle va mal monter en charge

Mais que fait Oracle?

- 1ère solution : faire monter en charge la machine
- Partitionning
 - Une table peut être découpée sur plusieurs disques durs
 - Réduit les problèmes d'I/O
 - Ne résout pas les problèmes de crash machine
- Cher
 - Option payante
 - Matériel coûteux



Application sur notre exemple

Problème

- Quels sont mes amis qui viennent au JUG?

Solution Oracle «partitionning»

- On utilise le champs date pour l'événement, et on partitionne par mois
- Réduction du nombre de lignes requêtées: on ne fait les requêtes (lecture/écriture) que sur le mois en cours

Que penser de cette solution?

- Mêmes qualités:
- Très sécurisée: transactions, intégrité...
- Très simple à mettre en place avec Hibernate
- Peut résoudre les problèmes d'IO si les données le permettent... Mais ce n'est pas vraiment le cas de notre exemple!
- Tout est sur la même machine: Single Point of Failure (SPoF)

Mais que fait Oracle? (2)

- 2ème solution : Oracle RAC
- Permet de monter en charge « linéairement » d'après la publicité: résout le problème du crash machine
- En fait la base locke toujours les ressources en écriture
 - Pose problème si vous avez beaucoup d'écritures (de toute manière, c'est rarement la lecture qui pose problème)
 - Pose des problèmes de latence réseau : géo-cluster non recommandé
- Très cher
 - Option payante très coûteuse
 - Matériel (dont réseau) très coûteux



Application sur notre exemple

Problème

- Quels sont mes amis qui viennent au JUG?

Solution Oracle «RAC»

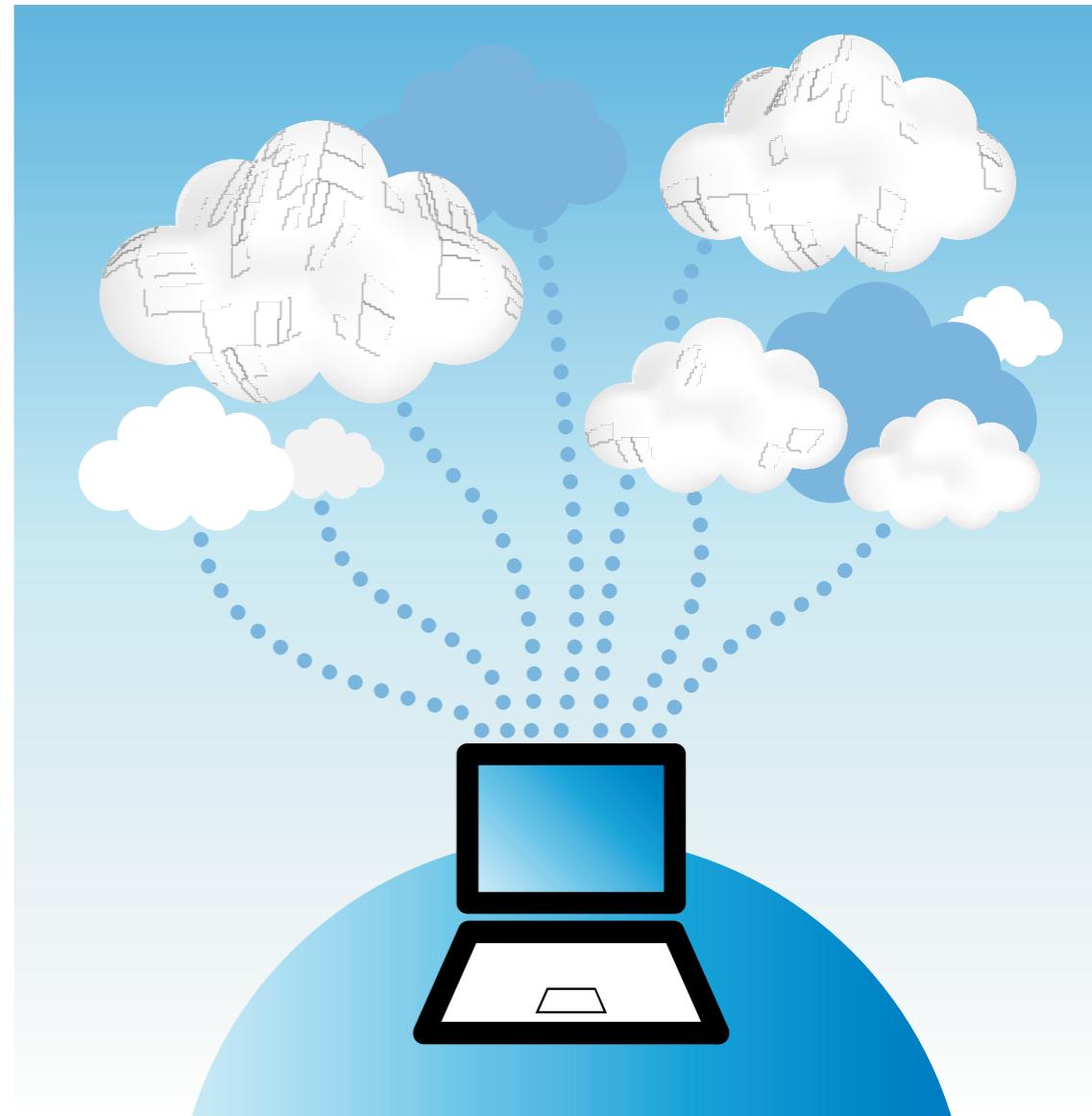
- Même requête qu'au début, avec deux jointures
- Cette requête est exécutée sur **l'un des noeuds** de la grille

Que penser de cette solution?

- Mêmes qualités que précédemment...
- Va résoudre certains problèmes en lecture: les requêtes sont réparties sur chaque noeud
- Plus de Single Point of Failure (SPoF)
- Mais toujours un problème en écriture...

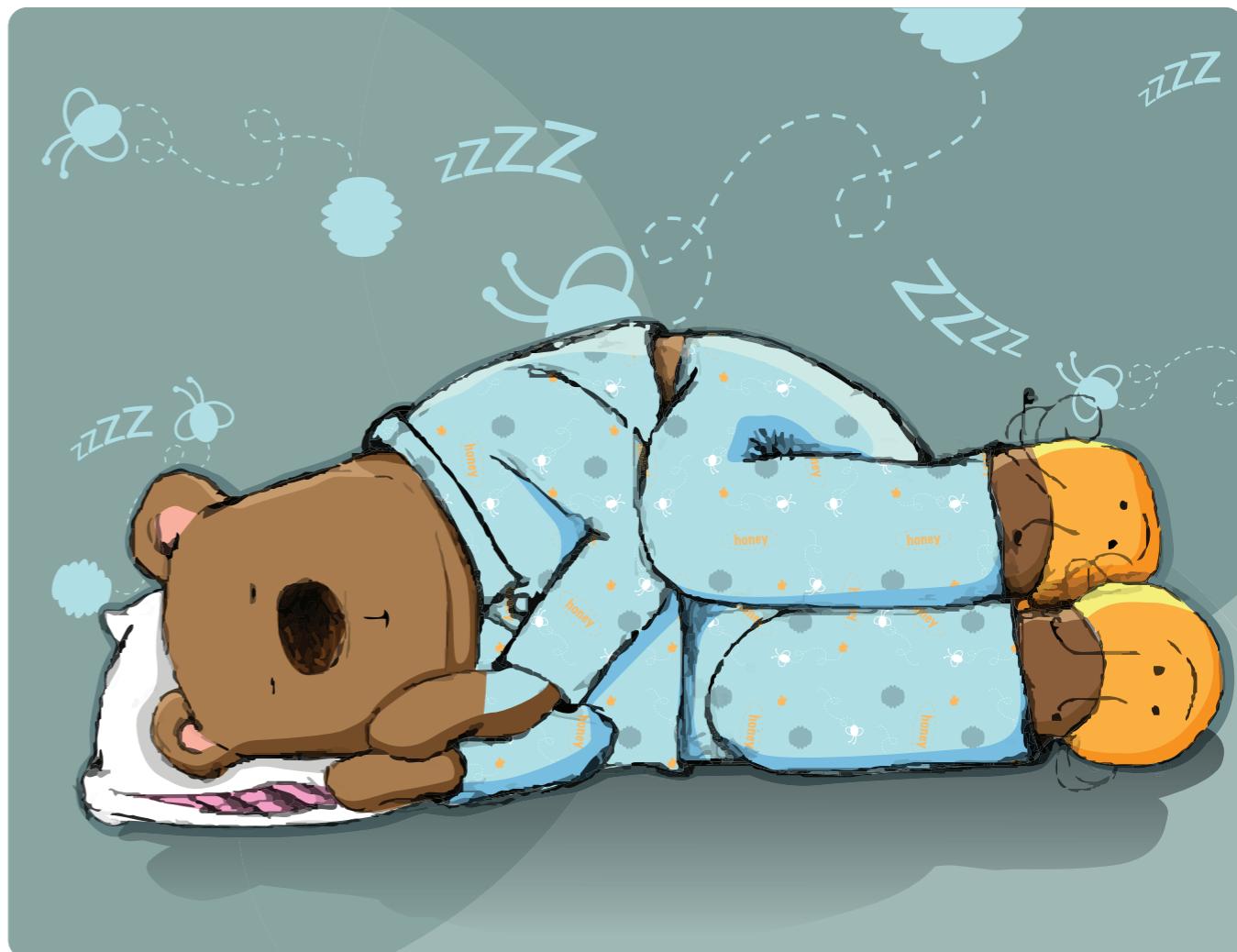
On m'aurait menti?

- Oui, votre application Java monte en cluster facilement, « dans le cloud »
 - Tant que vous utilisez des « sticky sessions »
 - Ou que vous stockez votre état ailleurs (cookie ou base de données?)
- Non, votre base de données ne monte pas en cluster
 - Or, c'était déjà probablement elle qui était le point de contention
- En conclusion, le « Cloud » ne vous sert pas à grand chose



Et Hibernate?

- Hibernate s'appuie sur les SGBDR
 - Il souffre donc des mêmes problèmes de scalabilité
 - Il faut même le configurer pour qu'il utilise correctement leurs fonctionnalités avancées (partitionning)
- Donc, votre application Hibernate ne devrait pas monter en charge

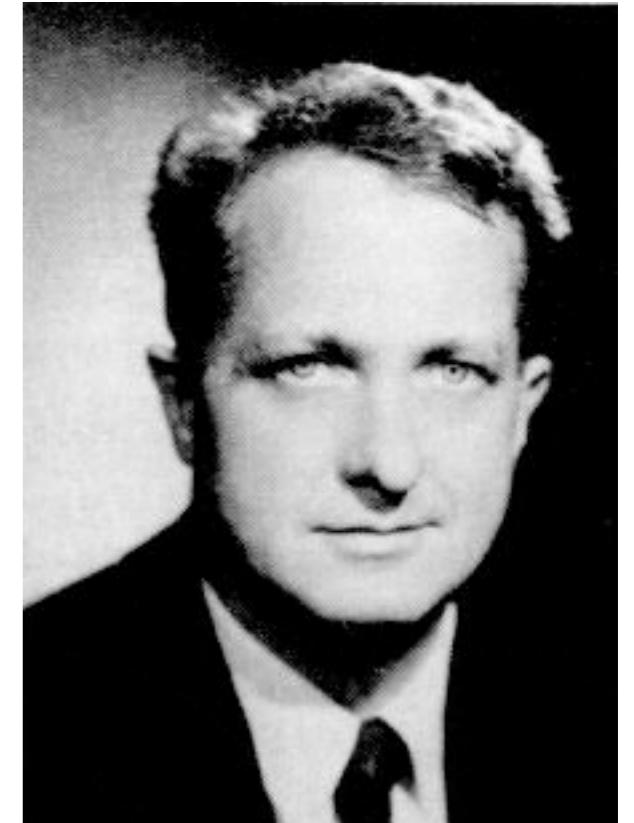


Une solution: le cache de 2ème niveau

- Hibernate possède un système de cache
 - Cache de 1er niveau : la transaction en cours
 - Cache de 2ème niveau : les données en base
- Ce cache de deuxième niveau permet d'éviter les requêtes en lecture sur la base
 - Tant que les requêtes sont simples (lecture d'une ligne en utilisant sa PK)
 - Ou qu'on utilise Hibernate de manière intelligente : faire $n+1$ requêtes sur le cache peut être plus performant qu'un outer join
 - Et que les données sont bien réparties...

La loi de Zipf

- Initialement une loi sur les mots utilisés
 - Certains sont beaucoup plus courants que d'autres
 - D'où une distribution très particulière des données
- Empiriquement, on retrouve le même phénomène dans de nombreuses applications
 - Une petite partie des données est tout le temps utilisée
 - Le reste n'est presque jamais lu
- Si votre application suit cette loi, votre cache sera très performant



Georges Kingsley Zipf
(1902-1950)

Application sur notre exemple

Problème

- Quels sont mes amis qui viennent au JUG?

Solution «Cache»

- Plusieurs solutions possibles, en voici une
- Mettre les participants d'un événement en cache
- Mettre les amis de l'utilisateur en cours en cache
- Faire la jointure en Java
- ... et stocker le résultat en cache!

Que penser de cette solution?

- Mêmes qualités que précédemment...
- Résout beaucoup de problèmes de lecture (à condition de suivre la loi de Zipf): on stocke les résultats en cache!
- Mais on ajoute des problèmes de désynchronisation du cache...
- On a toujours un problème en écriture
- Et il y a à nouveau un SPoF

Le cache distribué

- Ce cache peut être distribué sur un cluster
 - Ehcache (simple)
 - Terracotta, Coherence, Gemfire : produits commerciaux, hauts de gamme
- Réduit considérablement les problèmes de désynchronisation du cache
- Cela résout les problèmes de montée en charge d'applications avec beaucoup de lectures
 - Solution concurrente d'Oracle RAC, en moins cher, plus performant, plus intelligent

Le write-behind

- La plupart des caches distribués (Terracotta, Coherence) sont transactionnels
 - On committe dans le cache, qui se charge ensuite de gérer la base de données
- Les données sont insérées en base par batch, de manière asynchrone
 - Plus de performance
 - Moins de lock



Application sur notre exemple

Problème

- Quels sont mes amis qui viennent au JUG?

Solution «Cache distribué»

- Idem que pour un cache «simple»
- Mais on peut avoir plusieurs serveurs en parallèle
- En même faire un batch en parallèle, qui utilise le même cache

Que penser de cette solution?

- Mêmes qualités que précédemment...
- Résout quasiment tous les problèmes en lecture, et certains problèmes en écriture
- Mais à certaines conditions:
 - Acheter une solution de clustering (prix raisonnable par rapport à Oracle RAC)
 - Ajouter du monitoring (pour éviter le syndrome du « split brain »)

Et si on cassait
tout pour
reprendre à zéro?



Le NoSQL

- Derrière ce terme se cachent de nombreuses solutions et concepts différents
- Nous allons nous focaliser sur Apache Cassandra
 - L'une des solutions les plus populaires



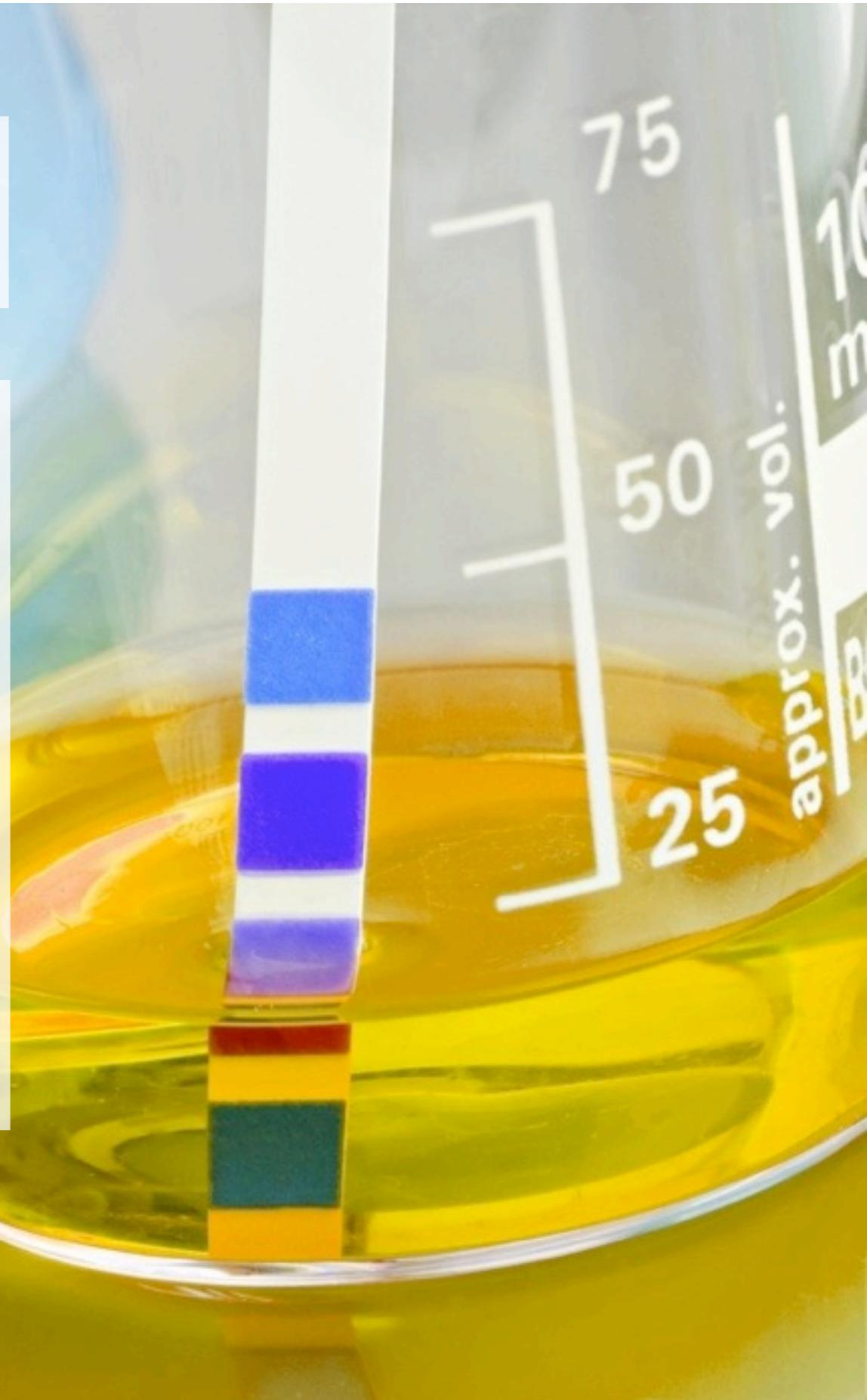
Cassandra

- Cassandra est une base de données distribuée, orientée colonne
- Cassandra n'a pas de schéma
- En fait, il n'a ni transaction, ni intégrité référentielle
- Pour reprendre le théorème de CAP: Cassandra est hautement disponible, tolérant aux pannes réseau, mais pas toujours cohérent
- Il peut donc être scalable, sans single point of failure (spof)
- Il est particulièrement efficace en écriture



BASE

- Encore un nouvel acronyme amusant
- Basically Available, Soft-state, Eventually consistent
- Opposé de ACID
- Quizz : que veut dire ACID ?



«Eventual Consistency»

- Comment ça, pas «coherent»?
- « Eventual » est un faux ami en français : cohérence « au final »
- A un moment donné la base n'est peut-être pas cohérente, mais à terme elle le sera
- En fait Cassandra vous donne le choix entre :
 - Une application très cohérente, mais lente en écriture
 - Une application très rapide en écriture, mais avec des risques au niveau de la cohérence
 - Les valeurs par défaut vous proposent une solution très rapide, avec des risques très limités

Exemple typique

- I. Vous écrivez sur un nœud
2. Cette écriture met du temps à se répliquer sur un deuxième nœud (on parle en millisecondes)
3. Quelqu'un lit sur ce deuxième nœud, en préférant la rapidité à la consistance (c'est son choix)
4. Cette personne obtient donc une donnée ancienne
5. En background, Cassandra valide la donnée et répare cette erreur

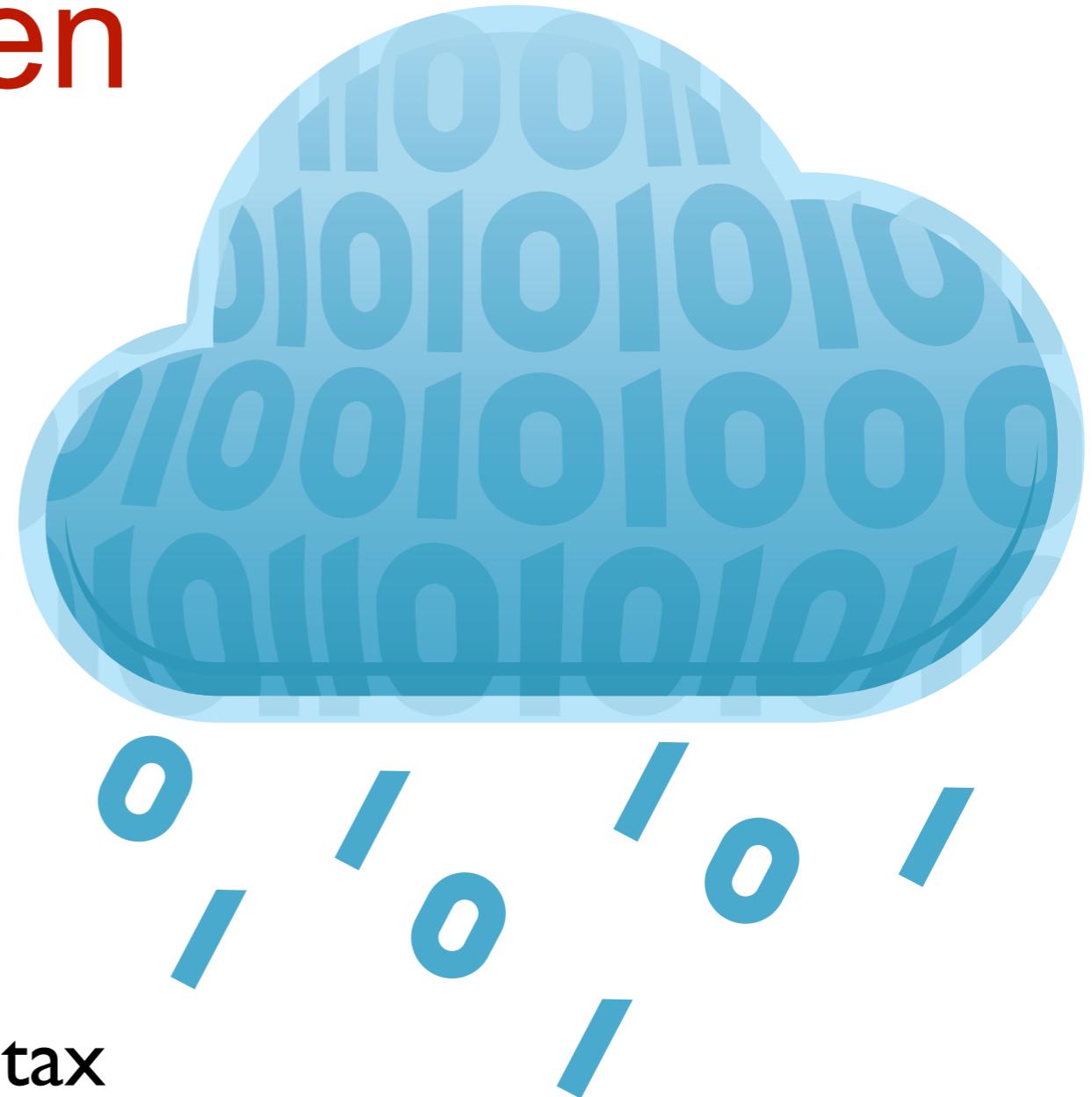
Mais que fait l'HADOPI?



- Cassandra fait du P2P entre ses différents nœuds
- On peut facilement en rajouter ou en enlever un
- Les données sont réparties entre ces nœuds automatiquement
- Mais si on veut une « bonne » répartition, les choses deviennent plus compliquées (ce n'est pas magique non plus)

Les problèmes en production

- La qualité des données :
 - Monitoring & production ne sont pas au même niveau que sous Oracle
 - Mais les outils sont nettement plus simples à utiliser
 - Offre commerciale : DataStax
 - Il peut y avoir des données non consistantes à un moment donné
 - Il n'y a ni transaction, ni intégrité référentielle
 - Tout dépend de ce que vous faites : cela n'est pas nécessairement grave



Les problèmes en développement

- Une manière très différente de gérer ses données
- Pas d'API de haut niveau (Hibernate), ni même d'API générique (JDBC)
- Il faut donc coder des DAOs à la main de manière assez fastidieuse, comme avant JDBC (bon courage)
- Pas de lazy-loading, etc : gros retour en arrière



Exemple de code

```
ColumnSlice<String, String> result =  
    createSliceQuery("MY_KEYSPACE ", stringSerializer,  
    stringSerializer, stringSerializer)  
        .setColumnFamily("UTILISATEURS")  
        .setKey(email)  
        .setColumnNames(  
            "first_name",  
            "last_name",  
            "last_login")  
        .execute()  
        .get();
```

Application sur notre exemple

Problème

- Quels sont mes amis qui viennent au JUG?

Solution «Cassandra»

- Faire une «Column Family» qui pour un utilisateur lui permet de retrouver tous ses amis
- Faire une autre « Column Family» qui stocke tous les utilisateurs qui participent à un événement
- Bien mélanger...

Que penser de cette solution?

- Plus aucune des qualités précédentes: transactions, intégrité, facilité de développement...
- Les données sont directement insérées dans un format proche des besoins métier
- Résout les problèmes en lecture (mais un cache reste le bienvenu)
- Résout les problèmes en écriture
- Pas de Single Point of Failure

Et Hibernate dans tout ça?

Hibernate OGM

- Utilisation des APIs d'Hibernate sur des bases NoSQL
- Officiel depuis aujourd'hui (17/06/2011)



Agissez avec nous : www.fne.asso.fr

Kundera

- <http://code.google.com/p/kundera/>
- Permet de faire du JPA directement
 - Spécifique Cassandra à l'origine
 - HBase et MongoDB en alpha

Mais il y a des limitations

- Il reste difficile avec ces outils d'utiliser pleinement les possibilités de Cassandra
 - Gestion des «wide rows» par exemple
- Ces outils sont encore très jeunes
 - Essentiellement du CRUD pour l'instant

Quizz

- Parmi les solutions étudiées, laquelle vous paraît la plus pertinente pour notre exemple?
 - Oracle «simple»
 - Oracle «partitionning»
 - Oracle «RAC»
 - Cache
 - Cache distribué
 - Cassandra
 - Une autre solution?

Conclusion

- Ne laissez pas tomber votre base de données relationnelle
 - Transactions, intégrité, sécurité
 - Largement suffisante pour des besoins normaux
 - Avec des outils inégalés en productivité (Hibernate)
- Travaillez sur le cache, et en particulier le cache de 2nd niveau d'Hibernate
- Le NoSQL sert à des besoins spécifiques
 - Certaines solutions (Cassandra) sont très bien adaptées au Cloud
 - A terme vous pourrez y accéder avec Hibernate/JPA

Questions? Réponses!

- Des questions sur le contenu de cette présentation ?
- Des ajouts ?
- Des commentaires ?
- N'hésitez pas à échanger et partager sur Twitter : @juliendubois

ippon
TECHNOLOGIES



<http://creativecommons.org/licenses/by-nc-nd/3.0/>