CONSERVATOIRE NATIONAL DES ARTS ET METIERS

le cnam
Midi-Pyrénées

IPST-CNAM

Formation Ingénieur Informatique
option systèmes d'information

ENG221 Information et communication pour l'ingénieur

Bases de données graphes : comparaison de NEO4J et OrientDB

Par Nicolas Vergnes

Soutenu en mai 2015

Sommaire

Index des figures

Index des tableaux

1 Introduction et définitions

Ce document compare deux SGBD : Neo4J version 2.1.7 et OrientDB 2.0.4.

Pourquoi comparer des SGBD ?

Lors de la construction d'une brique du SI (Figure 1), le besoin est d'abord modélisé en processus métier. La spécification des données intervient juste après. Ensuite, les interactions entre les différents états des données (fonctions) sont définies, ainsi que l'architecture applicative puis technique (couche infrastructure).

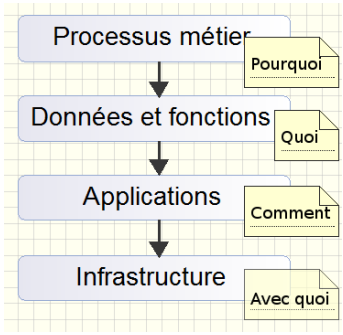


Figure 1: Les couches du SI

Les choix des données, de leur structuration et d'un SGBD sont des décisions cruciales car ce sont les fondations sur lesquelles les couches suivantes se construiront pour répondre au besoin métier. C'est pourquoi, un mauvais choix de SGBD engendrera de lourdes conséquences sur le résultat final. Si l'on s'en rend compte trop tardivement, migrer vers un autre modèle de données et un autre SGBD sera risqué et éprouvant en terme de conception, développement et validation.

Ce document a donc pour objectifs de présenter Neo4J et OrientDB puis de fournir les clefs qui faciliteront leur adoption en les comparant dans les différentes couches du SI (Figure 1).

Après avoir défini les bases de données NoSQL et graphes, Neo4J et OrientDB seront introduits l'un après l'autre. En suivant la logique de la Figure 1, leur comparaison traitera de la couche des données et fonctions, puis des différents types d'architectures possibles. La partie opérationnelle et les performances seront ensuite examinées. Enfin, une synthèse exposera les avantages et les inconvénients de ces 2 produits, disséminés dans une constellation de SGBD divers et variés.

1.1 Histoire du NoSQL

Basées sur un modèle de données structuré et sur l'algèbre relationnelle, les bases de données relationnelles existent depuis 1970 et sont encore à ce jour les plus utilisées dans le monde. De très rares SGBD relationnels n'utilisent pas le langage SQL, comme le NoSQL crée par Strozzi[1].

Les bases de données non relationnelles existent depuis les années 60 mais leur population a sérieusement augmenté dans les années 2000 avec l'essor d'Internet et des réseaux sociaux. En 2004, Google démarre ses projets BigTable, GFS et *MapReduce*. A partir de ce moment, tout s'accélère : le monde commence à s'intéresser sérieusement aux bases de données non relationnelles.

Les SGBD relationnels ont montré leurs limites avec des bases de très grande taille, notamment sur les performances et la difficulté de maintenir des modèles de données rigides. En effet, suivant le besoin, *ACID*, données structurées, jointures et transactions ne sont pas nécessaires ; ils deviennent alors des fardeaux. De plus, les SGBD distribués, prévus pour gérer ces nouvelles volumétries de données, vont préférer la résistance au morcellement et vont devoir renoncer à l'une des 2 autres contraintes du théorème de CAP.

Le terme NoSQL (Not SQL) est réutilisé pour catégoriser ces multiples SGBD non relationnels.

> *« 2009: Eric Evans [...] reintroduces the term NoSQL as specifically referring to non-relational databases. Strozzi has since said that the term should really be NoREL, [...] »* [2]

1"NoSQL Relational Database Management System: Commands at a Glance."
2"NoSQL or NoREL? A Short Account of Taxonomic Development - DATAVERSITY."

Nous savons maintenant que Strozzi avait raison de vouloir appeler « NoREL » des bases non relationnelles car plusieurs SGBD NoSQL supportent maintenant la syntaxe SQL. Le sigle NoSQL a évolué en Not Only SQL.

On dénombre à ce jour 150 SGBD NoSQL[3]. Ils peuvent être regroupés par la volumétrie supportée et la complexité des données ou en fonction de leur choix de contraintes du théorème de CAP mais ils sont généralement catégorisés en 4 types de modèles de données : clef-valeur, colonnes, document et graphes. Les graphes sont actuellement le type de modèle le plus structurant dans le NoSQL.

1.2 Les bases de données orientées graphes

Les bases de données orientées graphes ont été inventées dans les années 80, après les bases hiérarchiques, relationnelles et réseaux. On pourrait d'ailleurs trouver des similitudes au niveau de la représentation des données entre les bases graphes et les bases réseaux mais la ressemblance s'arrête là. En effet, leurs implémentations sont totalement différentes.

Les bases de données orientées graphe se basent sur la théorie des graphes. Un nœud représente une ou plusieurs données tandis qu'une arrête représente une relation entre un couple de données. Un SGBD orienté graphe stocke et manipule des graphes composés de nœuds et de relations qui peuvent tous avoir des propriétés.

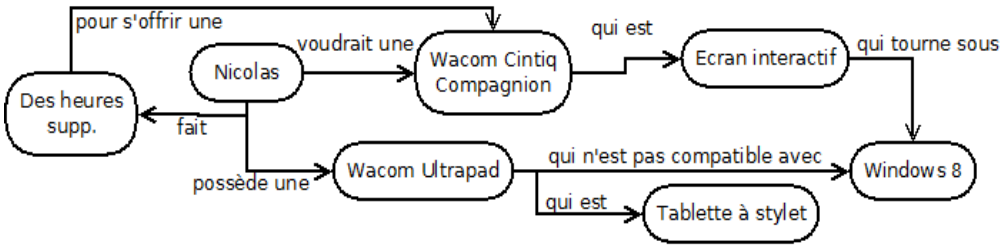


Figure 2: Exemple d'une représentation de données

Les SGBD relationnels sont plus performants lorsqu'il s'agit de retourner toutes les données d'une entité sans avoir à les filtrer et à réaliser des jointures coûteuses en temps, contrairement aux SGBD graphes qui sont conçus pour parcourir des modèles complexes à partir d'un ou plusieurs points de départ.

L'intérêt des graphes est de permettre la modélisation aisée d'une très grande quantité des données dans des modèles complexes et très connectés tout en offrant une structuration flexible, des opérations de modification et de parcours performantes. De plus, comme l'explique Alistair Jones dans une de ces vidéos [4] (minutes 16 à 22), la structure graphe simplifie le modèle relationnel. Cette simplification rend aussi sa lecture plus lisible qu'un modèle logique entité-relation.

Il existe 3 sous-familles de SBGD orientés graphe :

- Les graphes basés en mémoire sont les plus véloces en manipulation mais sont limités en taille par la quantité de mémoire centrale d'un serveur et sont volatiles.
- Les graphes basés sur disque sont eux limités par la taille des volumes locaux ou distants (plus de place mais de moins bonnes performances).
- Les graphes en cluster gèrent les plus grosses volumétries mais, du fait de leur architecture distribuée, sont par conséquent plus lents.

3“NOSQL Databases.”
4“Modelling with Graphs | SkillsCast | 31st August 2011.”

2 Neo4J (version 2.1.7)

2.1 Présentation

Neo4J est un SGBD orienté graphe de la famille NoSQL crée en 2000 par la société Neo Technology.

Neo4J est sous licence GPLv3 sous la dénomination de Community Edition. La version commerciale sera abordée dans le chapitre 2.3 Entreprises.

L'étude de Neo4J a été réalisée à partir de http://neo4j.com/developer/ et http://neo4j.com/docs. Afin d'éviter une surcharge de références dans chaque page, des liens et citations ne seront explicitement donnés que dans certains cas spécifiques.

2.2 Communauté

Le code source est disponible sur GitHub https://github.com/neo4j/neo4j où sont hébergées les deux éditions. Javadoc est utilisé pour générer une documentation riche à partir d'un code source normalisé est commenté[5]. En plus de la documentation du produit et de la Javadoc, une page dédiée aux développeurs fait une synthèse des concepts de base et des fonctionnalités mises en avant. L'outil de recherche de la documentation est particulièrement efficace comparé à celui d'OrientDB. En effet, il trie les résultats par pertinence et affiche un indicateur.

Neo Technology propose sur son site les liens vers les canaux communautaires. Neo4J bénéficie d'une communauté très active avec plus de 31 500 commits sur le code source depuis 2007 malgré une centaine de contributeurs (70 actifs depuis un an). On trouve 6700 questions sur StackOverflow, dont 1/3 porte sur le langage de requêtes, et 6500 sujets sur Google Groups. En revanche, sur 1100 problèmes ouverts depuis 2012, 1/3 sont encore à l'état ouvert.

Cependant, ces indicateurs n'informent pas sur l'évolution de Neo4J. Son adoption est-elle en perte de vitesse ? Son évolution explose-t-elle depuis peu ? Il serait bien difficile de répondre à la première question. En revanche, il est possible de connaître la vitesse d'évolution et de correction d'un logiciel par ses commits :

```
## Telechargement de l'historique des commits de
la branche master sur le repository
for p in `seq 1 886`;do  wget
"https://github.com/neo4j/neo4j/commits/master?
page="$p -O $p.in ;done
## Extraction des dates
egrep "authored " *.in |  awk -F"[><]" '{print
$3}' > dates.out
## Découpage des dates en 1er jour du  mois
cat dates.out | awk '{print $1" 1, "$3}' | while
read line;do echo "$line ";done > dates.month.out
## Comptage par mois et transformation en fichier
CSV ( timestamp pour tirage dans calc ;  dates ; nbre commmits dans le mois)
cat dates.month.out | while read line;do echo "` date -d\"$line\" '+%Y%m%d'`;` date -d\"$line\"
'+%d/%m/%Y'`;`egrep -c  "$line" dates.month.out`";done | sort -u > Neo4J_commits_count.csv
```

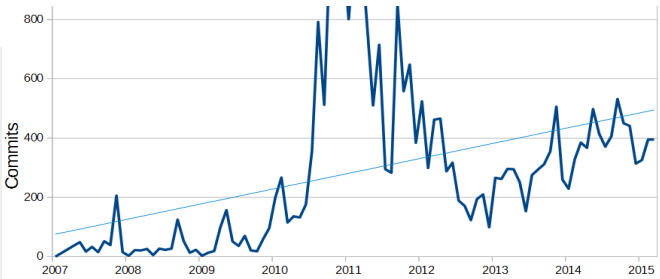


Figure 3: Neo4J : Evolution des commits depuis le début du dépôt GIT

La Figure 3 démontre une activité croissante sur le code source au fil des années.

5"30.5. Contributing Code to Neo4j - - The Neo4j Manual v2.1.7."

2.3 Entreprises

Une version commerciale distribuée sous licence propriétaire et AGPLv3 permet de bénéficier du support client et de fonctionnalités supplémentaires :

- haute disponibilité ;
- *scalabilité* horizontale en lecture ;
- sauvegarde à chaud ;
- surveillance améliorée ;
- système de cache amélioré ;
- outils de gestion d'administration.

Les quatre premières fonctionnalités sont primordiales en environnement de production. De cette manière, Neo Technology force l'utilisation de Neo4j en production par cette offre payante. Les types de licences (achat, location) ainsi que leur coût de maintenance ne sont pas publics mais il est fait mention sur Internet d'au moins 10K$ annuel pour du support sur des architectures de base, d'autres parlent de 24K$ pour une instance[6]. Afin d'inciter les très petites entreprises ayant un faible chiffre d'affaire « à mettre un pied » dans Neo4J, une formule Personal permet l'utilisation de la version Enterprise sans frais mais sans support.

En plus d'un grand nombre de ressources disponibles, tutoriels, livres, formation à distance, Neo Technology propose des formations en administration, modélisation de graphes et manipulation des données. Ils proposent aussi des services de consulting sur toutes les phases du cycle de vie d'un SI.

Le business model de Neo Technology est restrictif, non concurrentiel (par rapport à OrientDB) et plutôt sibyllin. Les questions sur les droits d'usage ne manquent pas dans les espaces communautaires.

Neo Technology compte 80 clients dont ebay, Cisco, Wallmart, HP.

2.4 Architecture générale

Neo4J est développé en Java et en Scala. Il utilise Java 7 au minimum. Il est supporté sur les OS Linux, HP UX et Windows mais les FS ext4 et ZFS sont recommandés pour supporter l'ensemble des fonctionnalités. Le SGBD fonctionne en *standalone* et en cluster master/slave.

La Figure 4 est une modification d'un graphique de la documentation. Elle permet de prendre connaissance des différentes briques qui composent Neo4J. La version Enterprise est un package de classes qui viennent hériter de celles ci-contre pour étendre les fonctionnalités.

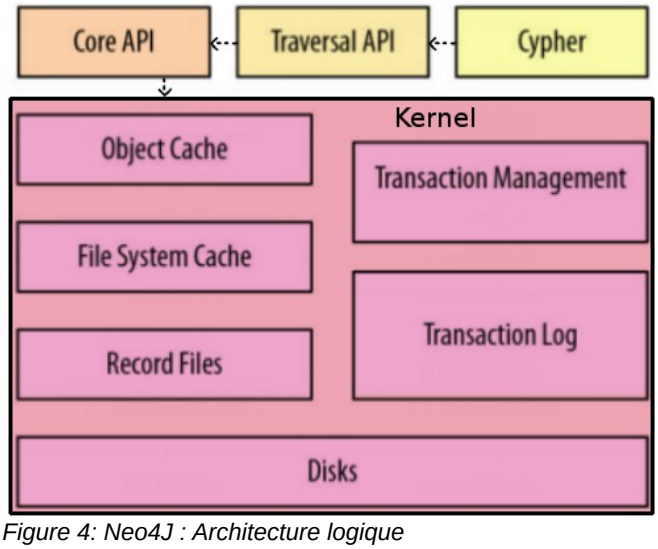


Figure 4: Neo4J : Architecture logique

6“(51) Neo4j - Google Groups.”

3 OrientDB (version 2.0.4)

3.1 Présentation

OrientDB est un SGBD multi-modèle crée en 2010 par la société Orient Technologies. Il supporte les modèles clef/valeur, document, graphe et objet.

OrientDB est sous licence Apache version 2 sous la dénomination de Community Edition. La version payante sera abordée dans le chapitre Entreprises. Deux autres versions ne seront pas étudiées : une version spécifique pour l'embarqué et celle pour les services de cloud computing fournis par des partenaires.

L'étude de OrientDB a été réalisée à partir de http://www.orientechnologies.com/docs/last/. Comme pour Neo4J, les liens et citations ne seront explicitement donnés que dans certains cas spécifiques.

3.2 Communauté

Le code source est disponible sur GitHub https://github.com/orientechnologies/orientdb. Orient Technologies utilise aussi Javadoc. Il fournit une documentation riche mais son moteur de recherche est moins performant que celui de Neo4J : pas de triage ni d'indicateur de pertinence. Comme sur Neo4J, le wiki de GitHub ne semble pas vraiment actif.

La communauté de développeurs d'OrientDB est moins nombreuse que celle de Neo4J avec 40 contributeurs actifs depuis un an mais elle est aussi beaucoup plus jeune que celle Neo4J qui est de 10 ans son aînée.

On trouve 500 questions sur StackOverflow et 6000 sujets sur Google Groups. L'indice de popularité de Google, Figure 5, vient appuyer l'idée que OrientDB est moins connu, a une plus petite communauté et ne tend pas à rattraper Neo4J.

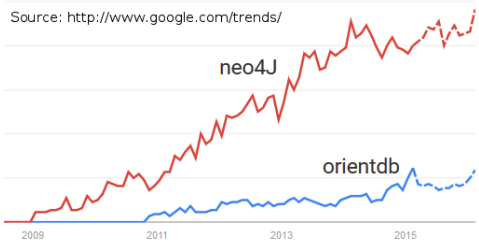


Figure 5: Comparaison des indices de popularité Google Trends

Sur 2700 incidents ouverts depuis décembre 2012, 13% sont des demandes d'amélioration et 7 % sont des bugs. Malgré une communauté de contributeurs plus grande, Neo4J a un pourcentage de bugs non corrigés plus élevé.

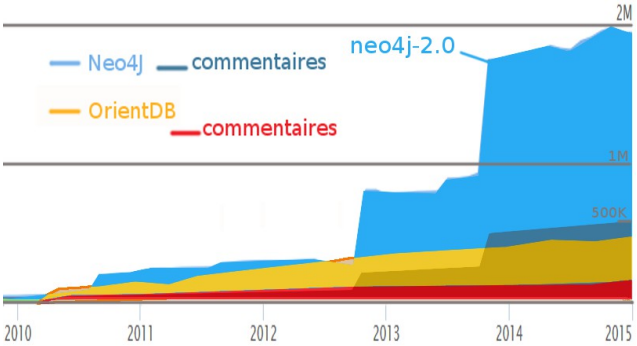


Figure 6: Nombre de lignes de code et de commentaires dans le code source (données utilisées:www.openhub.net)

D'après la Figure 6, OrientDB possède à ce jour 261 mille lignes de code source et évolue de façon linéaire, contrairement à Neo4J qui a doublé de taille avec sa version 2.0. Les contributeurs d'OrientDB maintiennent 4 fois moins de code que ceux de Neo4J.

3.3 Entreprises

La version commerciale OrientDB Enterprise Edition sous licence propriétaire fournit du support ainsi que les fonctionnalités suivantes :

- un profileur de requêtes ;
- une IHM pour administrer les clusters ;
- des sauvegardes à chaud ;
- du stockage et un outil d'analyse graphique pour un grand nombre d'indicateurs ;
- un service de supervision avec des alertes email et des déclenchements automatiques d'actions par des requêtes HTTP.

Contrairement à Neo Technology, Orient Technologies ne bloque pas les fonctionnalités primordiales en production dans la Comunity Edition, telles que de haute disponibilité et la sauvegarde sans coupure de service.

Les prix de la version Enterprise sont publiés sur le site internet, ils varient aux alentours de 2000 euros par nœud. Une formule pour développeurs intègre des prestations de consulting à distance. Les start-up bénéficient de 50 % de réduction sur les prix. En plus de la documentation et des tutoriels, une formation à distance gratuite est disponible. Des sessions de formation (payantes) à distance et sur site sont aussi proposées.

L'offre aux entreprises d'Orient Technology est plus complète, plus transparente, moins contraignante et moins chère que celle de Neo Technology.

OrientDB Technologies compte 60 clients dont Warner, Cisco, l'ONU, Verisign ou encore Lufthansa.

3.4 Architecture générale

OrientDB est développé en Java. Il est supporté sur les OS Linux (architecture ARM incluse), Windows, Solaris, HP-UX, AIX et Mac OS X et utilise au minimum Java 1.6. Il fonctionne sur un serveur *standalone* et en architecture distribuée multi-master.

Il est construit par un empilement des couches. La Figure 7 montre qu'OrientDB supporte les 3 sous-familles des SGBD orientés graphe vues dans §1.2 (p6). Elles sont les fondations et implémentent l'interface de stockage de données.

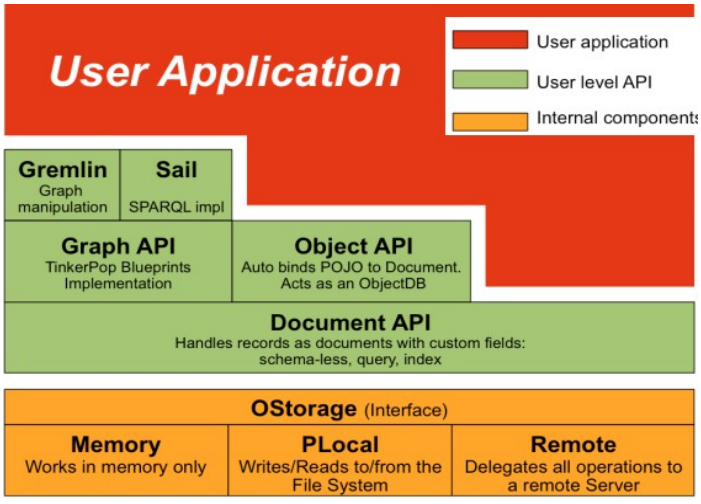


Figure 7: OrientDB : architecture logique

Cette figure met aussi en évidence le rôle central de la couche de gestion Document sur laquelle se reposent les autres modèles de données. Par défaut, les applications clientes peuvent attaquer n'importe quelle couche de données et ont accès à l'intégralité des informations stockées. Cette structure laisse entrevoir un risque sur l'intégrité des informations, comme par exemple la modification directe des données de graphes en passant par l'API Document : les fonctionnalités fournies dans les couches supérieures, comme le mécanisme transactionnel des nœuds et des arêtes des graphes, ne seraient alors pas utilisées. On se trouverait alors dans un état inconsistant.

4 Données et fonctions

4.1 Type et modèles de données

La différence majeure avec Neo4J est qu'OrientDB gère des modèles de données autres que les graphes : clef/valeur, document et objet. Cette étude n'a pas pour objectif de traiter ces 3 autres modèles, mais il est important de prendre en compte cet aspect-là lors d'un choix entre ces 2 applications.

Connaître le modèle de données passe par la définition des notions de nœud, arête, propriété, traversée, collection, classe et label.

Les *collections* d'objets regroupant des *nœuds* sémantiquement proches sont appelées *classes* sous OrientDB et *labels* sous Neo4J.

Ces nœuds sont mis en relation par des *arêtes*.

Les nœuds et les arêtes possèdent des *propriétés* de type *clef:valeur*. Ces valeurs peuvent être de plusieurs types. Dans une collection, les clefs varient d'un nœud à un autre ; par défaut, il n'y a pas de contraintes qui structurent les nœuds. Ces derniers ont quand même intérêt à partager une certaine quantité de clefs communes afin de donner un sens à leur regroupement.

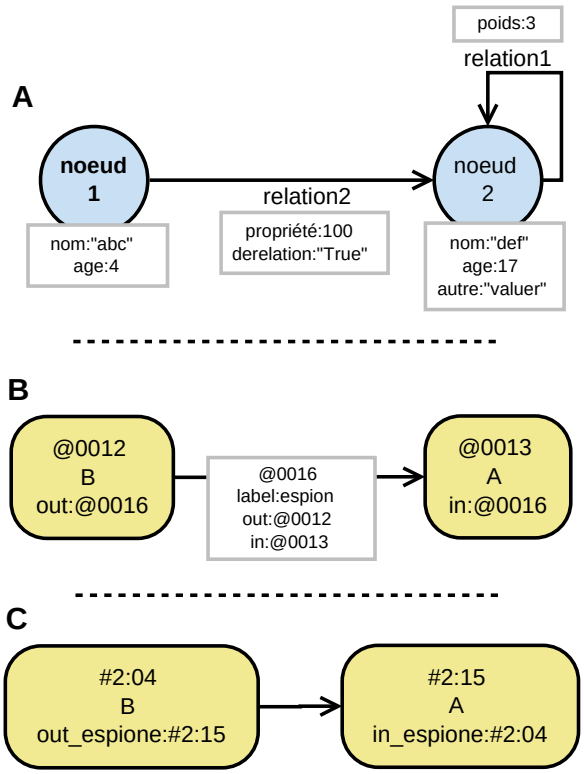


Figure 8: (A) Exemple d'un graphe. (B) Modélisation d'une relation avec une arête en tant qu'entité. (C) Arête sans propriété avec OrientDB

La Figure 8 graphe B détaille les liens entre les nœuds. Les nœuds stockent les adresses physiques de leurs relations en tant que propriétés. Les relations stockent les adresses physiques des nœuds qu'elles relient. Les relations 1→N, N→1 et N→M sont créées grâce à des listes d'adresses en propriétés. Le SGBD n'a plus qu'à naviguer d'adresse en adresse pour circuler dans le graphe. Cette traversée de nœud en nœud est réalisée très rapidement et sans opération complexe car il n'y a pas de jointure comme dans le modèle relationnel.

Neo4J a l'avantage de permettre la définition de plusieurs labels pour un même nœud.

Les classes de nœuds dans OrientDB étendent la classe V et les classes d'arêtes étendent la classe E. Les entrées possèdent un champ unique nommé Record ID ou RID, formé de la façon suivante :

#cluster-id:position-dans-le-cluster

Sachant gérer les modèles de données objets, les classes de nœuds des graphes sous OrientDB bénéficient de l'héritage.

Une optimisation peut être activée manuellement pour la création d'arêtes (Figure 8 graphe C) : si cette relation ne dispose pas de propriété alors l'arête n'est pas créée dans la classe E mais en tant que propriété dans les deux nœuds en relation. L'intérêt est de traverser les graphes plus rapidement mais l'inconvénient est que la relation n'est dans aucune classe d'arêtes et donc moins visible.

Sous OrientDB, un *cluster* est une partition d'une classe qui stocke les données. Il en existe un par défaut pour chaque classe. Il est possible de partitionner une classe sur plusieurs *clusters* et ainsi de repartir les données dans chacun d'eux. Le comportement d'un cluster est détaillé dans le §5.3.

Le Tableau 1 présente les différents types de propriétés gérés par Neo4J et OrientDB.

Type de propriété	Neo4J	OrientDB	Limites
Byte	Oui	Oui	[-128,127]
Short	Oui	Oui	[-32 768, 32 767]
Integer	Oui	Oui	[-2 147 483 648, +2 147 483 647]
Long	Oui	Oui	$[-2^{63}, +2^{63}-1]$
Float	Oui	Oui	$[2^{-149}, (2-2^{-23})*2^{127}]$
Double	Oui	Oui	$[2^{-1074}, (2-2^{-52})*2^{1023}]$
String et objets sérialisés	Oui	Oui	
Booléen	Oui	Oui	0, 1
JSON	Non	Oui	
XML	Non	Non	
Datetime , Date	Non	Oui	
Binaire, Objet	Non	Oui	
List, set, map	Non	Oui	41 000 000 éléments
Embedded (équivalent Oracle BFILE)	Non	Oui	
Link (Adresses des arêtes et des nœuds)	Non	Oui	
List , set, map de Embedded et Link	Non	Oui	41 000 000 éléments
BLOB, CLOB (Binary & Character Large OBject)	Non	Non	
Divers, Custom , BigDecimal, Transient	Non	Oui	

Tableau 1: Types de données supportés

Neo4J a des lacunes sur les types de données supportés. Il se limite en effet aux primitives Java. Ne pas supporter les imbrications de propriétés (JSON) et les dates pourra devenir rapidement problématique pour les requêtes Cypher. Cependant, il sera toujours possible de sérialiser ces types depuis une API externe.

OrientDB n'a pas ce problème et supporte une grande variété de types.

La taille limite des valeurs est indiquée dans le Tableau 5: Comparaison des limitations des données.

4.2 Langage

Gremlin était au départ le langage par défaut sur Neo4J mais Neo Technology a décidé de créer un nouveau langage, le Cypher, pour faciliter la prise en main, la lecture et l'écriture des requêtes.

La Figure 7 (p10) montre l'imbrication des différentes couches internes qui constituent OrientDB. L'API gérant les graphes implémente *TinkerPop* et repose sur la gestion du modèle de données de type document. Il est recommandé d'utiliser des pilotes qui passent par *TinkerPop* pour manipuler les graphes afin d'assurer la consistance des données, comme les commandes pseudo-SQL spécifiques aux graphes, Sail, Gremlin ou des pilotes externes, par exemple Bulbflow pour Python.

Les langages pseudo-SQL et Cypher vont être comparés dans les Tableau 2 et Tableau 3 à partir du graphe Figure 9.

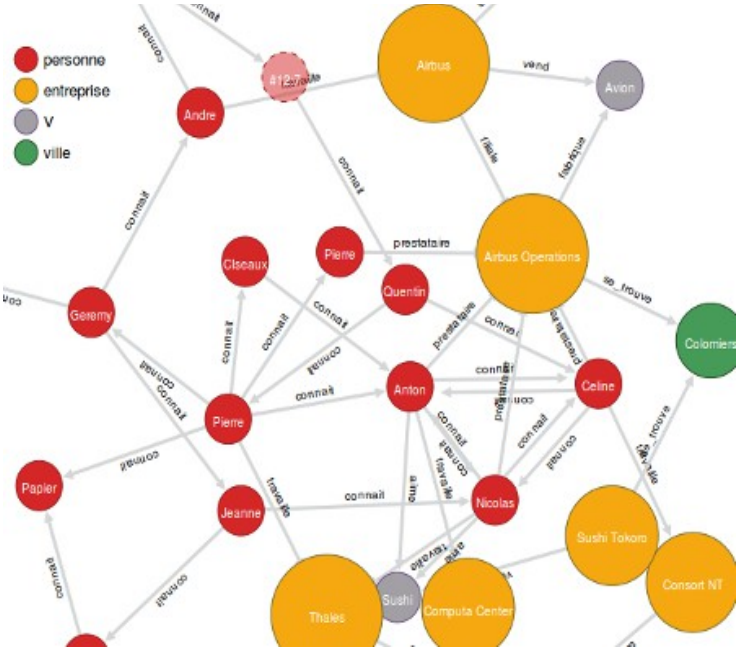


Figure 9: Représentation sous OrientDB du graphe utilisé pour la comparaison des langages

	Neo4J	OrientDB
Création d'une collection pour les nœuds	Effectuée lors d'une première utilisation d'un nœud labellisé	CREATE CLASS Personne EXTENDS V
Création d'une collection pour les arêtes	Effectuée lors d'une première utilisation d'une arête labellisée	CREATE CLASS connait EXTENDS E
Création d'un index	CREATE INDEX ON :Personne(nom)	CREATE INDEX I_perso ON Personne (nom)
Création d'une contrainte	CREATE CONSTRAINT ON (a:Personne) ASSERT a.nom IS UNIQUE	CREATE INDEX I_perso ON Personne (nom, prenom, date_naissance) UNIQUE
Suppression d'une collection	MATCH (n) REMOVE n: Personne Supprimer un label ne supprime pas le nœud associé.	DROP CLASS Personne Il faut supprimer son contenu avant de supprimer une classe
Renommer une collection	MATCH(n:Personne) SET n: Personne2 REMOVE n: Personne	ALTER CLASS Personne NAME Personne2
Définition de la méthode de routage d'une classe vers ses clusters	Pas de cluster avec neo4j	ALTER CLASS Entreprise CLUSTERSELECTION ROUND-ROBIN
Création d'une propriété structurée pour une classe	Fonctionnalité indisponible	> ALTER PROPERTY Personne.nom STRING > ALTER PROPERTY Personne.sexe REGEXP [H|F|A]

Tableau 2: Comparatif de Cypher et pseudo-SQL : Définition des données

	Neo4J	OrientDB
Création d'un nœud	CREATE (n:Entreprise { name : 'Sushi Tokoro', type : 'SARL', SIREN:'802748103' })	> CREATE VERTEX Entreprise CONTENT { "nom":"Sushi Tokoro", "type":"SARL", "SIREN":"802748103" } > CREATE VERTEX Entreprise SET nom="Airbus", type="SAS"
Création d'une relation	MATCH (c:Personne),(n:Personne) WHERE c.prenom = 'Celine' AND n.prenom = 'Nicolas' CREATE (c)-[r:connait]->(n)	CREATE EDGE connait FROM (SELECT FROM Personne WHERE prenom="Celine") TO (SELECT FROM Personne WHERE prenom="Nicolas")
Création d'une relation avec propriété	MATCH (p:Personne),(e:Entreprise) WHERE p.prenom='Nicolas' AND e.nom='Thales' CREATE (p)-[r:travaille {depuis:2012,fin:2014}]->(e)	CREATE EDGE travaille FROM (SELECT FROM Personne WHERE prenom="Nicolas") TO (SELECT FROM Entreprise WHERE nom="Thales") SET depuis=2012, fin=2014
Création d'un nœud si inexistant	MERGE (n:Personne { prenom:'Nicolas', age:34 }	Pas de commande prévue, voir avec contrainte d'unicité ou tester manuellement avant un ajout
Suppression d'un nœud	> MATCH (p:Personne { prenom:'Jean-Pierre'}) WHERE p.nom =~ 'Belma.*' DELETE p > MATCH (p:Decedee) DELETE p	> DELETE VERTEX #13:5 > DELETE VERTEX Entreprise WHERE nom.LEFT(5) = "Airbus" > DELETE VERTEX Personne WHERE in.@Class = "PersonneDecedee"
Suppression d'une relation	MATCH (n:Personne {prenom:'Marine'})-[r:travaille]->(e:Entrepise {nom:'Airbus'}) DELETE r	> DELETE EDGE #17:19 > DELETE EDGE travaille WHERE depuis < "1990" > DELETE EDGE FROM #17:15 TO #17:35
Suppression multiple	MATCH (n:Personne)-[r:connait]-(m:Personne) WHERE r.status='fauxami' OR r.date<="130801" DELETE r	DELETE EDGE connait FROM #10:20 TO #10:30 WHERE status = "fauxami" OR date <= "130801"
Ajout d'une propriété	MATCH (p:Personne {prenom:'Nicolas'} SET p.adresse = "2, rue des pinpons" RETURN p	UPDATE (SELECT FROM Personne WHERE prenom="nicolas") SET adresse = "2, rue des pinpons"
Modification d'une propriété	> MATCH (p:Personne {prenom:'Nicolas'} SET p.adresse = "2, rue des pinpons" > MATCH (n:Personne) WHERE n.salaire < 1300 WITH n ORDER BY n.salaire LIMIT 10 SET n.salaire = n.salaire +100	> UPDATE (SELECT FROM Personne WHERE prenom="Nicolas") SET adresse = "2, rue des pinpons" > UPDATE Personne INCREMENT salaire = 100 WHERE salaire <= 1300 LIMIT 10
Copie d'une propriété	MATCH (n {nom:'Nicolas'}),(a {nom:'Anton'}) SET a.salaire = n.salaire RETURN a,n	UPDATE (SELECT FROM Personne WHERE prenom="Anton") SET salaire = (SELECT salaire FROM Personne WHERE prenom="Nicolas")
Suppression d'une propriété	MATCH (p:Personne {prenom:'Nicolas'} SET p.adresse = NULL	UPDATE #12:0 REMOVE adresse
Ajout de plusieurs propriétés sur un noeud	MATCH (n:Personne {prenom:"Nicolas"}) SET n.sexe='M' SET n.age=34 RETURN n	UPDATE (SELECT FROM Personne WHERE prenom="nicolas") SET sexe='M', age=34
Suppression des données d'une collection	MATCH (v:Ville) DELETE v	TRUNCATE CLASS Ville
Sélection d'un membre	MATCH (p:Personne {prenom:'Nicolas'} RETURN p	SELECT FROM #15:5
Sélection des champs d'un membre	MATCH (p:Personne {prenom:'Nicolas'} RETURN p.nom, p.prenom, p.salaire	SELECT nom, prenom, salaire FROM #15:3
Sélection d'un ensemble dans une collection	MATCH (p:Personne {prenom:'Celine'}) WHERE p.nom =~ '.*defo.*' RETURN p	SELECT FROM Personne p WHERE p.prenom="Celine" AND p.nom LIKE '%defo%'
Sélection de tous les membres	MATCH (e:Entreprise) RETURN e	SELECT FROM Entreprise

Sélection avec tri inversé	MATCH (p:Personne) RETURN p ORDER BY p.anciennete, p.salaire DESC	SELECT FROM Personne ORDER BY anciennete,salaire DESC
Sélection limitée et tri	MATCH (p:Personne) RETURN p ORDER BY p.salaire LIMIT 20	SELECT FROM Personne ORDER BY salaire ASC LIMIT 20
Transformation des champs dans une sélection et tri	MATCH (n:Personne) WHERE n="Pierre" OR n="Jean" RETURN upper(p.nom) ORDER BY p.nom	SELECT nom.TOUPPERCASE(), adresse.rue.numero FROM [#15:5,#15:20] ORDER BY nom
Sélection des prédécesseurs d'une relation donnée	MATCH (a { name:'Airbus' })-[:Prestataire]->(p) RETURN p.prenom	SELECT in('prestataire').prenom FROM Entreprise WHERE nom="Airbus Operations"
Sélection des nœuds en relations directe avec un autre	MATCH (n { prenom:'Nicolas' })-[:Connait*1..4]-(p) RETURN p	SELECT FROM (TRAVERSE * FROM #12:9 WHILE $DEPTH <= 8) WHERE @class ='Personne'

Tableau 3: Comparatif de Cypher et pseudo-SQL : Manipulation des données

En étudiant la documentation de Cypher[7] et du pseudo-SQL[8], on constate un grand nombre de possibilités et chacun possède des avantages sur l'autre en terme de fonctionnalités de langage.

Les Tableau 2 et Tableau 3 montrent que Cypher est plus simple et permet d'écrire des requêtes de traversée plus facilement que le pseudo-SQL. En plus de cibler plus finement les données, Cypher permet de leur affecter des variables pour ensuite les ré-utiliser, ce qui rend les requêtes d'autant plus lisibles.

OrientDB offre des alias pour définir les classes et sait pointer directement vers une donnée lorsqu'on connaît son adresse physique. Avec le pseudo-SQL, on se retrouve rapidement avec un grand nombre de niveaux d'imbrication de sous-requêtes : la compréhension d'une requête est plus longue qu'avec Cypher. On pourrait penser que sa proximité avec le SQL est un avantage mais c'est aussi un choix dangereux car beaucoup de développeurs, formés aux SGBD relationnels, pourraient ne pas bien comprendre ce qu'ils manipulent réellement, continuer à développer « du SQL » et faire ainsi des erreurs de conception de structure ou de manipulation de données.

Mais le langage standard le plus puissant en terme de possibilités est Gremlin[9], et qui est par conséquent le plus complexe des trois. En plus de l'API standard Gremlin, OrientDB offre une console interactive. Même si Neo4J l'a sorti de ses binaires depuis l'arrivée de Cypher (qui s'inspire justement de Gremlin pour la syntaxe de traversée), il existe malgré tout un plugin tiers pour continuer à l'utiliser.

4.3 Fonctionnalités

4.3.1 Index

Avec Neo4J, il est possible de créer des index d'une propriété de nœuds appartenant à un label à partir de Cypher, de l'interface REST ou d'une extension Java. Avec la version 2.0, de nouveaux index sont apparus sous le nom d'index de schéma. Il est encore possible d'utiliser les index dépréciés car (à ce jour) les nouveaux index ne sont que d'un seul type et ne s'appliquent pas aux propriétés d'arêtes d'un graphe. L'intérêt du nouveau mécanisme d'index est qu'il accélère le traitement des requêtes. L'exemple suivant montre la création d'un index sur la propriété "nom" du label Auditeur.

```
create index on :Personne(nom)
```

7“Cypher Query Language - - The Neo4j Manual v2.1.7.”
8“Commands | OrientDB Manual - Version 2.0.”
9“GremlinDocs.”

OrientDB gère plusieurs types d'index : arbre B amélioré, hash, texte et spatial ; on peut même implémenter son propre mécanisme d'index. Ces index s'appliquent à une propriété ou un groupe de propriétés (index composite) des nœuds et des arêtes mais aussi aux *RID*. L'exemple suivant montre la création d'un index composite unique sur la classe Application.

```
CREATE INDEX I_appli ON Application (nom, version, build) unique
```

4.3.2 Contraintes

Des contraintes d'unicité peuvent être créées sur les propriétés de groupes (labels et classes) de nœuds.

Voici un exemple de création de contrainte avec Neo4J :

```
CREATE CONSTRAINT ON (v:Ville) ASSERT v.nom IS UNIQUE
```

On ne peut plus créer de contraintes sur une propriété si celle-ci est déjà utilisée dans un des nœuds du label. On peut facilement visualiser l'état des index et des contraintes depuis la console.

```
neo4j-sh (?)$ schema
Indexes
  ON :Ville(nom) ONLINE (for uniqueness constraint)
Constraints
  ON (ville:Ville) ASSERT ville.nom IS UNIQUE
```

OrientDB gère en plus les contraintes d'unicité sur les arêtes et le nombre d'arêtes de classe par couple de nœuds. Il gère aussi les contraintes sur le typage, la déclaration obligatoire, les valeurs nulles, les limites numériques, l'affectation obligatoire et les règles d'expressions régulières.

Les langages fournissent des commandes de contournement pour manipuler des nœuds sans tenir compte des contraintes. Ces commandes seront par exemple utiles pour OrientDB en environnement distribué pour la création des nœuds et arêtes, si ceux-ci ont des contraintes.

4.3.3 Transactions

Neo4J et OrientDB font partie des SGBD NoSQL qui assurent les propriétés *ACID*. Un verrou d'écriture bloque un nœud, une arête ou une arête et son couple de nœuds.

Les 2 solutions préconisent de découper les transactions modifiant de gros volumes afin de ne pas saturer la mémoire centrale qui les stocke. Les transactions et leur propagation en architecture distribuée sont expliquées dans le chapitre 5.2 (p22).

OrientDB manipule par défaut les graphes en transactionnel ; ce qui n'est pas le cas de ses autres modèles de données.

4.3.4 Procédures stockées et fonctions

Les procédures stockées ont comme intérêts d'être dans la base de données, de factoriser le code et d'être compilées. Elle sont donc plus performantes car il n'y a pas besoin d'interpréter et compiler les commandes à chaque utilisation.

Neo4J n'a pas de mécanisme de procédures stockées. Il est cependant possible de développer des plugins mais Neo Technology préconise d'utiliser Cypher le plus souvent possible.

OrientDB stocke et exécute des *fonctions* qui sont l'équivalent des procédures stockées sur les SGBD relationnels. Elles peuvent être codées dans n'importe quel langage supporté. Elles supportent la récursivité, gèrent les arguments en entrée et peuvent être appelées à partir d'autres

fonctions, depuis la console, l'IHM, l'API REST ou encore dans une classe Java.

Les 2 SGBD fournissent des fonctions pour manipuler, rechercher et traverser les graphes. Les plus intéressantes sont celles des calculs de chemins : tous les chemins, chemin le plus court dont l'algorithme de Dijkstra. OrientDB fournit des fonctions supplémentaires pour manipuler des ensembles.

4.3.5 Triggers et hooks

Une des raisons d'être du NoSQL est d'écrire et de lire rapidement de très gros volumes de données ; utiliser des *triggers* ralentirait les traitements. Il est toujours possible d'intégrer ces actions dans le code client mais lorsqu'il y a plusieurs applications clientes il est parfois judicieux de les placer au plus près des données pour sécuriser et factoriser certains comportements. En développement, l'équivalent des *triggers* se nomment notamment des hooks (SVN, GIT), EventHandler (Java) et suivent le comportement du patron de conception Observateur.

Neo4J ne fournit pas de *triggers* en tant que fonctionnalité dans Cypher ou dans la console. Il est toutefois possible d'implémenter l'interface org.neo4j.graphdb.event.TransactionEventHandler, qui est utilisée en interne pour gérer les transactions[10], et de l'attacher au SGBD par un plugin Java.

OrientDB supporte les hooks qui sont rattachables sur des événements CRUD d'une classe de données ; il en existe deux types :

- Les hooks Java natifs sont des classes Java qui étendent la classe abstraite OrecordHookAbstract. Les inconvénients sont les suivants : il faut déployer les classes Java hook sur tous les nœuds du cluster, les insérer dans un fichier de configuration puis redémarrer toutes les instances du SGBD. L'avantage est qu'ils sont plus rapides à l'exécution que les hooks dynamiques.
- Les hooks dynamiques sont pris en compte dès leur création depuis la console ou l'IHM. Ils ont comme autre avantage de pouvoir exécuter n'importe quelle *fonction (§4.3.4)*.

4.3.6 Mécanisme de caches

Neo4J Enterprise possède deux niveaux de caches. Le premier est logique, il gère les entités alors que le second est un cache physique qui stocke des données telles qu'elles sont écrites dans les fichiers. Les caches de Neo4J ne se vident pas lors des redémarrages et il n'existe pas de commande pour les vider manuellement. Il est par contre possible de les régler très finement[11].

Chaque serveur OrientDB possède son propre cache logique indépendant. En architecture distribuée, un deuxième niveau de cache logique vient s'ajouter, c'est un cache commun à tous les nœuds[12]. Un dernier niveau de cache physique existe pour les bases de données stockées en local. La documentation n'est pas aussi complète que celle de Neo4J en ce qui concerne le paramétrage des caches.

4.3.7 Chargement de données en masse

Les 2 produits proposent des moyens de réaliser des ETL. A partir d'un fichier CSV local (file://) ou distant (http://), ils chargent des données dans la base en donnant un sens aux colonnes. L'exemple ci-dessous montre comment Neo4J charge les données et crée leurs liens. On constate dans la Figure 10 qu'un seul nœud de l'auditeur 'Nicolas' a été créé grâce à la commande MERGE et

10“org.neo4j.graphdb.event (Neo4j Community 2.0.3 API).”
11“22.6. Caches in Neo4j - - The Neo4j Manual v2.1.7.”
12“Caching | OrientDB Manual.”

non quatre.

```
LOAD CSV WITH HEADERS FROM "file:/tmp/exemple1.csv" AS line
MERGE (auditeur:Auditeur { nom: line.auditeur })
CREATE (ue:UE { code: line.UE, cursus: line.cursus })
CREATE (auditeur)-[:VALIDE { note:toInt(line.note), annee:toInt(line.annee)}]->(ue)
Added 5 labels, created 5 nodes, set 17 properties,
created 4 relationships, [..] in 148ms
```

```
cat /tmp/exemple1.csv
UE,label,cursus,auditeur,note,annee
NFA004,Archi machines,L1,Nicolas,1,2009
NFA008,Bases de données,L2,Nicolas,2,2009
NFA010,Graphes,L2,Nicolas,3,2010
ENG221,Infocom ingénieur,ING,Nicolas,4,2015
```

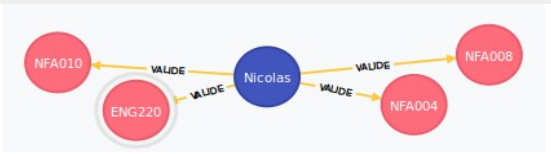


Figure 10: Neo4J : résultat à la suite du chargement CSV

OrientDB va plus loin en offrant l'extraction de données à partir d'un format JSON, d'une sortie Shell ou directement depuis une base de données tiers grâce à JDBC. Parmi tous les convertisseurs qui permettent de transformer les données brutes extraites, on retrouve les convertisseurs de nœuds et d'arêtes. Contrairement à Neo4J, le chargement s'effectue par une commande Shell et un fichier de configuration JSON.

```
$ORIENTDB_HOME/bin/oetl.sh charge_exemple1.json
```

L'ETL de Neo4J est plus simple et plus rapide à configurer mais celui d'OrientDB permet plus de paramétrage.

4.4 API et pilotes

	Protocole binaire	C/C++	Cypher	Clojure	Erlang	Go	Gremlin	Haskell	Java	JavaScript	.NET	Node.js	Perl	Php	Python	Rest	R	Ruby	Sail	Scala	Pseudo SQL	TinkerPop
Neo4J			**Oui**	Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui		Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui		Oui
OrientDB	Oui	Oui		Oui			**Oui**		Oui	Oui	Oui	Oui	Oui	Oui	Oui	Oui		Oui	**Oui**	Oui	Oui	Oui

Tableau 4: API clientes prises en charges

Le Tableau 4 présente un grand nombre de pilotes disponibles pour différents langages de développement. Les langages les plus utilisés sont pris en charge et disposent de plusieurs implémentations qui communiquent avec les interfaces Java ou REST (HTTP) de ces deux SGBD. Il n'existe pas à ce jour de pilote stable C/C++ pour Neo4J, ni de pilote Erlang pour OrientDB.

Plusieurs pilotes communiquent avec OrientDB grâce à un protocole binaire par TCP/IP, dont les principaux avantages sont des gains en temps de traitement et en débit.

4.5 Limites

Maximum	Neo4J	OrientDB
Bases de données hébergées	1	au moins 1000
Nombre de nœuds	2^35 (≅ 1,6.10^16)	2^78-1 (≅ 7,5.10^34)
Nombre de relations	2^35	Information non trouvée
Nombre de propriétés	2^36	Information non trouvée
Nombre de label/clusters	2 milliards	2^15-1
Nombre d'entrées par label/cluster	2^35	2^63-1
Nombre d'index	Information non trouvée	2 milliards
Taille pour une entrée	1To	2Go
Nombre de réponses à une requête	2 milliards	2 milliards
Longueur d'une entrée	Information non trouvée	Pas de limitation
Quantité de données modifiables dans une transaction	Espace libre en mémoire	Espace libre en mémoire

Tableau 5: Comparaison des limitations des données

Le Tableau 5 présente les limites de capacité des 2 SGBD. OrientDB peut gérer 4,7.10^18 (exa) plus de nœuds que Neo4J. Ce dernier, limité par son architecture master/slave, ne peut pas faire de *sharding*.

Neo4J gère jusqu'à 1To de propriétés pour une entrée ; c'est plus qu'OrientDB qui, d'ailleurs, pour des raisons de performances, recommande de ne pas stocker directement une entrée volumineuse (>10Mo) et d'utiliser un type de valeur qui fait référence à un fichier local. Il faut en effet se poser la question de l'utilité d'une base orientée graphe pour gérer un grand volume de propriétés mais peu de nœuds/relations.

4.6 Conclusion

Les deux solutions proposent des approches différentes de l'implémentation du modèle de données et de son utilisation.

Cypher est un langage simple et complet. Le pseudo-SQL apporte aussi des fonctionnalités intéressantes. Gremlin les surpasse en fonctionnalités mais est plus complexe.

Gérer des modèles de données hétérogènes avec un seul et même produit est un avantage en terme de formation, d'administration, maintenance et mise à jour. Mais cela peut éventuellement être aussi un inconvénient : un modèle de données est-il aussi bien géré dans une solution qui supporte plusieurs modèles différents que dans une solution spécifiquement conçue pour ?

Le SGBD multi-modèle OrientDB prouve que c'est possible et offre même beaucoup plus de fonctionnalités que le SGBD spécifique orienté graphe Neo4J. Le fait qu'OrientDB possède 4 fois moins de lignes de code source que son concurrent est d'autant plus surprenant.

5 Architectures

5.1 Réplication et failover

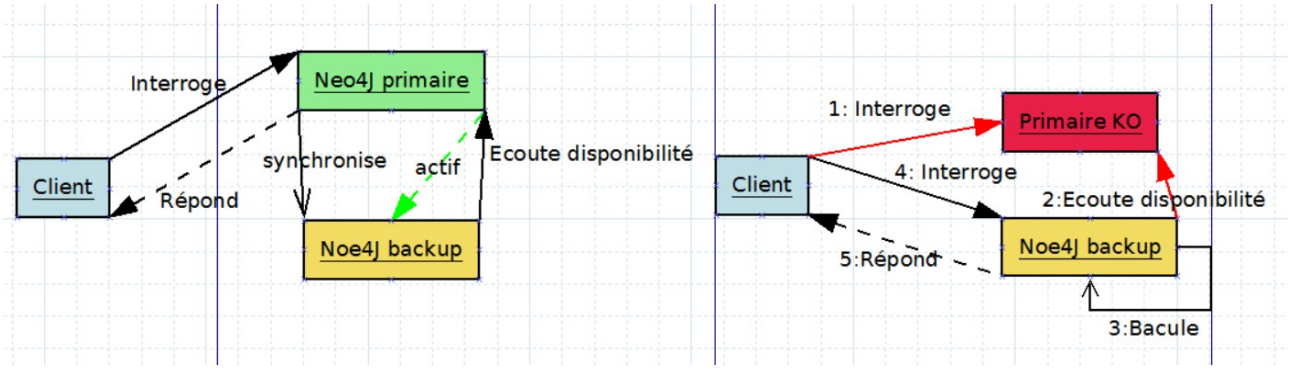


Figure 11: Neo4J : diagrammes de collaboration des modes nominal et dégradé pour la haute disponibilité

Neo4J Enterprise apporte de la haute disponibilité de la façon suivante : un serveur primaire traite toutes les requêtes à la manière d'un serveur *standalone* et envoie les requêtes à une seconde instance sur un serveur de backup(Figure 11 diagramme de gauche) , prête à basculer dès qu'une indisponibilité du primaire sera détectée (diagramme de droite). Ce comportement se rapproche de la solution Data Guard d'Oracle. Il est possible d'activer l'instance de backup pour traiter les requêtes de lecture, comme le font notamment Oracle Active Data Guard ou PostgreSQL Hot Standby. Neo4J sait synchroniser ses bases de données d'un master vers plusieurs slaves. Cette fonctionnalité sera détaillée dans le prochain chapitre.

Pour bien comprendre le fonctionnement d'OrientDB en architecture distribuée, il faut avoir vu la notion de *cluster* présentée dans le chapitre 4.1. Tous les serveurs d'une grappe communiquent entre eux de façon décentralisée grâce à l'application distribuée *HazelCast*.

Un nouveau serveur se connecte d'abord sur le bus de communication puis synchronise les *replicas* des *clusters* qu'il doit héberger avant de devenir actif. Les serveurs informent leurs clients de cet ajout. Lorsqu'un serveur tombe, ses clients se redirigent automatiquement vers d'autres serveurs actifs. Le serveur pourra ensuite rejoindre le *cluster* en resynchronisant ses *clusters* obsolètes aux autres.

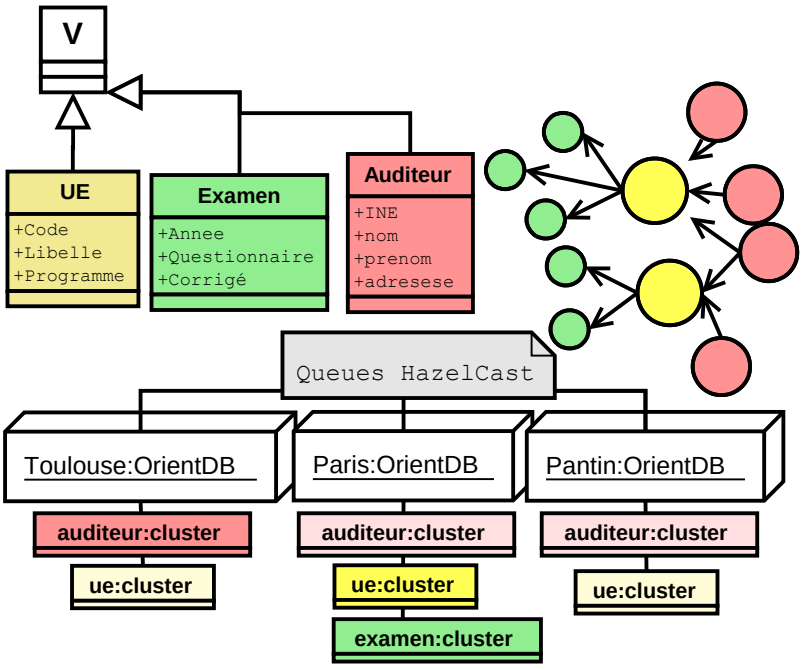


Figure 12: OrientDB : Les 3 classes de nœuds sont répliquées sur 3 instances

La Figure 12 propose un exemple de 3 classes de nœuds en relation composant un graphe hébergé par OrientDB en mode distribué sur 3 nœuds d'un *cluster*. L'instance Toulouse héberge le *cluster* par défaut de la classe Auditeur qui est répliqué sur les 2 autres serveurs tandis que le *cluster* de la classe Examen ne se trouve que sur Paris.

5.2 Communication et élasticité

L'architecture distribuée de Neo4J repose sur des serveurs qui hébergent la même base de données. Cette dernière reste synchronisée grâce au master qui gère les transactions et les verrous de ressources de tous les slaves. Cette gestion centralisée est représentée par les flèches bleues sur la Figure 13.

Un nœud A reçoit une transaction et l'envoie au master. Ces 2 serveurs vont exécuter chaque opération en même temps en bloquant les ressources nécessaires. Une fois les opérations de la transaction exécutées, le master va commit en premier la transaction et la diffuser ensuite vers tous les autres slaves du cluster, excepté sur le nœud A qui, ayant déjà effectué les opérations a juste besoin d'un commit. Un slave n'arrivant pas à réaliser la transaction diffusée par le master se comportera de la même manière qu'un master déchu.

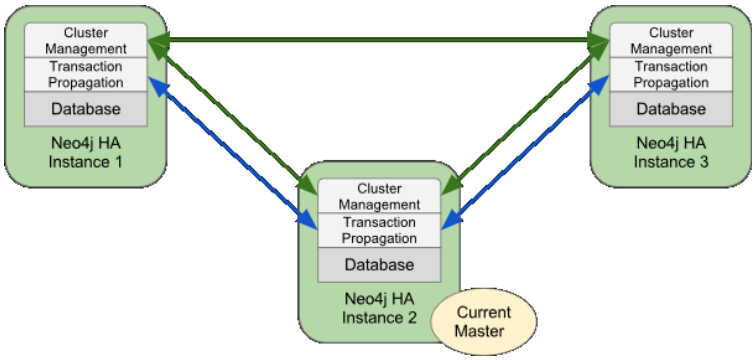


Figure 13: Neo4J : architecture distribuée d'un cluster à 3 nœuds

En cas d'indisponibilité du master, un slave est promu à sa place. L'ancien master se déconnectera du *cluster* et se resynchronisera avec le nouveau master en écrasant l'ancienne base puis deviendra un slave.

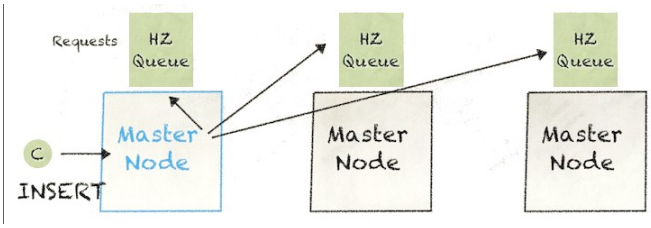
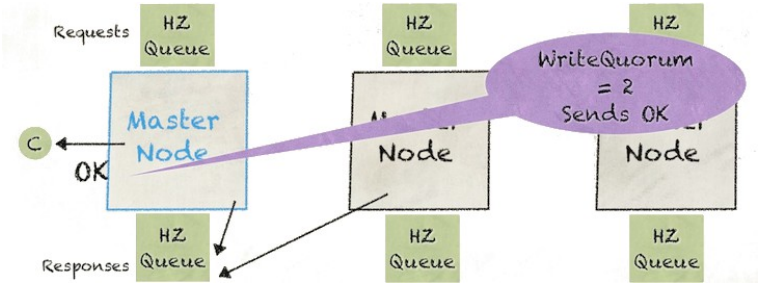


Figure 14: OrientDB : communication pour une architecture distribuée d'un cluster à 3 nœuds



La Figure 14 explique les différentes phases de communication dans une grappe de serveurs OrientDB [13].

N'importe quel serveur peut traiter des requêtes de lecture ou d'écriture (l'usage de transactions est obligatoire pour bénéficier des propriétés ACID et ainsi assurer la consistance des graphes). Le serveur qui reçoit la requête va l'injecter dans une pile pour chacun des serveurs de la grappe et va attendre un nombre minimal de réponses positives de traitement avant de communiquer au client que son action s'est terminée avec succès. Certains nœuds se désactiveront si la mise à jour s'est mal déroulée pour eux. Le master qui a reçu la requête les resynchronisera afin qu'il puissent réintégrer le cluster le plus rapidement possible.

13"Distributed Architecture | OrientDB Manual - Version 2.0."

5.3 Sharding

Le *sharding* est une fonctionnalité en architecture distribuée disponible sur OrientDB. Aussi appelée partitionnement horizontal en français, elle désigne la possibilité de morceler une classe de données sur plusieurs nœuds. La classe possède son *cluster* par défaut sur un des nœuds qui, contrairement à une simple réplication, est le seul point d'entrée pour traiter les manipulations de données dans son périmètre et mettre à jour ses répliquas. Une application cliente recevra une erreur si elle essaye explicitement d'attaquer un répliqua avec une requête.

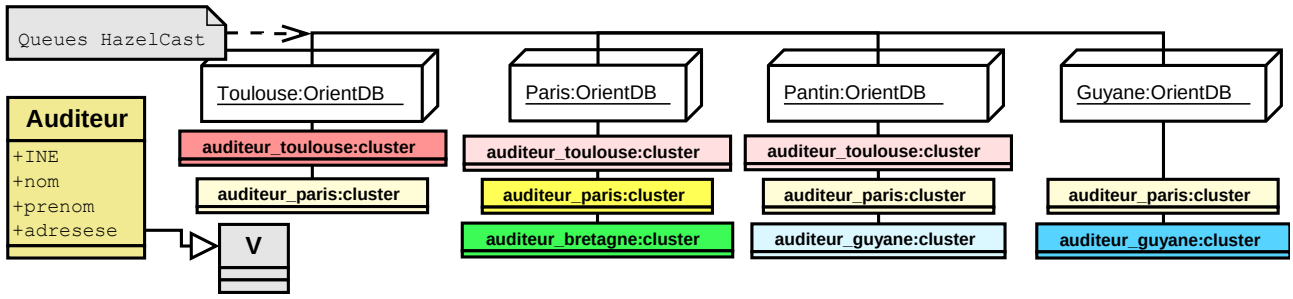


Figure 15: OrientDB : la classe Client est utilisée en partitionnement horizontal avec réplication

En exemple, la Figure 15 propose le partitionnement horizontal de la classe Auditeur en fonction de son centre de rattachement.

5.4 MapReduce

OrientDB implémente le *MapReduce*.

Neo4J ne gère pas cette fonctionnalité mais certains ont contourné cette limitation en utilisant *Hadoop* comme orchestrateur de Neo4J [14]. *Hadoop* va découper et transformer ses données directement en fichiers physiques de bases de données Neo4J et va les transmettre aux *workers* qui vont utiliser ce SGBD pour traiter ces morceaux séparément. *Hadoop* va ensuite rassembler, corréler les données et fournir le résultat final à travers Neo4J.

5.5 Conclusion

Les 2 SGBD supportent la haute disponibilité en architecture distribuée. A partir d'un serveur *standalone*, il sera possible de passer en *cluster* très facilement sans réaliser de migrations applicatives ou de données. Il n'aura même pas besoin de couper le service avec OrientDB.

En 2012, Orient Technologies a su faire évoluer son produit pour répondre au besoin *d'élasticité* en passant de master-slave à l'architecture actuelle. OrientDB offre une solution de haute disponibilité élastique, facile à concevoir et mettre en œuvre.

Neo4J centralise la réplication et les transactions, le master deviendra un goulot d'étranglement dès qu'une de ses ressources sera saturée : charge processeur, mémoire, trafic réseau, lecture et écriture des disques durs, etc. Le master pouvant être n'importe quel nœud, il n'est pas envisageable d'améliorer un seul serveur master pour lui faire soutenir plus de charge. Il est regrettable que Neo4J n'apporte pas de la *scalabilité* en écriture. Mais cet inconvénient se transformera peut-être en avantage. Dans bien des cas, le modèle master-slaves communique plus simplement et plus rapidement qu'un modèle multi-master.

14“Combining Neo4J and Hadoop (part I) | Xebia Blog.”

6 Exploitation en production

Neo4J et OrientDB ont-ils été pensés pour la production informatique ?

6.1 Installer, configurer et démarrer

Les 2 produits sont aussi faciles à installer et à configurer l'un que l'autre. Les étapes sont :

1. L'installation par la compilation en byte code d'un code source java n'apportant pas de gain de performances, on lui préférera la décompression des binaires ;
2. la personnalisation de la configuration ;
3. le déploiement du script en tant que service ;
4. enfin, le démarrage du service.

Dû à la diversité des dépendances utilisées, les fichiers de configuration présents dans conf/ et config/ ne sont pas homogènes. En effet, on y trouve des fichiers Java properties, JSON et XML.

Neo4J Community démarre grâce à la commande ci-dessous ou par les services.

```
$NEO4J_HOME/bin/neo4j console
```

Le serveur Neo4J démarre par la classe org.neo4j.server.Bootstrapper qui s'occupe de lancer, soit la version Enterprise, soit la Community. Pour ce dernier, org.neo4j.server.CommunityNeoServer démarre à son tour les différents modules aux noms explicites : DBMSModule, RESTApiModule, ManagementApiModule, ThirdPartyJAXRSModule, WebAdminModule, Neo4jBrowserModule, AuthorizationModule et SecurityRulesModule.

OrientDB démarre grâce à la commande ci-dessous ou par les services.

```
$ORIENTDB_HOME/bin/server.sh
```

Une instance OrientDB qui démarre vérifie sa consistance avant de passer en ligne. Dans le cas où elle s'est arrêtée à la suite d'un crash, elle revient automatiquement au dernier état consistant. Dans une architecture distribuée, le nœud en erreur se synchronisera avec les nœuds opérationnels. Au démarrage, la classe com.orientechnologies.orient.server.Oserver instancie un listener et les plugins. Les plugins sont configurés dans le fichier de configuration config/orientdb-server-config.xml. Parmi eux, l'interface REST et l'IHM qui sont créés dans ...orient.server.network.protocol.http.ONetworkProtocolHttpDb.

OrientDB fournit une procédure d'installation spécifique pour Docker, un gestionnaire de conteneurs applicatifs Open Source en pleine ascension déjà utilisé par de grandes sociétés du Web.

6.2 Superviser et analyser

La supervision d'une base de données permet d'analyser le comportement du SGBD grâce à la surveillance d'indicateurs techniques stockés dans le temps. Elle alerte et déclenche automatiquement des actions suivant des seuil établis.

Comme vu précédemment, il n'y a qu'un seul processus Neo4J à surveiller.

Une interface JMX de Neo4J Enterprise permet d'accéder au comportement du SGBD et son activité générale à un instant T. Pour avoir une vraie supervision, il faudra développer soi-même une application spécifique communiquant par JMX ou utiliser un plugin d'un outil de supervision.

OrientDB Enterprise stocke des statistiques précises et possède un IHM d'analyse pour faciliter le tuning et détecter les goulots d'étranglements. De plus, un service de supervision permet de configurer des seuils et d'être alerter par email ou de déclencher des actions automatiques au travers de requêtes HTTP.

6.3 Administrer

L'administration de Neo4J s'effectue en ligne de commande, par l'IHM et par l'interface REST.

L'interface HTTP d'administration de Neo4J est découpée en trois parties. A gauche, un menu permet la consultation et la manipulation de la configuration et des données. Ce ne sont en fait que de simples raccourcis personnalisables de commandes Cypher : en cliquant dans le menu, la commande Cypher ou REST associée est affichée dans la seconde partie, en haut de l'IHM. Il est possible de modifier cette commande, de l'enregistrer dans le menu ou de l'exécuter. Le résultat s'affichera alors dans la troisième partie au format texte, JSON ou dans un graphe interactif.

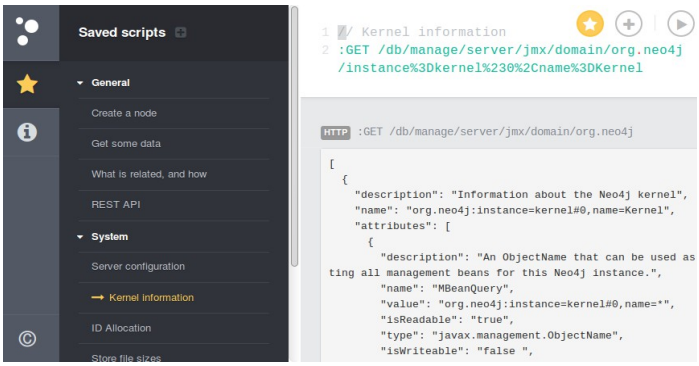


Figure 16: Neo4J : une IHM simple et efficace

L'administration manuelle est facile à prendre en main et l'interface REST permet l'automatisation de tâches d'administration faisant de l'IHM Neo4J une application HTTP simple et efficace.

Celle d'OrientDB est quant à elle plus complexe et plus longue à prendre en main, en raison de ces nombreuses fonctionnalités.

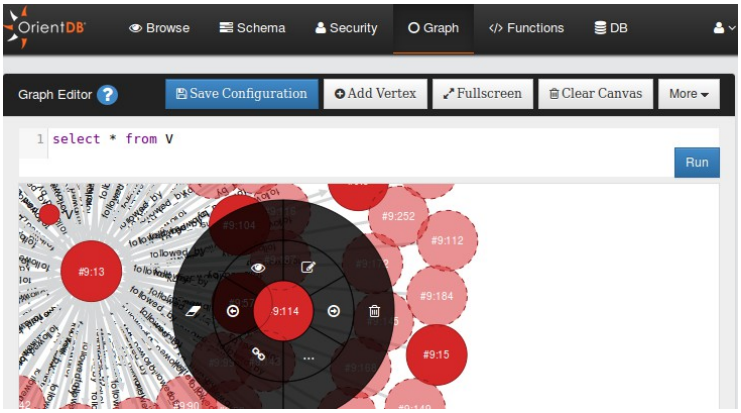


Figure 17: OrientDB : l'éditeur de graphes de l'IHM complet reste intuitif

En plus de permettre la manipulation des données, de leur structure et des graphes, l'interface HTTP d'administration permet de configurer finement l'instance et ses clusters, les procédures stockées (fonctions), les hooks, la gestion des utilisateurs et même d'exporter la base de données complète au format JSON.

L'affichage interactif des graphes est plus complet que celui de NEO4J. Comme le montre la Figure 17, la navigation et la personnalisation offrent de la lisibilité pour des graphes complexes grâce à une interface circulaire contextuelle riche et géographiquement proche du nœud en question. On remarquera des bugs et des problèmes d'affichage qui font perdre du temps lors de la consultation des graphes, notamment le rafraîchissement des titres des nœuds et le fait de devoir déployer les relations manuellement à chaque consultation.

Comme pour Neo4J, l'administration s'effectue aussi en ligne de commande et par l'interface REST.

6.4 Sauvegarder et restaurer

Les 2 solutions permettent au moins de réaliser un export (sauvegarde logique) des données. Mais ce n'est pas une sauvegarde de la base et il ne sera pas possible de la restaurer directement à la suite d'un crash par exemple.

Neo4J Community Edition n'a pas d'outil dédié ; le seul moyen de réaliser une sauvegarde est de couper le service en arrêtant le SGBD et en réalisant une copie des fichiers.

Neo4J Enterprise possède une commande pour effectuer des sauvegardes à chaud d'un nœud distant sur un serveur ayant les binaires installés. Le résultat de la sauvegarde est une copie conforme et fonctionnelle de la base de données. Si le répertoire de destination contient déjà un backup précédent, la sauvegarde sera incrémentale et mettra à jour la base de sauvegarde. C'est pourquoi l'étape de restauration consiste à un simple remplacement du répertoire de la base de données par celui de la sauvegarde.

OrientDB propose une sauvegarde « tiède » (base ouverte mais verrous pour l'écriture sur toutes les ressources durant l'action) pour un serveur seul. Si le serveur héberge la base de données sur un FS de type LVM ou s'il est en cluster, la sauvegarde à chaud est alors possible.

OrientDB Enterprise permet la sauvegarde à chaud.

En cas de crash, une restauration pourra donc être faite mais il n'existe pas de mécanisme de « redolog » comme sur Oracle pour retrouver un état consistant postérieur à une sauvegarde. Mais l'utilisation d'une architecture distribuée réduit les risques de perte de donnée lié à ce problème. Si les données sont critiques il est toujours possibles de dédier un des nœuds du cluster pour sauvegarder fréquemment la base.

6.5 Sécuriser

Il n'est jamais inutile de rappeler que la sécurisation d'un SGBD ne se substitue en aucun cas à la sécurisation des couches inférieures (configuration système, chiffrement du FS, contrôle des flux réseaux, accès physique) et supérieures (applications clientes du SGBD). Il conviendra au minimum de créer un utilisateur applicatif non root dédié au SGBD et de restreindre ses droits et son arborescence au minimum.

6.5.1 Gestion des accès

Neo4J n'intègre pas de gestion de permissions ni d'authentification des utilisateurs. Il est toutefois possible d'étendre les classes de sécurité de Neo4J pour mettre en place un système de permission d'accès sur les données par leur URL, mais cela nécessite un développement Java spécifique.

OrientDB possède une gestion fine et efficace des permissions des rôles et utilisateurs, en mode console ou par l'IHM. On associe à un rôle des permissions sur les différentes ressources du SGBD grâce à un masque 4 bits représentant les opérations CRUD.

6.5.2 Communication

Les 2 SGBD permettent de choisir le port d'écoute et l'interface réseau. Ils prennent aussi en charge les communications par SSL mais seul Neo4J l'utilise par défaut.

L'utilisation d'un *reverse proxy* pallie leurs lacunes, en configurant des IP autorisées à utiliser le SGBD. Il sera par la même occasion possible de mettre en place une barrière d'authentification par le *reverse proxy* pour Neo4J.

6.5.3 Protection des données

Les 2 produits ne savent pas chiffrer leurs données en base, à l'exception d'un trigger d'OrientDB qui chiffre les mots de passe utilisateurs. Un chiffrement des données en amont par l'application ou un chiffrement du FS sont alors les seules alternatives.

6.6 Migrer une base de données vers un autre produit

Les 2 produits permettent d'exporter leurs données simplement sous plusieurs formats libres. GraphML est un format libre de stockage de graphes.

OrientDB dispose d'une commande Gremlin pour rapatrier directement un export GraphML d'une base Neo4J.

```
g.loadGraphML('/data/baseneo4j.export.graphml');
```

Avec le plugin Gremlin de Neo4J, l'inverse doit aussi être possible. Attention aux fonctionnalités qu'il ne sera plus possible de mettre en place sur Neo4J.

Les migrations vers d'autres SGBD, orientés graphes ou non, devront faire l'objet d'une étude particulière : il faudra avant tout se poser les questions sur la volumétrie supportée et les autres limites sur les données, ainsi que les fonctionnalités attendues.

6.7 Conclusion

Dans le domaine de l'exploitation, OrientDB surpasse Neo4J en tout point.

Il faut toutefois remarquer la qualité et la simplicité de l'IHM de Neo4J et son système de marque-pages offrant une personnalisation fine et didactique. En effet, chaque lien du menu affiche le Cypher associé, ce qui accélère grandement l'apprentissage du langage.

7 Performances

7.1 Généralités

Plusieurs comparaisons de performances de ces deux produits ont été réalisées et publiées sur Internet mais elles sont anciennes ou ne sont pas exploitables à cause du manque d'information sur les protocoles de tests ou simplement des lacunes dans les procédés. De plus, le comportement d'une base de données varie énormément suivant la version utilisée et son environnement.

Pour avoir des résultats utiles, il faut mettre en place une maquette de test se rapprochant le plus possible de ce que serait l'environnement de production. Les éléments suivants sont donc à prendre en considération :

- le protocole de communication entre les nœuds ;
- la qualité du réseau de communication interne au cluster ;
- la qualité du réseau métier reliant les clients aux serveurs ;
- les ressources physiques propres à chaque nœud : CPU, mémoire ;
- le choix des FS, performance en lecture/écriture des disques durs ;
- le type de stockage local, RAID, SAN ;
- l'impact du choix de l'OS sur les performances ;
- la préparation des scénarios et des jeux de données (volumétrie, structure des données, enchaînement des opérations) ;
- la charge du serveur au moment des tests ;
- l'état des caches des SGBD.

L'enjeu est de répondre à la problématique « De combien et de quelles ressources physiques et logicielles avons-nous besoin pour répondre au besoin métier présent et futur » et se situer par la même occasion entre performances et fonctionnalités.

Il faudra aussi faire attention en extrapolant les résultats des tests. Par exemple, les performances d'un cluster de 10 nœuds pourraient n'avoir aucune similitude avec les résultats d'un test à 5 serveurs. Ou encore : si l'ajout de 1000 nœuds et 2000 relations à un graphe prend 5 secondes, il n'est pas du tout certain que l'insertion de 10 000 nœuds et 20 000 relations prenne 50 secondes.

7.2 Un exemple de bonnes pratiques

Réaliser des écritures de masse puis des lectures de masse est très souvent un comportement éloigné de la réalité des opérations effectuées sur des bases de données. Il faut utiliser des modèles et des situations se rapprochant le plus avec le besoin. L'exemple suivant montre une méthodologie scientifique qui a pour avantage de produire des résultats exploitables, utiles.

L'étude *Graph Database Benchmarking on Cloud Environments with XGDBench* [15] met au point une méthode de mesure de débits dans un environnement *HPC* par l'utilisation d'un framework.

Les auteurs génèrent une structure, des scénarios et des jeux de données dans le but de se rapprocher du comportement d'une situation réelle. Pour affermir leur propos, ils choisissent de mettre en pratique leur framework pour les réseaux sociaux dont ils en définissent les caractéristiques :

- des volumétries conséquentes ;

15Miyuru, Dayarathna; Toyotaro, Suzumura, “Graph Database Benchmarking on Cloud Environments with XGDBench.”

- 0.95 de probabilité d'une lecture et 0.05 d'écriture ;
- les relations entre les nœuds (personnes) varient peu comparées à leurs propriétés (contenu) ;
- les mises à jour de propriétés des graphes sont donc plus importantes que les mises à jour de leur structure ;
- trouver les nœuds en lien direct avec un nœud ;
- traverser plusieurs nœuds à partir d'un point de départ ;
- l'environnement distribué.

Hélas, les chercheurs ont été limités pour ce dernier point par d'autres SGBD étudiés et ont effectué leurs tests avec seulement un nœud du cluster. Les comportements relatifs aux choix d'architectures master/slave de Neo4J et multi-master d'OrientDB ne seront donc pas pris en compte.

Ils ont d'abord créé des données grâce à un générateur de graphes paramétrable et ont ensuite défini plusieurs scénarii parmi lesquels des charges en lecture-écriture à 50-50 (Workload A) , 95-5 (Workload B) et 100-0 (Workload C).

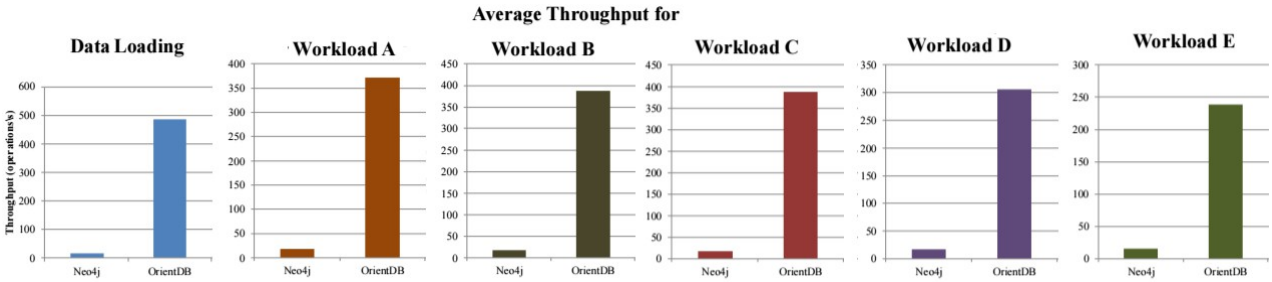


Figure 18: Débits moyens en réponse aux différents jeux de tests (en nombre d'opérations par secondes)

Neo4J version 1.6.1 Community Edition et OrientDB 1.0rc9 ont été utilisés pour ces tests réalisés en 2012.

Les résultats de ces mesures Figure 18, affichent clairement de meilleures performances d'OrientDB par rapport à Neo4J dans tous les cas mais, comme indiqué précédemment, ces résultats n'ont strictement aucun lien avec les versions des SGBD étudiés et sont associés à une situation spécifique.

8 Conclusion générale

Neo4J et OrientDB ont été comparés en descendant dans les différentes couches du SI.

Les 2 SGBD ont des lacunes dans leur documentation sur certains paramétrages et possibilités.

OrientDB manque encore de maturité pour un SGBD pour entreprise et quelques bugs dans la visualisation des graphes sont à corriger dans l'IHM. Pourtant, il apparaît qu'OrientDB possède bien plus de fonctionnalités que Neo4J dont certaines sont incontournables lorsqu'on gère des volumétries importantes en production.

En effet, Neo4J n'est pas élastique horizontalement et ne gère pas le MapReduce ni les accès aux données. S'il souhaite concurrencer OrientDB sur les fonctionnalités, il va devoir se transformer en profondeur et perdre son architecture master/slave. Neo Technology devrait, peut-être aussi, revoir son business model.

Mais il ne faut pas s'arrêter à ce constat : il est en général impossible de choisir sérieusement une solution à un besoin sans se poser au préalable la question de ce qu'on veut faire avec. D'autant plus que chaque solution NoSQL est spécialisée pour résoudre un type particulier de problème et ne conviendra pas pour un autre.

Le Tableau 6 (p32) liste et valorise l'ensemble des points étudiés. Afin de lui donner un sens, il conviendra :

1. de l'utiliser pour une étude ;
2. de réaliser plusieurs séries de tests de performance et faisabilité (§7.1) ;
3. d'ajouter dans le tableau les résultats des scénarios de tests et d'autres comparaisons spécifiques au besoin ;
4. de pondérer chaque ligne du tableau en fonction des priorités (dernière colonne) ;
5. de calculer les points des 2 solutions pour chaque ligne (points x pondération).
6. Faire le total des points permettra de faire un choix sur la meilleure solution à utiliser dans le cas étudié.

Les tests de faisabilité et performance sont décisifs dans le choix : leurs résultats dépendent de l'architecture, de sa configuration, des données manipulées et de l'environnement global.

En outre, il sera intéressant d'élargir l'étude aux autres SGBD orientés graphe, parmi lesquels : AllegroGraph, HyperGraphDB, InfoGrid, Sparksee(DEX), Titan ou encore Weaver.

§ de réf	**Généralités**	**Neo4J**	**OrientDB**	Facteur exemple
2.1& 3.1	Type de SGBD	Orienté graphe	Multi-modèle	1
2.1& 3.1	Licence (versions CE)	GPL v3	Apache 2	1
2.1& 3.1	Licence (versions Enterprise)	AGPLv3/Propriétaire	Propriétaire	1
2.4& 3.4	OS supportés	-	+	1
2.1& 3.1	Années d'existence	+	-	0
2.2& 3.2	Communauté	+	-	0.1
2.2& 3.2	Documentation, fonctionnalités, structure, exemples	+	-	0.5
2.2& 3.2	Détails des informations dans la documentation	-	+	0.5
2.3& 3.3	Services aux entreprises	-	+	1
2.3& 3.3	Offre de consulting aux entreprises	+	-	1
2.3& 3.3	Coût du support	-	+	2
	Fonctionnalités			
4.1 p11	Stockage base en mémoire	Non	**Oui**	0.5
4.1 p11	Stockage base sur disque	**Oui**	**Oui**	2
4.1 p11	Gestion de base distante	Non	**Oui**	0
4.1 p11	Assurer les propriétés ACID	**Oui**	**Oui**	2
4.1 p11	Types de propriétés gérés	-	+	
4.2 p13	Simplicité d'utilisation du langage (avis subjectif)	+	-	1
4.2 p13	Lisibilité des requêtes	+	-	1
4.2 p13	Syntaxe du langage en lecture et écriture	+	-	1
4.2 p13	Fonctionnalités de traversée	+	-	1
4.2 p13	Utilisation du langage Gremlin	-	+	1
4.2 p13	Schéma	Optionnel	Optionnel	1
4.4 p18	API	=	=	1
	API utilisée dans le projet	A faire soi-même	A faire soi-même	5
4.3.1 p15	Index	-	+	1
4.3.2 p16	Contraintes	-	+	1
4.3.4 p16	Procédures stockées	-	+	1,5
4.3.5 p17	Triggers	-	+	1
6.3 p26	Interface REST	**Oui**	**Oui**	1
5.1 p21	Haute disponibilité (versions CE)	Non	**Oui**	1
5.1 p21	Haute disponibilité (versions Enterprise)	**Oui**	**Oui**	8
5.3 p23	Partitionnement horizontal	Non	**Oui**	1
5.1 p21	Tolérance de panne	**Oui**	**Oui**	1
5.1 p21	Réplication	-	+	1
5.2 p22	Scalabilité horizontale	En lecture seulement	**Oui**	3
5.2 p22	Élasticité horizontale	Non	**Oui**	2
4.3.3 p16	Gestion des transactions	**Oui**	**Oui**	5
5.4 p23	MapReduce	Non	**Oui**	1
5.2 p22	Transactions distribuée optimistes	**Oui**	**Oui**	1
4.5 p19	Capacité des données	-	+	1
4.3.7 p17	ETL	**Oui**	**Oui**	1
6.2 p25	Profiler de requêtes (Enterprise)	Non	**Oui**	1
6.2 p25	Stockage et analyse d'indicateurs (Enterprise)	Non	**Oui**	1
4.3.6 p17	Niveaux de caches	2	3	1
	Exploitation			
6.1 p25	Installation et configuration	=	=	1
5.1 p21	Installation et configuration d'un cluster	-	+	1
6.2 p25	Supervision (Enterprise)	-	+	1
6.2 p25	Alertes et déclenchements automatiques d'actions (Enterprise)	Non	**Oui**	1
6.3 p26	Prise en main de l'interface Web (avis subjectif)	+	-	1
6.3 p26	Fonctionnalités de l'interface Web	-	+	0.5
6.3 p26	Personnalisation des graphes	-	+	1
	Interface console	Non étudié	Non étudié	1
6.4 p27	Sauvegarde sans coupure de service (CE)	Non	**Oui**	1
	Sauvegarde à chaud (Enterprise)	**Oui**	**Oui**	5
6.5.1p27	Gestion des accès	Non	**Oui**	2
6.6 p28	Facilité pour migrer les données	=	=	1
	Performances			
	Performance des caches	Non étudié	Non étudié	1
	Résultats scénario 1	A faire soi-même	A faire soi-même	8
	Résultats scénario 2	A faire soi-même	A faire soi-même	12
	Etc. Autres points à comparer	...	...	...

Tableau 6: Récapitulation du comparatif

Légende	Non comparable ou non étudié	Égalité	**Avantage**	Faiblesse	Manque
Points	0	3	**3**	1	0

9 Glossaire

ACID

Acronyme désignant les propriétés d'Atomicité, de Cohérence, d'Isolation et de Durabilité que les SGBD doivent garantir pour fournir un mécanisme transactionnel.

API

(Application Program Interface) Ensemble de fonctions d'une librairie, d'un logiciel qui communique avec d'autres par le biais d'un protocole commun.

Base

Base de données.

Base de données

Une base de données stocke de l'information de façon plus ou moins structurée. Elle fournit des données en traitant des requêtes concurrentes au travers d'API.

Cluster (architecture physique)

Groupe, grappe de serveurs travaillant ensemble dans un but défini.

Cluster (OrientDB)

Partition d'une classe de données dans OrientDB qui peut avoir plusieurs replicas synchronisés et hébergés sur les instances souhaitées. Terme abordé dans le chapitre 4.

CRUD

(Create Read Update Delete) Opérations standards que l'on peut réaliser sur les données.

Élasticité

L'élasticité horizontale est l'ajout ou suppression d'un nœud dans un cluster et de ses instances applicatives distribuées pour se rapprocher du besoin en charge d'un service qui évolue dans le temps. Contrairement à la scalabilité, l'élasticité varie à la demande et sans coupure de service.

Pour subvenir au même besoin, l'élasticité verticale est l'ajout ou la suppression de ressources physiques (CPU, mémoire, stockage) et logicielles (démon) sur un nœud d'un cluster.

ETL

(Extract Transform Load) Fonctionnalité qui vise à extraire des données d'un environnement tiers, de les transformer en un format souhaité pour ensuite les charger plus facilement dans une base de données.

Failover

Caractéristique d'une architecture permettant de basculer automatiquement d'un équipement en panne à un autre pour préserver la disponibilité du service délivré.

FS

(FileSystem) Système de fichiers.

Framework

Ensemble de règles et d'outils définissant un cadre de travail pour une application.

Graphe

Un graphe est une représentation d'un ensemble de nœuds reliés entre eux par des arrêtes (graphe non-orienté) ou des arcs (graphe orienté). Ces relations peuvent produire des boucles, des graphes multigraphes, des arbres, etc.

Hadoop

Solution de stockage et de calcul distribués de données volumineuses.

HPC

(High Performance Computing) Clusters et grilles de serveurs dédiés au calcul intensif.

IT

(Information Technology) Les technologies de l'information représentent l'usage de la technologie pour le stockage, la communication ou le traitement de l'information. [16]

MapReduce

Processus de traitement distribué d'informations volumineuses. Les données sont découpées et dispatchées sur des nœuds qui réaliseront des travaux indépendants. Leurs résultats seront ensuite rassemblés pour fournir le résultat final.

Nœud (théorie des graphes)

Un nœud, ou sommet, est une représentation d'une entité ou d'une information dans un graphe.

Nœud (architecture physique)

Le terme nœud est aussi employé pour désigner un des membres d'un groupe de serveurs informatiques constituant un cluster ou une grille.

OS

(Operating System) Système d'exploitation. En exemple : Windows, Solaris, Linux

Replica

Réplique d'une information, une base de données par exemple.

16“ITILV3_Glossary_French_v3_2007.doc,” 25.

Reverse proxy

Service qui permet l'acheminement de flux clients vers plusieurs serveurs habituellement cachés derrière ce reverse proxy.

RID

(Record ID) Adresse d'une entrée dans une base OrientDB. Elle est formée comme ceci :

#cluster-id:position-dans-le-cluster

Scalabilité

Anglicisme du terme scalability : capacité pour un service de supporter la montée en charge par l'ajout de ressources physiques et/ou logicielles sur un nœud existant (scalabilité verticale) ou nouveau (scalabilité horizontale).

SI

Système d'Information

SGBD

(Système de Gestion de Base de Données) Logiciel qui gère et maintient une base de données en garantissant la sécurité, la confidentialité, la concurrence et la cohérence des données.

Sharding

Partitionnement horizontal : désigne la possibilité de morceler une classe de données sur plusieurs nœuds en environnement distribué.

Standalone

Un serveur standalone délivre un service seul. Il n'est donc pas inclus dans une architecture distribuée.

Théorie des graphes

Théorie informatique et mathématique ayant de très nombreuses applications comme les réseaux de transports et de communication. Des systèmes sont modélisés pour décrire un état ou résoudre un problème.

TinkerPop

Application libre de gestion de graphes qui fournit un framework et des API.

Trigger

Concept lié aux SGBD relationnels. Il déclenche une action à chaque fois qu'un événement pour lequel il a été configuré arrive, par exemple un INSERT de données dans une table ou un ALTER d'une table.

Worker

Serveur/nœud esclave dans un cluster qui exécute des tâches, des calculs.

10 Bibliographie

“22.6. Caches in Neo4j - - The Neo4j Manual v2.1.7.” Accessed March 21, 2015. http://neo4j.com/docs/stable/configuration-caches.html.

“30.5. Contributing Code to Neo4j - - The Neo4j Manual v2.1.7.” Accessed March 1, 2015. http://neo4j.com/docs/2.1.7/community-contributing-code.html.

“(51) Neo4j - Google Groups.” Accessed March 14, 2015. https://groups.google.com/forum/#!forum/neo4j.

“Caching | OrientDB Manual.” Accessed March 21, 2015. http://www.orientechnologies.com/docs/2.0/orientdb.wiki/Caching.html.

“Combining Neo4J and Hadoop (part I) | Xebia Blog.” Accessed March 26, 2015. http://blog.xebia.com/2012/11/13/combining-neo4j-and-hadoop-part-i/.

“Commands | OrientDB Manual - Version 2.0.” Accessed March 31, 2015. http://www.orientechnologies.com/docs/last/Commands.html.

“Cypher Query Language - - The Neo4j Manual v2.1.7.” Accessed March 31, 2015. http://neo4j.com/docs/2.1.7/cypher-query-lang.html.

“Distributed Architecture | OrientDB Manual - Version 2.0.” Accessed March 14, 2015. http://www.orientechnologies.com/docs/last/Distributed-Architecture.html.

“GremlinDocs.” Accessed March 31, 2015. http://gremlindocs.com/.

“ITILV3_Glossary_French_v3_2007.doc.” Accessed February 21, 2015. http://www.best-management-practice.com/gempdf/ITILV3_Glossary_French_v3_2007.doc.

Miyuru, Dayarathna; Toyotaro, Suzumura, Tokyo Institute of Technology. “Graph Database Benchmarking on Cloud Environments with XGDBench.” Accessed March 9, 2015. https://docs.google.com/viewer?a=v&pid=sites&srcid=ZGVmYXVsdGRvbWFpbnx0b2t5b3RlY2hzdXp1bXVyYWxhYmVuZ3xneDoyMGRiOGFlM2Y2OGY5Mzhj.

“Modelling with Graphs | SkillsCast | 31st August 2011.” Accessed February 21, 2015. https://skillsmatter.com/skillscasts/2569-graph-modelling.

“NOSQL Databases.” Accessed February 25, 2015. http://nosql-database.org/index.html.

“NoSQL or NoREL? A Short Account of Taxonomic Development - DATAVERSITY.” Accessed February 25, 2015. http://www.dataversity.net/nosql-or-norel-a-short-account-of-taxonomic-development/.

“NoSQL Relational Database Management System: Commands at a Glance.” Accessed February 26, 2015. http://www.strozzi.it/cgi-bin/CSA/tw7/I/en_US/NoSQL/Commands%20at%20a%20glance.

“org.neo4j.graphdb.event (Neo4j Community 2.0.3 API).” Accessed March 15, 2015. http://neo4j.com/api_docs//current/org/neo4j/graphdb/event/package-summary.html.